Source Listing for project tut1

Ulrich Mutze www.ulrichmutze.de

July 1, 2020

Contents

1 Introduction 1

2 alloca.h 2

3 cpmangle.h 3

4 cpmangle.cpp 9

5 cpmbas.h 17

6 cpmbasicinterfaces.h 18

7 cpmbasictypes.h 20

8 cpmc.h 22

9 cpmc.cpp 30

10 cpmcompdef.h 37

11 cpmdefinitions.h 40

12 cpmf.h 41

13 cpmfa.h 51

14 cpmfl.h 61

15 cpmfo.h 84

16 cpmfr.h **88**

17 cpmgreg.h **96**

18 cpmgreg.cpp **102**

19 cpminterfaces.h **110**

20 cpmm.h **120**

21 cpmmacros.h **131**

22 cpmmpi.h **134**

23 cpmnumbers.h **141**

24 cpmnumbers.cpp **177**

25 cpmp.h **185**

26 cpms.h **193**

27 cpmsr.h **208**

28 cpmsystem.h **215**

29 cpmsystem.cpp **230**

30 cpmsystemdependencies.h **246**

31 cpmtypes.h **248**

32 cpmtypes.cpp **258**

33 cpmuc.h **268**

34 cpmuc.cpp **273**

35 cpmv.h **274**

36 cpmv.cpp **329**

37 cpmva.h **331**

38 cpmviewport.h **344**

39 cpmviewport.cpp **355**

40 cpmvo.h **368**

41 cpmvr.h **383**

42 cpmword.h **387**

43 cpmword.cpp **402**

44 cpmx.h **413**

45 cpmzinterval.h **435**

46 cpmzinterval.cpp **444**

47 features.h **450**

48 mpfr.h **460**

49 stdio.h **483**

50 stdlib.h **501**

51 time.h **522**

52 tut1.cpp **529**

53 survey.txt **540**

54 headerdependencies.txt **550**

1 Introduction

Project tut1 is the first project treated in the C+− tutorial at `www.ulrichmutze.de/softwaredescriptions/tut.pdf`. It deals with the simplest type of program supported by C+− , a console program without graphical output. The topic under consideration is reference counting in arrays. We compare execution time of copy-construction for C+− arrays and for std::vector. The longer the arrays are the larger the factor by which `std::vector` takes longer than `CpmArrays::V`.

The present version of code assumes C++11 and should compile without warnings under the option `-pedantic`. Since about three years all my C++ work is being

done under Linux (actually Ubuntu) so that all remaining Windows-related remarks may not be up to date.

2 alloca.h

```
/* Copyright (C) 1992-2018 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   The GNU C Library is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
   Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public
   License along with the GNU C Library; if not, see
   <http://www.gnu.org/licenses/>.  */

#ifndef _ALLOCA_H
#define _ALLOCA_H 1

#include <features.h>

#define __need_size_t
#include <stddef.h>

__BEGIN_DECLS

/* Remove any previous definitions.  */
#undef alloca

/* Allocate a block that will be freed when the calling function exits.  */
extern void *alloca (size_t __size) __THROW;

#ifdef __GNUC__
# define alloca(size) __builtin_alloca (size)
#endif /* GCC.  */

__END_DECLS

#endif /* alloca.h */
```

3 cpmangle.h

```
//? cpmangle.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_ANGLE_H_
#define CPM_ANGLE_H_
////////////////////////////////////////////////////////////////////////
// Description: A class describing angles and the 1-dimensional
//   torus
////////////////////////////////////////////////////////////////////////
#include <cpmc.h>
#include <cpmtypes.h>
#include <cpmv.h>

namespace CpmGeo{

   using CpmRoot::R;
   using CpmRoot::Z;
   using CpmRoot::Word;
   using CpmRoot::C;
   using CpmArrays::V;

enum AngleUnit { CYCLE, HOUR, DEG, RAD};
   // cycle (full angle), hour, degree, radian
enum AngleRange { POS, NEG};
   // always positive, negative allowed
enum AngleHex {DEC, MIN, SEC};
   // full decimal, minutes, minutes and seconds
// friend functions need to be declared as belonging to the
// present namespace
class Angle;
Angle mean(const CpmArrays::V<Angle>& phi);
Angle mean(const CpmArrays::V<Angle>& angles,
   const CpmArrays::V<R>& weights);
```

```
Angle RealToAngle(R x, AngleUnit u, AngleHex h);
Angle realToAngle(R c);

class Angle{ // angles
   // An instance of Angle is an angle, the manifold of all angles is
   // a 1-dimensional torus. There are two interpretations.
   // (i) Angle as a coordinate on a circle (as a curve) and
   // (ii) angle as a descriptor of a rotation arround a given axis.
   // Both are closely related due to the fact, that the rotations act
   // transitively and effectively on a circle. For defining a proper set
   // of methods we have not to make a choice in favor of one of these
   // interpretations.
   // Interpretation (i) endows Angle most directly with the structure of
   // an oriented metric space, and (ii) with the structure of an
   // Abelian group.

   static void normalizeAngle(R& a0);
      // changes a0 into a value in the interval (-Pi,Pi] by adding
      // a multiples of 2*Pi

   R a_; // is read as alpha, is the measure of the angle as arc length
      // on the unit circle (for effective usage of built-in
      // trigonometric functions. The range is by definition (-Pi,Pi].

   void normalize(void){ normalizeAngle(a_);}

public:
   typedef Angle Type;
   typedef R ScalarType;

   CPM_IO
   CPM_ORDER

   CPM_SUM_PLAIN
   CPM_DIFFERENCE_PLAIN
      // Addition and subtraction among angles as borrowed from the
      // mathematical model by residue classes in R 'modulo 2*Pi'.
      // A further mathematical model is given by
      // { z \in C : |z|=1 }. In  this model addition of angles corresponds
      // to multiplication of complex numbers and subtraction of angles
      // corresponds  to division of numbers. If we are simply given a
      // torus (e.g. as a thin ring of polished metal) then we need to mark
      // on it a point as origin and an arrow as direction in order map it
      // to any of the mathematical models considered here.
      // This implies that the algebraic structure of an abelian group
      // exhibited by both mathematical models is not inherent in the
      // geometry of the pure torus but depends on the selection of an
      // origin and a direction (orientation) as indicated above. Obviously
      // the selected origin plays the role of the neutral element of the
      // abelian group. Notice that the pure torus is a metric space
```

```
      // where the distance between two points is the length of the
      // curve along the torus (i.e. the shortest arc) which connects the
      // points.

   CPM_SCALAR_M
   // The meaning of multiplication with real numbers derives by
   // continuity from the meaning of multiplication by rational
   // numbers. Let us consider first multiplication by a natural
   // number n. The proper definition follows from the group
   // structure: alpha*n := alpha + ... + alpha (n terms).
   // Multiplication by 1/m for natural number:
   // alpha*(1/m) := smallest angle beta for which beta*m=Angle(),
   // where Angle() is the zero angle which is the unit element of
   // the group. Smallness, obviously, refers to the distance to
   // the unit element. Obviously, one may add (2*Pi)/m to beta
   // (let the result be called beta') and one also has
   // beta'*m=Angle().

   CPM_CONJUGATION
      // An idempotent conjugation operation. If the angle is interpreted
      // as the phase angle of a complex number, this corresponds to
      // the complex conjugation of the number. This is a natural
      // geometric operations for angles, resulting from the distinction
      // of one angle as the zero-angle (origin on torus).

   Word nameOf()const{ return Word("Angle");}

   R operator /(const Angle&)const;
      // alpha/beta is the number r for which beta*r=alpha.
      // Notice the comment to ScalarType on multiplying
      // Angles by numbers. This is not defined in residue
      // classes modulo 2*Pi.
      // Products of angles don't give angles; they are
      // omitted here.

   Angle operator+(R shift)const{ return Angle(a_+shift,false);}
      // Returning the angle which results from *this by adding an arc
      // which may be any length (e.g. much larger than 2*Pi) and sign.
      // This gives Angle the structure of an affine space.

// constructors

   Angle(void):a_(0.){}
      // angle zero

   explicit Angle(R angle, bool deg=true);
      // Constructor for Angles out of real numbers. The first argument
      // is always interpreted as an angle in degrees, unless deg==0
      // Never enable automatic conversion. Would result in ambiguous
      // arithmetics. Input needs not to be normalized.
```

```
   Angle(R vAngle, AngleUnit au);
      // creates an angle which is vAngle*au

   explicit Angle(C const& z):a_(z.arg()){}
      // Constructs the phase angle (polar angle) of z

   C expi()const{ return C(1.,a_,"polar");}
      // Returns the complex number exp(i*a). Thus Angle now has a
      // bi-directional efficient interface to C.

// automatic conversion in one direction Angle-->R is OK

   operator R(void)const{return a_;}
   // cast to R for making trigonometric functions defined for angles
   // No longer used in the implementation of Angle.

   R toR()const{ return a_;}
   // Explicit conversion in a standard manner. The implementation of
   // Angle now relies on this.

   R cut( R c)const;
   //: cut
   // Returns a value in the range (c*degree-2*Pi,c*degree] which from
   // a_ is derived by adding or subtracting a multiple of 2*Pi.
   // To have such a facility turned out to be very useful in
   // dealing with angular motion of stepper motors.

   void qun_(Z n);
      //: quantize
      // Changes angle *this into the nearest of n equi-spaced
      // angles which are equi-distributed over the whole range.
      // For n<1, no action.

   R toDeg(Z i=0)const;
      //: to degrees
      // Returns the angle *this in degrees. For the default
      // value 0 of the argument, the range is the standard range
      // (-180,180], and for 1 the range is [0,360).

   Word toWord(AngleUnit u, AngleRange r, AngleHex h, Z digits=1)const;
      // Returns a string which gives the angle in the units
      // given by u, in the range indicated by r, in a hexagesimal
      // subdivision characterized by h and with a number of figures
      // after the decimal point is given by digits. If digits is -1,
      // there is one figure behind the decimal point and this is rounded
      // to 0 or 5 corresponding to the accuracy with which the hour
      // circle of my C5 telescope can be read. The meaning of the
      // parameters follows from the explanation to the
      // enumerations AngleUnit, AngleRange, and AngleHex.
```

```
   friend Word AngleToWord(const Angle& alpha,
      AngleUnit u, AngleRange r, AngleHex h, Z digits=1)
      { return alpha.toWord(u,r,h,digits);}

// modifying an angle
   void setArc(R alpha);
      // input is an arc (i.e. something like a, but need not be
      // normalized)

// absolute value
   R abs(void)const{ return (a_>=0. ?  a_ : -a_);}
      // absolute value: positive arc as a number (distance from zero
      // angle)

   R dis(const Angle& phi)const;
      //: distance
      // Lets angles form a metric space. Is symmetric in the arguments.
      // Implementation corrected 2016-03-25 (symmetry was not guaranteed
      // before)

   bool isPos()const{ return a_>=0;}
      // returns true if the angle is in quadrant 1 or 2 (by definition,
      // 0 belongs to quadrant 1, Pi to quadrant 2 ) and false else

   void pos(){ if (a_<0) a_=-a_;}
      // makes the angle positive by conjugation

// mean values taking into account the embedding in the complex numbers
   friend Angle mean(const CpmArrays::V<Angle>& phi);
      // mean of an array of angles.
      // Polar angle of exp(i*phi[1])+...+exp(i*phi[n]).
      // There seems to be no easy way to define this on R/~
      // where a1 ~ a2 <==> a1 = a2 (mod 2Pi)

   friend Angle mean(const CpmArrays::V<Angle>& angles,
      const CpmArrays::V<R>& weights);
      // weighted mean of an array of angles
      // Polar angle of
      // weights[1]*exp(i*phi[1])+...+weights[n]*exp(i*phi[n])

   friend Angle RealToAngle(R x, AngleUnit u, AngleHex h);
      // Returns an angle alpha which is given by a real number
      // according the format indicated by the unit u and the
      // subdivision schema indicated by h.

   friend Angle realToAngle(R c){ return Angle(c);}
      // Converts a real number, interpreted as an arc into a
      // instance of class Angle.
```

```
   static V<R> smoothAngleList(const V<Angle>& va);
      // Given a list va of Angles, we return a list res of corresponding
      // R's such that
      //    res[i]==va[i] mod(2*Pi)
      // where adding or subtracting of 2*Pi is done such that the
      // resulting list is as smooth as possible. More precisely we
      // assure that for each i
      // | res[i+i]-res[i]| < |res[i+1]+2*Pi-res[i]|
      // and
      // | res[i+i]-res[i]| < |res[i+1]-2*Pi-res[i]|
      // As a result of this the res[i] may grow to much larger values
      // than 2*Pi as in a list of successive angular positions of a
      // rotating wheel.

    static V<R> smoothList(const V<R>& vr);
      // Similar to previous function. The argument list is not a list
      // of angles but a list of their conversions to numbers (based
      // on the RAD unit).

    static Angle deg1, deg90, deg180, deg270, deg360;
};

} // namespace

#endif
```

4 cpmangle.cpp

```
//? cpmangle.cpp
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


//Description: see cpmangle.h

#include <cpmangle.h>

using namespace CpmStd;

using namespace CpmGeo;
using CpmRoot::R;
using CpmRoot::Z;
using CpmRoot::C;
using CpmRoot::Word;

namespace{
   const R fullAngle=cpmpi*2;
   const R iFullAngle=R(1.)/fullAngle;
   const R hour=fullAngle/24;
}

Angle Angle::deg1(1,DEG);
Angle Angle::deg90(90,DEG);
Angle Angle::deg180(180,DEG);
Angle Angle::deg270(270,DEG);
Angle Angle::deg360(360,DEG);

void Angle::normalizeAngle(R& a)
{
   if (a>-cpmpi && a<=cpmpi) return;
      // then no normalization needed. Introduced 2006-03-29
      // after it was observed that normalizing the zero-angle
      // caused difficulties
```

```
   a*=iFullAngle;
   a-=cpmfloor(a);
   if (a>0.5) a-=1;
   a*=fullAngle;
}

Angle::Angle(R angle, bool deg):a_(angle)
{
   if (deg) a_*=cpmdeg;
   normalize();
}

Angle::Angle(R angle, AngleUnit au):a_(angle)
{
   if (au==RAD){
      ;
   }
   else if (au==DEG){
      a_*=cpmdeg;
   }
   else if (au==CYCLE){
      a_*=fullAngle;
   }
   else if (au==HOUR){
      a_*=hour;
   }
   else{
      cpmerror("Angle(R, AngleUnit): unvalid AngleUnit");
   }
   normalize();
}

void Angle::setArc(R alpha)
{
   a_=alpha;
   normalize();
}

namespace{
void quantize(R& x, Z n)
{
   R y=x*n;
   y=cpmfloor(y+0.5);
   x=y/n;
}

}
Angle& Angle::operator +=(const R& s)
{
   a_+=s;
```

```
   normalize();
   return *this;
}

Angle& Angle::operator *=(const R& s)
{
   a_*=s;
   normalize();
   return *this;
}

Angle Angle::operator -(void)const
{
   Angle res;
   res.a_=-a_;
   res.normalize();
   return res;
}

Angle& Angle::operator +=(const Angle& s )
{
   a_+=(s.a_);
   normalize();
   return *this;
}

Angle& Angle::operator -=(const Angle& a1 )
{
   a_-=a1.a_;
   normalize();
   return *this;
}

Angle Angle::operator +(const Angle& a1)const
{
   return Angle(a_+a1.a_,RAD);
}

Angle Angle::operator -(const Angle& a1)const
{
   return Angle(a_-a1.a_,RAD);
}

R Angle::operator /(const Angle& a1)const
{
   R a1Inv=cpminv(a1.a_);
   return a_*a1Inv;
}

Angle Angle::con(void)const
```

```
{
   if (a_==cpmpi) return *this;
   else return Angle(-a_,RAD);
}

R Angle::cut(R c)const
{
   R lim=c*cpmdeg;
   R ar=a_+fullAngle;
   if (ar>lim) ar-=fullAngle;
   return ar;
}

bool Angle::prnOn(ostream& out)const
{
   return CpmRoot::write(a_,out);
}

bool Angle::scanFrom(istream& in)
{
   bool res=CpmRoot::read(a_,in);
   normalize();
   return res;
}

Angle CpmGeo::mean(const CpmArrays::V<Angle>& angles)
{
   Z i,n=angles.dim();
   Word loc("Angle CpmGeo::mean(const CpmArrays::V<Angle>& angles)");
   cpmassert(n>0,loc);
   C z,sum(0,0);
   for (i=1;i<=n;i++){
      z.polar(1.,angles[i].toR());
      sum+=z;
   }
   R r,phi;
   sum.toPolar(r,phi);
   return Angle(phi,RAD);
}

Angle CpmGeo::mean(const CpmArrays::V<Angle>& angles,
   const CpmArrays::V<R>& weights)
{
   Word loc("Angle CpmGeo::mean(...)");
   Z i,n=angles.dim(),n1=weights.dim();
   cpmassert(n>0,loc);
   cpmassert(n1>=n,loc);
   C z,sum(0,0);
   for (i=1;i<=n;i++){
      z.polar(1.,angles[i].toR());
```

```
      sum+=z*weights[i];
   }
   R r,phi;
   sum.toPolar(r,phi);
   return Angle(phi,RAD);
}

Z Angle::com(const Angle& a1)const
{
   if (a_<a1.a_) return 1;
   else if (a_>a1.a_) return -1;
   else return 0;
}

R Angle::dis(const Angle& phi)const
{
   Angle a1=*this - phi;
   Angle a2= phi - *this;
   R d1=cpmabs(a1.toR());
   R d2=cpmabs(a2.toR());
   return cpminf(d1,d2)*iFullAngle;
}

R Angle::toDeg(Z i)const // 'to degrees'
   {
      static R iDegree=R(180)/cpmpi;
      R b=a_*iDegree;
      if (i==1){
         if (b<0.) b+=360.;
      }
      return b;
   }

Word Angle::toWord(AngleUnit u, AngleRange r, AngleHex h, Z digits)const
{
   const R iDegree=R(180)/cpmpi;
   const R ihour=R(12)/cpmpi;
   R y=a_;
   if (r==POS && y<0) y+=fullAngle;
    Z flag=0;
   Word res, sep=":";
   const char* ff="%-#3.1f";
   const char* fff="%2d";
   if (digits==0)  ff="%-#2.0f";
   if (digits==1)  ff="%-#3.1f";
   if (digits==-1){ff="%-#3.1f"; flag=1;}
   if (digits==2)  ff="%-#4.2f";
   if (digits==3)  ff="%-#5.3f";
   if (digits==4)  ff="%-#6.4f";
   if (digits==5)  ff="%-#7.5f";
```

```
   if (digits==6)  ff="%-#8.6f";

   if (y<0){
      res="-";  // this is the first part of the result
                // further parts will be appended later
      y=-y;
   }
   else{
      res=" ";
   }

   if (u==CYCLE){
      y*=iFullAngle; // positive angle in full angles (turns)
   }
    else if (u==HOUR){
      y*=ihour; // positive angle in hours
   }
   else if (u==DEG){
      y*=iDegree; // positive angle in degrees
   }

   if (h==DEC){
      if (flag) quantize(y,2);
      res=res&CpmRoot::toWord(y,ff);
      return res;
   }

   Z ip=cpmtoz(y); // integer part (degrees or hours depending on u)
   res=res&CpmRoot::toWord(ip,fff);
   y-=ip;
   y*=60;  // y=minutes (time or angle depending on u)
   if (h==MIN){
      if (flag) quantize(y,2);
      res=res&sep&CpmRoot::toWord(y,ff);
      return res;
   }

   ip=cpmtoz(y); // integer part (minutes of time or angle)
   res=res&sep&CpmRoot::toWord(ip,fff);
   y-=ip;
   y*=60; // y=seconds (time or angle depending on u)
   if (h==SEC){
      if (flag) quantize(y,2);
      res=res&sep&CpmRoot::toWord(y,ff);
      return res;
   }
   return res;  // this should not be reached, for symmetry and for
      // avoiding a warning
}
```

```
Angle CpmGeo::RealToAngle(R x, AngleUnit u, AngleHex h)
{
   const R i60=1./60.;
   Z sign;
   Angle res;
   if (h==DEC){
      if (u==CYCLE) x*=fullAngle;
      if (u==HOUR) x*=hour;
      if (u==DEG) x*=cpmdeg;
      Angle::normalizeAngle(x);
      res.a_=x;
      return res;
   }
   if (x<0.){
      sign=-1; x=-x;
   }
   else{
      sign=1;
   } // now x is positive
   R ix=cpmfloor(x);
   R y=(x-ix)*100;
   if (h==MIN){
      x=ix+y*i60;
      if (u==CYCLE) x*=fullAngle;
      if (u==HOUR) x*=hour;
      if (u==DEG) x*=cpmdeg;
        x*=sign;
      Angle::normalizeAngle(x);
      res.a_=x;
      return res;
   }

   R iy=cpmfloor(y);
   R z=(y-iy)*100;
   if (h==SEC){
      x=ix+(iy+z*i60)*i60;
      if (u==CYCLE) x*=fullAngle;
      if (u==HOUR) x*=hour;
      if (u==DEG) x*=cpmdeg;
      x*=sign;
      Angle::normalizeAngle(x);
      res.a_=x;
      return res;
   }
   return res;
}

V<R> Angle::smoothAngleList(const V<Angle>& va)
{
   Z i,n=va.dim();
```

```
   V<R> res(n);
   for (i=1;i<=n;i++) res[i]=va[i].toR();
   for (i=2;i<=n;i++){
      R diff=res[i]-res[i-1];
      R dz=cpmabs(diff);
      if (dz<cpmpi) continue;
      R dp=cpmabs(diff+fullAngle);
      R dm=cpmabs(diff-fullAngle); //p,z,m stands for plus, zero, minus
      R dMin=cpminf(dp,dz,dm);
      if (dp==dMin) res[i]+=fullAngle;
      else if (dm==dMin) res[i]-=fullAngle;
      else ;
   }
   return res;
}

V<R> Angle::smoothList(const V<R>& vr)
{
   Z i,n=vr.dim();
   V<R> res=vr;
   for (i=2;i<=n;i++){
      R ri=res[i];
      R rim=res[i-1];
      R diff=ri-rim;
      R dz=cpmabs(diff);
      if (dz<cpmpi) continue;
      R dp=cpmabs(diff+fullAngle);
      R dm=cpmabs(diff-fullAngle); //p,z,m stands for plus, zero, minus
      R dMin=cpminf(dp,dz,dm);
      if (dp==dMin){
         res[i]+=fullAngle;
      }
      else if (dm==dMin){
         res[i]-=fullAngle;
      }
      else ;
   }
   return res;
}

void Angle::qun_(Z n)
{
   if (n<1) return; // do nothing for meaningless input
   R a=(a_+cpmpi)*iFullAngle*n; // ia a value between 0 and n
   a=cpmrnd(a);
   a*=(fullAngle/n);
   a_=a-cpmpi;
}
```

5 cpmbas.h

```
//? cpmbas.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_BAS_H_
#define CPM_BAS_H_
/*
   Description: Collects header files from cpm0/include in one file
   in order to make working with basic C+- easier
*/
   #include <cpmtypes.h>
   #include <cpmfr.h>
   #include <cpmvr.h>
   #include <cpmsr.h>
   #include <cpmm.h>
   #include <cpmp.h>
   #include <cpmc.h>
   #include <cpmangle.h>
   #include <cpmgreg.h>
#endif
```

6 cpmbasicinterfaces.h

```
//? cpmbasicinterfaces.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_BASIC_INTERFACES_H_
#define CPM_BASIC_INTERFACES_H_
/*

   Description: see cpminterfaces.h

*/

// The following makes sure that no client class will use Type in a way
// that would need copy constructor and assignment.
// Should be placed in the private section

#include <cpmbasictypes.h>
   // for Z in CPM_ORDER

#define CPM_INVAR(TypeName)\
   Type& operator = (Type const&);\
   TypeName(Type const&);

// example for usage
/*
   template <class X, class Y>
   class SUMFUO: public ...{
      typedef SUMFUO<X,Y> Type;
      CPM_INVAR(SUMFUO)
   public:
      ...
   };
*/
```

```
// order related stuff

#define CPM_ORDER_PLAIN\
   bool operator == ( Type const& x)const;\
   bool operator != (Type const& x)const;\
   bool operator < (Type const& x)const;\
   bool operator > (Type const& x)const;\
   bool operator <= (Type const& x)const;\
   bool operator >= (Type const& x)const;

// consistent generation of all six order-related operators from a
// single member functions. Notice that it would be useless to make com
// virtual since any useful re-definition probably differs in the
// type of the second argument too.

// see cpmnumers.h for explanations to 'com'

#define CPM_ORDER\
   CpmRoot::Z com(Type const&)const;\
   bool equalTo(Type const& x)const{ return com(x)==0;}\
   bool priorTo(Type const& x)const{ return com(x)==1;}\
   bool operator == ( Type const& x)const\
      { return com(x)==0;}\
   bool operator != (Type const& x)const\
      { return com(x)!=0;}\
   bool operator < (Type const& x)const\
      { return com(x)>0;}\
   bool operator > (Type const& x)const\
      { return com(x)<0;}\
   bool operator <= (Type const& x)const\
      { return com(x)>=0;}\
   bool operator >= (Type const& x)const\
      { return com(x)<=0;}

#endif
```

7 cpmbasictypes.h

```
//? cpmbasictypes.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_BASIC_TYPES_H_
#define CPM_BASIC_TYPES_H_
/*
   Description: Introduces mathematics style named
      aliases for basic integer number types.
*/
#include <cstddef> // for std::ptrdiff_t and std::size_t
#include <cpmdefinitions.h> // for CPM_LONG and CPM_MP

namespace CpmRoot{
   // Basic number-related C+- types and functions.

// This defines what types are to be used to represent
// C+-'s notion of integer numbers. Floating point types
// will be treated in cpmnumbers.h

// Integer types L, N, Z.
// Remember that N and Z are standard names for the sets of
// natural numbers and integer numbers in mathematics.

#if defined(CPM_LONG) || defined(CPM_MP) || defined(CPM_MPREAL) // take the
// largest possible data types
   typedef long int Z;
      // integer numbers
   typedef unsigned long int N;
      // natural numbers
      // provides cyclic definition of addition, without overflow.
#else // take normal int and unsigned
   typedef int Z;
      // integer numbers
```

```
   typedef unsigned int N;
      // natural numbers
      // Type N will be used only in implementation code,
      // where it is important that addition is a cyclic operation
      // which never creates overflow. As type of function
      // arguments I don't use N. The reason is the following:
      // Function arguments for which Z or N would be appropriate types
      // occur frequently. Often, in cases in which N seemed the natural
      // choice when defining a function in the first place, it has to
      // be discovered that an extension to Z could be useful. So one
      // would be tempted so switch (or oscillate!) between using Z and N.
      // This is to be avoided!
#endif // CPM_LONG || defined(CPM_MP)

   typedef unsigned char L;
      // 'L' for 'letter' represents a byte which after 'integral
      // promotion' is a value between 0 and 255.
      // Mainly needed for storing pixel values in image matrices.
      // Gets written to files as a number \in {0,1,...,255}
} // CpmRoot

#endif // CPM_BASIC_TYPES_H_
```

8 cpmc.h

```
//? cpmc.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_C_H_
#define CPM_C_H_
/*
   Description: Declaration of a class of complex numbers
       using the system provided type <complex> for implementation was
       tried. But this class has only very limited ammount of
       functions and my component access via [] can't be implemented
       here since the components are private.
       Division by zero will write a message, return 0 and continue.
       Test_c<C> d(10,1,1) gave a result which ~10-7 (including read
       write difference which is OK.
*/

#include <cpmword.h>
#include <complex>
   // connection with std complex numbers is useful when interfacing
   // with the Eigen library.

namespace CpmRoot{

class C;
// These are declarations of functions in scope CpmRoot
// most of them have a friend declaration within class C
// which enables a more efficient implementation.
R arg(C const& z);
C log10(C const& z);
C pow(C const& z, Z const& n);
C pow(C const& z, C const& n);
C sin(C const& z);
C cos(C const& z);
```

```
C tan(C const& z);
C cot(C const& z);
C sinh(C const& z);
C cosh(C const& z);
C tanh(C const& z);
C coth(C const& z);
C arcsin(C const& z);
C arccos(C const& z);
C arctan(C const& z);
C arccot(C const& z);
C arcsinh(C const& z);
C arccosh(C const& z);
C arctanh(C const& z);
C arccoth(C const& z);
C operator /(C const& z, R const& s);
bool isVal(C const& z);

class C { // complex numbers

   R re, im;
   typedef C Type;

public:

//// general infrastructure, in part with inline implementation
// This is essentially the general interface of templates Vr<> and Fr<>
// appart from the fact that here no function is virtual since we don't
// need to derive from this 'ultimate' class.

// order and comparison
   CPM_ORDER

// I/O operations
   CPM_IO

// test devices
   C ran(Z j=0)const;
// value of res.re is in the open interval (-|re|,|re|)
// value of res.im is in the open interval (-|im|,|im|)

   R dis(C const& y)const;
   C test(Z)const;
// descriptors
   Word nameOf(void)const{ return Word("C");}

   Word toWord()const;
      //: to word
      // for a nice printable representation

   Z hash(void) const;
```

```
      //: hash value

   C net(Z i=0)const{ if (i==1) return C(1.,0.); else return C();}
      //: neutrals
      // net(0): neutral element of addition,
      // net(1): neutral element of multiplication

   C neg()const{ return C(-re,-im);}
      //: negative

   C inv(void)const;
      //: inversion
      // Returns C(0,0) upon division by zero, but creates
      // warnings on cpmcerr if this happens. The total
      // number of warnings that can arrise in this way is limited
      // so that this file can't grow too much by this mechanism.

   C operator !(void)const{ return inv();}

// complex conjugation
   C con(void)const{ return C(re,-im);}
   void con_(){im=-im;}

   C operator~(void)const{ return C(re,-im);}
   //: ~
   // instead of z.con() we may write ~z

   C operator|(C const& z)const{ return con()*z;}
   //: (|) scalar product
   // The combination ~z1*z2 can also be written as (z1|z2)

// more specific topics

   static C I;  // 'imaginary unit'
   static C one; // number one = C(1,0)
   static C Ibar; // conjugate of I = - I = C(0,-1)

   C(void):re(0.),im(0.){}

   explicit C(R const& u, R const& v=0. ):re(u),im(v){}
      // constructor from real and imaginary part
      // no automatic conversion from R to C

   explicit C(std::complex<R> const& z):re(z.real()),im(z.imag()){}
      // construction from std::complex

   C(C const& z):re(z.re),im(z.im){}
      // standard construction operator

   C(R r, R phi, Word w):
```

```
  re(r*cpmcos(phi)),im(r*cpmsin(phi)){}
     // constructor from polar coordinates
     // In calling the constructor it is helpful to
     // use w="polar"

  R real(void)const{ return re;}

  R imag(void)const{ return im;}

  std::complex<R> std()const{ return std::complex<R>{re,im};}

  C& operator=(C const& c){ re=c.re; im=c.im; return *this;}
     // standard assignment operator

  C& operator=(std::complex<R> const& c)
  { re=c.real(); im=c.imag(); return *this;}

  Z dim(void)const{ return 2;} // dimension as a real linear space
     //: dimension

  R arg(void)const{ return cpmarg(re,im);}
     //: argument
     // The value is in the interval (-Pi,Pi] independent
     // of the convention followed by the system's ::atan2 function

//  R abs(bool careful=true)const
// Warning: not clear whether this works for choices of
// R different from float, double, long double.
R abs(bool careful=true)const
  { return  careful ? std::abs(std::complex<R>(re,im)) :
     cpmsqrt(re*re+im*im);}
     //: rho
     // common symbol for the polar radius

  R absFast(void)const
  { return cpmsqrt(re*re+im*im);}
     //: rho fast
     // No provision against overflow by sqaring potentially large
     // numbers.

  R absSqr(void)const{ return re*re+im*im;}
     //: absolute (value) squared

  R abs2(void)const{ return re*re+im*im;}
     //: absolute (value) squared

  bool isVal(void)const{ return cpmiva(re)&&cpmiva(im);}
     //: is valid

  C scaleTo(R const& r)const;
```

```
      // returns a complex number of absolute value |r| and the direction
      // of *this for positive r and antidirection for negative r.

   C timesI(void)const{ return C(-im,re); }

   C dividedByI(void)const{ return C(im,-re);}

   C dividedBy2I(void)const{ return C(im*0.5,re*(-0.5));}

   void pol_(R r, R phi);
      //: polar
      // changes *this into r*exp(i*phi)
      // 'modern' indication of mutating nature of the function

   static C pol(R r, R phi){C z; z.pol_(r,phi); return z;}
      //: polar
      // construction from polar coordinates as a static
      // function instead of a constructor

   void pol_1_2(R& r, R& phi)const{ r=abs(); phi=arg();}
      //: polar
      // 'modern' indication of reference nature of first
      // and second argument

// heritage versions of functions related to polar coordinates
   void polar(R const& r, R const& phi){ pol_(r,phi);}
      // changes *this into r*exp(i*phi)
      // not conforming to the C+- naming convention

   void toPolar(R& r, R& phi)const;
      // after call r and phi have the values radius() and arg()

// end of heritage versions of functions related to polar coordinates

   R nor_(R r=1)
      //: normalize
   {
      R a=abs();
      if(a!=0){R fac=r/a;re*=fac;im*=fac;}
      else{re=r;}
      return a;
   }
      // turns *this into a C of length r and returns the norm of
      // the original in order not to loose information. If *this was
      // zero, it will be turned into  a 'standard C'
      // of length r.

   C sqrt()const;
   C sqr()const;
   C exp()const;
```

```
   C ln()const;

   C pow(C const& p)const;
   C sin()const;
   C cos()const;
   C tan()const;
   C cot()const;
   C sinh()const;
   C cosh()const;
   C tanh()const;
   C coth()const;
   C arcsin()const;
   C arccos()const;
   C arctan()const;
   C arccot()const;
   C arcsinh()const;
   C arccosh()const;
   C arctanh()const;
   C arccoth()const; // these added 2005-08-23

   friend R arg(C const& z);
   friend R rho(C const& z);
   static C expi(R phi){ return C(cpmcos(phi),cpmsin(phi));}
   friend C log10(C const& z);
   friend C pow(C const& z, const Z& n);
   friend C pow(C const& z, C const& n);
   friend C sin(C const& z);
   friend C cos(C const& z);
   friend C tan(C const& z);
   friend C cot(C const& z);
   friend C sinh(C const& z);
   friend C cosh(C const& z);
   friend C tanh(C const& z);
   friend C coth(C const& z);
   friend C arcsin(C const& z);
   friend C arccos(C const& z);
   friend C arctan(C const& z);
   friend C arccot(C const& z);
   friend C arcsinh(C const& z);
   friend C arccosh(C const& z);
   friend C arctanh(C const& z);
   friend C arccoth(C const& z);

// components for uniformity with arrays:  z.re=z[1], z.im=z[2]
// no exceptions !!!

   R& operator[](int i){ return i==2 ? im : re;}

   R const& operator[](int i)const{ return i==2 ? im : re;}
```

```
// mutating arithmetics
// There are a few more operations to be defined (such as C+R)
// which are not prepared in the interfaces; so everything
// is done explicitely
   C& operator +=(C const& x){re+=x.re; im+=x.im; return *this;}
   C& operator +=(R const& s){re+=s; return *this;}

   C& operator -=(C const& x){re-=x.re; im-=x.im; return *this;}
   C& operator -=(R const& s){re-=s; return *this;}

   C& operator *=(C const& x)
     {R re0=re; re=re*x.re-im*x.im; im=im*x.re+re0*x.im; return *this;}

   C& operator /=(C const& x){ return operator *=(x.inv());}

   C& operator *=(R const& s){re*=s; im*=s; return *this;}
   C& operator /=(R const& s);

// generating arithmetics
   C operator -(void)const{ return C(-re,-im);}
   C operator +(C const& z)const
      { return C(re+z.re,im+z.im);}
   friend C operator +(R const& s, C const& z2);
   C operator +(R const& s)const
      { return C(re+s,im);}
   C operator -(C const& z)const
      { return C(re-z.re,im-z.im);}
   friend C operator -(R const& s, C const& z2);
   C operator -(R const& s)const
      { return C(re-s,im);}
   C operator *(C const& z)const
      { return C(re*z.re-im*z.im,im*z.re+re*z.im);}
   friend C operator *(R const& s, C const& z2);
   C operator *(R const& s)const
       { return C(s*re,s*im);}
   C operator /(C const& z)const
      { return (*this)*z.inv();}
   friend C operator /(C const& z, R const& s);
};

// These are definitions of functions in scope CpmRoot, declared friend
// in class C.
inline C operator +(R const& s, C const& z2)
      { return C(z2.re+s,z2.im);}
inline C operator -(R const& s, C const& z2)
      { return C(s-z2.re,-z2.im);}
inline C operator *(R const& s, C const& z2)
       { return C(s*z2.re,s*z2.im);}
// The following functions need not to be friends of C.
// their argument in ways that need them to be declared friends of C.
```

```
inline bool isVal(C const& x){ return x.isVal();}
inline C ran(C const& x, Z j){ return x.ran(j);}
inline R dis(C const& x1, C const& x2){ return x1.dis(x2);}
inline C test(C const& x, Z cpl){ return x.test(cpl);}
inline Z hash(C const& x){ return x.hash();}
inline R abs(C const& x){ return x.abs();}
inline R rho(C const& x){ return x.abs();}
inline C net(C const& x, Z i=0){ return x.net(i);}
inline C inv(C const& x){ return x.inv();}
inline C con(C const& x){ return x.con();}
inline C sqrt(C const& z){ return z.sqrt();}
inline R Re(C const& z){return z.real();}
inline R Im(C const& z){return z.imag();}
inline C exp(C const& z){return z.exp();}
inline C expI(R phi){ return C::expi(phi);}
      // even for Angle alpha, we can write directly expI(alpha) since
      // CpmGeo::Angle gets automatically converted to an R-valued arc
      // So, instead of saying
      // C z; z.polar(r,phi);
      // we may, much nicer, say:
      // C z=r*expI(phi);
inline C ln(C const& z){ return z.ln();}
inline C log(C const& z){ return z.ln();}
inline C asin(C const& z){ return CpmRoot::arcsin(z);}
inline C acos(C const& z){ return CpmRoot::arccos(z);}
inline C atan(C const& z){ return CpmRoot::arctan(z);}
inline C acot(C const& z){ return CpmRoot::arccot(z);}
inline C asinh(C const& z){ return CpmRoot::arcsinh(z);}
inline C acosh(C const& z){ return CpmRoot::arccosh(z);}
inline C atanh(C const& z){ return CpmRoot::arctanh(z);}
inline C acoth(C const& z){ return CpmRoot::arccoth(z);}

} // namespace
#endif
```

9 cpmc.cpp

```
//? cpmc.cpp
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#include <cpmc.h>
#include <cpmtypes.h>

using namespace CpmRoot;
using namespace CpmSystem;

C CpmRoot::C::I=C(0.,1.);
C CpmRoot::C::Ibar=C(0.,-1.);
C CpmRoot::C::one=C(1.,0.);

Word C::toWord()const
      // for a nice printable representation
{
   Word sep_i=" "; // initial
   Word sep_f=" "; // final separator
   Word res;
   if (im==0.) res=cpmwrite(re);
   else if (re==0.){
      if (im>0.) res="i*"&cpmwrite(im);
      else res="-i*"&cpmwrite(-im);
   }
   else if (im>0.){
      res=cpmwrite(re)&"+i*"&cpmwrite(im);
   }
   else{
      res=cpmwrite(re)&"-i*"&cpmwrite(-im);
   }
   return sep_i&res&sep_f;
}
```

```
bool C::prnOn(ostream& str )const
{
   //if (!CpmRoot::writeTitle("C",str)) return false;
   cpmwat;
   cpmp(re);
   cpmp(im);
   return true;
}

bool C::scanFrom(istream& str )
{
   cpms(re);
   cpms(im);
   return true;
}

// inversions

C C::inv(void)const
// same logic as inv(R) for error handling
// see Press et al. p. 177 (5.4.5) for the algorithm
{
   static const Z maxMes=100;
   static Z mes=1;
   R c_=(re>=0. ? re : -re);
   R d_=(im>=0. ? im : -im);
   if (c_>=d_){
      if (c_==0.) goto DIVBYZERO;
      R y=im/re; R z=R(1.)/(re+im*y); return C(z,-y*z);
   }
   else{
      if (d_==0.) goto DIVBYZERO;
      R y=re/im; R z=R(1.)/(re*y+im); return C(y*z,-z);
   }

DIVBYZERO:

      if (mes==maxMes){
         mes++; // needed !
         Message::message(
            "C::inv(): argument is (0,0) ... messages discontinued");
      }
      if (mes<maxMes){
         mes++;
         Message::warning(
            "C::inv(): argument is (0,0);  (0,0) returned");
      }
      return C();
}
```

```
C& C::operator /=(R const& s)
{
   R si=CpmRoot::inv(s);
   re*=si;
   im*=si;
   return *this;
}

C CpmRoot::operator /(C const& z, R const& s)
{
   R si=cpminv(s);
   return z*si;
}

// Functions related to polar representation

R CpmRoot::arg(C const& x)
{
   return x.arg();
}

void C::pol_(R r, R phi)
{
   re=r*cpmcos(phi);
   im=r*cpmsin(phi);
}

void C::toPolar(R& r, R& phi)const
{
   r=abs();
   phi=arg();
}

// Square

C C::sqr(void)const
{
   C res(*this);
   return res*res;
}

// Square root

C C::sqrt(void)const
{
   R r=abs(), phi=arg();
   r=cpmsqrt(r);
   phi*=0.5;
   return C(r*cpmcos(phi),r*cpmsin(phi));
}
```

```
// Elementary transcendental functions. We need the following real
// functions to be provided by the compiler:

//      exp, log, sin, cos, sqrt, atan. If complex numbers Ca are to be
//      defined based on a class Ra of numbers of arbitrary precission,
//      we have to implement these functions also for Ra.

C C::exp()const
{
   R r=cpmexp(re);
   return C(r*cpmcos(im),r*cpmsin(im));
}

C C::ln()const
{
   R r=abs(), phi=arg();
   r=cpmlog(r);  // library function
   return C(r,phi);
}

C CpmRoot::log10(C const& z)
// logarithm to basis 10
{
   static const R log10_=1./cpmlog(10.);
   return log10_*ln(z);
}

C CpmRoot::pow(C const& z, const Z& n)
{
   C res(1.,0);
   if (n==0) return res;

   C z_; Z n_;
   if (n>0){
      z_=z;
      n_=n;
   }
   else{
      z_=!z;
      n_=-n;
    }
   for (Z k=1; k<=n_; k++) res*=z_;
   return res;
}

C CpmRoot::pow(C const& z, C const& w)
{
   return exp(w*ln(z));
}
```

```
C CpmRoot::sin(C const& z)
{
   return (exp(z.timesI())-exp(z.dividedByI())).dividedBy2I();
}

C CpmRoot::cos(C const& z)
{
   return (exp(z.timesI())+exp(z.dividedByI()))*0.5;
}

C CpmRoot::tan(C const& z){ return sin(z)/cos(z);}

C CpmRoot::cot(C const& z){ return cos(z)/sin(z);}

C CpmRoot::sinh(C const& z){ return 0.5*(exp(z)-exp(-z));}

C CpmRoot::cosh(C const& z){ return 0.5*(exp(z)+exp(-z));}

C CpmRoot::tanh(C const& z){ return sinh(z)/cosh(z);}

C CpmRoot::coth(C const& z){ return cosh(z)/sinh(z);}

C CpmRoot::arcsinh(C const& z)
   // Gradshteyn Ryzhik 1.622 5.
{ return ln(z+(z*z+1).sqrt());}

C CpmRoot::arccosh(C const& z)
   // Gradshteyn Ryzhik 1.622 6.
{ return ln(z+sqrt(z*z-1));}

C CpmRoot::arctanh(C const& z)
   // Gradshteyn Ryzhik 1.622 7.
{ return ln((z+C(1))/(C(1)-z))*0.5;}

C CpmRoot::arccoth(C const& z)
   // Gradshteyn Ryzhik 1.622 8.
{ return ln((z+C(1))/(z-C(1)))*0.5;}

C CpmRoot::arcsin(C const& z)
   // Gradshteyn Ryzhik 1.622 1.
{ return (arcsinh(z.timesI())).dividedByI();}

C CpmRoot::arccos(C const& z)
   // Gradshteyn Ryzhik 1.622 2.
{ return (arccosh(z)).dividedByI();}

C CpmRoot::arctan(C const& z)
   // Gradshteyn Ryzhik 1.622 3.
{ return (arctanh(z.timesI())).dividedByI();}
```

```
C CpmRoot::arccot(C const& z)
   // Gradshteyn Ryzhik 1.622 4.
{ return (arccoth(z.timesI())).timesI();}

C C::pow(C const& p)const{ return CpmRoot::pow(*this,p);}
C C::sin()const{ return CpmRoot::sin(*this);}
C C::cos()const{ return CpmRoot::cos(*this);}
C C::tan()const{ return CpmRoot::tan(*this);}
C C::cot()const{ return CpmRoot::cot(*this);}
C C::sinh()const{ return CpmRoot::sinh(*this);}
C C::cosh()const{ return CpmRoot::cosh(*this);}
C C::tanh()const{ return CpmRoot::tanh(*this);}
C C::coth()const{ return CpmRoot::coth(*this);}
C C::arcsin()const{ return CpmRoot::arcsin(*this);}
C C::arccos()const{ return CpmRoot::arccos(*this);}
C C::arctan()const{ return CpmRoot::arctan(*this);}
C C::arccot()const{ return CpmRoot::arccot(*this);}
C C::arcsinh()const{ return CpmRoot::arcsinh(*this);}
C C::arccosh()const{ return CpmRoot::arccosh(*this);}
C C::arctanh()const{ return CpmRoot::arctanh(*this);}
C C::arccoth()const{ return CpmRoot::arccoth(*this);}
   // these added 2005-08-23

// Indicators

/*************** utility functions ******************************/

C C::test(Z cpl)const
{
   const R red=0.5;
      // complex functions like exp() should be included in tests and
      // don't behave uncritical for large arguments
   R a=0.;
   R ar=CpmRoot::test(a,cpl);
   R ai=ar*red;
   return C(ar,ai);
}

C C::ran(Z j)const
{
   R a,b;
   if (j==0){
      a=CpmRoot::ran(re,0);
      b=CpmRoot::ran(im,0);
   }
   else{
      Z j2=j*2;
      a=CpmRoot::ran(re,j2);
      b=CpmRoot::ran(im,j2+1);
```

```
   }
   return C(a,b);
}

R C::dis(C const& z)const
{
   return CpmRoot::disDefFun(abs(),z.abs(),(*this-z).abs());
}

Z C::com(C const& s)const
{
   if (re<s.re) return 1;
   if (re>s.re) return -1;
   if (im<s.im) return 1;
   if (im>s.im) return -1;
   return 0;
}

Z C::hash()const
{
   Z ix=CpmRoot::hash(re);
   Z iy=CpmRoot::hash(im);

   return ix^iy;
}

C C::scaleTo(R const& r)const
      // returns a complex number of absolute value |r| and the direction
      // of *this for positive r and antidirection for negative r.
{
   R rOrig=abs();
   if (rOrig==0.) return C(r,0);
   C res=*this;
   return res*(r/rOrig);
}
```

10 cpmcompdef.h

```
//? cpmcompdef.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


// no double include guard needed since this occurs only as
// '#include <cpmcompdef.h>' in cpmdefinitions.h and the letter file
// is guarded against double inclusion.
/*************************************************************************
   cpmcompdef.h
   Description: In this file one defines how compiler options of the group
   -D influence the compilation directives defined in cpmdefinition.h
*************************************************************************/
// ways to define R by compiler options of group -D
// one line in a makefile could for instance be
// defines = -D_CONSOLE -DCPM_RMP_C
// compilation than would result in an executable which uses a
// representation with 64 decimal places (actually the computed equivalent
// of binary places) for each number of type R.
// We have two ways to implement multiple precision arithmetics: based on
// Pavel Holoborodko's mpmreal.h (in which case CPM_USE_MPREAL has to be
// defined) or on boost::multiprecision (here CPM_USE_MPREAL has to be not
// defined). Presently the second way has the disadvantage that it can be
// used together with the Eigen library only if compilation is done under
// the -fpermissive option.
#if defined(CPM_R)  // R double
   #undef CPM_LONG
   #undef CPM_USE_MPREAL
   #undef CPM_MP
#elif defined(CPM_RMP) // multiprecision R by mpreal.h
   #undef CPM_LONG
   #undef CPM_USE_MPREAL
   #undef CPM_MP
   #define CPM_MP 16
#elif defined(CPM_RMP_A)
```

```
   #undef CPM_LONG
   #undef CPM_MP
   #define CPM_MP 24
   #define CPM_USE_MPREAL
#elif defined(CPM_RMP_B)
   #undef CPM_LONG
   #undef CPM_MP
   #define CPM_MP 32
   #define CPM_USE_MPREAL
#elif defined(CPM_RMP_C)
   #undef CPM_LONG
   #undef CPM_MP
   #define CPM_MP 64
   #define CPM_USE_MPREAL
#elif defined(CPM_RMP_D)
   #undef CPM_LONG
   #undef CPM_MP
   #define CPM_MP 128
   #define CPM_USE_MPREAL
#elif defined(CPM_RMP_E)
   #undef CPM_LONG
   #undef CPM_MP
   #define CPM_MP 256
   #define CPM_USE_MPREAL
#elif defined(CPM_RLONG)
   #undef CPM_MP
   #define CPM_LONG
#elif defined(CPM_R)
   #undef CPM_MP
   #undef CPM_LONG
#elif defined(CPM_RBMP_A)   // R (by) boost multiprecision
   #undef CPM_LONG
   #undef CPM_MP
   #define CPM_MP 24
   #undef CPM_USE_MPREAL
#elif defined(CPM_RBMP_B)
   #undef CPM_LONG
   #undef CPM_MP
   #define CPM_MP 32
   #undef CPM_USE_MPREAL
#elif defined(CPM_RBMP_C)
   #undef CPM_LONG
   #undef CPM_MP
   #define CPM_MP 64
   #undef CPM_USE_MPREAL
#elif defined(CPM_RBMP_D)
   #undef CPM_LONG
   #undef CPM_MP
   #define CPM_MP 128
   #undef CPM_USE_MPREAL
```

```
#elif defined(CPM_RBMP_E)
   #undef CPM_LONG
   #undef CPM_MP
   #define CPM_MP 256
   #undef CPM_USE_MPREAL
#endif
```

11 cpmdefinitions.h

```
//? cpmdefinitions.h
//? C+- by Ulrich Mutze. Status of work 2013-04-03.
//? Copyright (c) 2013 Ulrich Mutze, www.ulrichmutze.de
//? All rights reserved

#ifndef CPM_DEFINITIONS_H_
#define CPM_DEFINITIONS_H_
/*************************************************************************
   cpmdefinitions.h
   Description: In this file one can place defines that are valid in all
   translation units working with Cpm classes
   Cpm = Classes for Physics and Mathematics || C+- (C plus minus)
         -           -           -                  - -    -
   This is a special version of the file which allows manipulating the
   definition of R by defines given to the compiler as -DCPM_RMP_A or
   to -DCPM_RMP_E or -DCPM_RLONG if more than normal double is needed.
   This helps for automatic building series of executables for all
   possible ways of implementing floating point numbers
*************************************************************************/

#define CPM_NAMEOF

#define CPM_USECOUNT

#define CPM_RANGE_CHECK

#define CPM_Fn

//#define CPM_LONG

//#define CPM_MP

//#define CPM_USE_MPREAL

//#define CPM_USE_MPI

#include <cpmcompdef.h>
#endif
```

12 cpmf.h

```
//? cpmf.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_F_H_
#define CPM_F_H_
/*
   Purpose: see cpmfl.h

*/
#include <cpmv.h>
   // includes <cpmfl.h>, where the essential template F<X,Y>
   // is defined

namespace CpmFunctions{

   using CpmArrays::V;
   using CpmArrays::IvZ;

   template <class X, class Y>
   V<Y> apply(F<X,Y>  const& f, V<X>  const& xs)
      // action on sequences: applying f to xs gives the return value of
      // the present function
   {
      IvZ d=xs.dom(); // general indexing
      V<Y> res(d);
      for (Z i=xs.b();i<=xs.e();i++) res[i]=f(xs[i]);
      return res;
   }

namespace aux{
#ifdef CPM_Fn
template
   <class X, class P1, class P2, class P3, class P4, class P5, class Y>
```

```
class Par5FncObj : public FncObj<X,Y>{

   const P1 p1;
   const P2 p2;
   const P3 p3;
   const P4 p4;
   const P5 p5;
   Y (* const fp)
      (X const&, P1 const&, P2 const&, P3 const&, P4 const&, P5 const&);
public:
   Y operator()(X const& x)const
      { return (*fp)(x,p1,p2,p3,p4,p5);}
   Par5FncObj(
      Y (*g)(X const&,P1 const&,P2 const&,P3 const&,P4 const&,P5 const&),
      P1 const& q1,P2 const& q2,P3 const& q3,P4 const& q4, P5 const& q5):
      fp(g),p1(q1),p2(q2),p3(q3),p4(q4),p5(q5){}
};

template
   <class X, class P1, class P2, class P3, class P4,
   class P5, class P6, class Y>
class Par6FncObj : public FncObj<X,Y>{

   const P1 p1;
   const P2 p2;
   const P3 p3;
   const P4 p4;
   const P5 p5;
   const P6 p6;
   Y (* const fp)(X const&, P1 const&, P2 const&, P3 const&,
      P4 const&, P5 const&, P6 const&);
public:
   Y operator()(X const& x)const
      { return (*fp)(x,p1,p2,p3,p4,p5,p6);}
   Par6FncObj(
      Y (*g)(X const&,P1 const&,P2 const&,P3 const&,P4 const&,
         P5 const&, P6 const&),
         P1 const& q1,P2 const& q2,P3 const& q3,
         P4 const& q4, P5 const& q5, P6 const& q6):
         fp(g),p1(q1),p2(q2),p3(q3),p4(q4),p5(q5),p6(q6){}
};

#endif // CPM_Fn

} // aux

// For F5 and F6, we do not introduce the corresponding F5_1,...F6_6
// which would enable using the parameters as function arguments. Having
// this capability for up to four parameters is sufficient. See cpmfl.h for // // this matter.
```

```
#ifdef CPM_Fn
//////////////////////// class F5<> ////////////////////////////////////

template <class X, class P1, class P2, class P3, class P4,
   class P5, class Y>
class F5{ // functions with five parameters
    const P1 p1;
    const P2 p2;
    const P3 p3;
    const P4 p4;
    const P5 p5;
    typedef F5<X,P1,P2,P3,P4,P5,Y> Type;
    CPM_INVAR(F5)
public:
    F5( P1 const& p1_,P2 const& p2_,P3 const& p3_,P4 const& p4_,
       P5 const& p5_):
    p1(p1_),p2(p2_),p3(p3_),p4(p4_),p5(p5_){}
    F<X,Y> operator()(Y (*f)(X const&,P1 const&,P2 const&,
       P3 const&,P4 const&,P5 const&))const;
};

template <class X, class P1, class P2, class P3, class P4,
   class P5,  class Y>

F<X,Y> F5<X,P1,P2,P3,P4,P5,Y>::operator()(Y (*f)(X const&,P1 const&,
   P2 const&, P3 const&,P4 const&,P5 const&))const
{
   using namespace aux;
   return
   F<X,Y>(new Par5FncObj<X,P1,P2,P3,P4,P5,Y>(f,p1,p2,p3,p4,p5));
}

//////////////////////// class F6<> ////////////////////////////////////

template <class X, class P1, class P2, class P3, class P4,
   class P5, class P6, class Y>
class F6{ // functions with six parameters
    const P1 p1;
    const P2 p2;
    const P3 p3;
    const P4 p4;
    const P5 p5;
    const P6 p6;
    typedef F6<X,P1,P2,P3,P4,P5,P6,Y> Type;
    CPM_INVAR(F6)
public:
    F6( P1 const& p1_,P2 const& p2_,P3 const& p3_,P4 const& p4_,
       P5 const& p5_, P6 const& p6_):
    p1(p1_),p2(p2_),p3(p3_),p4(p4_),p5(p5_),p6(p6_){}
    F<X,Y> operator()(Y (*f)(X const&,P1 const&,P2 const&,
```

```
       P3 const&,P4 const&,P5 const&, P6 const&))const;
};

template <class X, class P1, class P2, class P3, class P4,
   class P5, class P6, class Y>

F<X,Y> F6<X,P1,P2,P3,P4,P5,P6,Y>::operator()(Y (*f)(X const&,P1 const&,
   P2 const&, P3 const&,P4 const&,P5 const&, P6 const&))const
{
   using namespace aux;
   return
   F<X,Y>(new Par6FncObj<X,P1,P2,P3,P4,P5,P6,Y>
      (f,p1,p2,p3,p4,p5,p6));
}

#endif // CPM_Fn

namespace aux{

template <class X1, class X2, class Y1, class Y2>
class cart: public FncObj<cpmX2<X1,X2>,cpmX2<Y1,Y2> >{ // CART stands for
    // Cartesian

   const F<X1,Y1> f1;
   const F<X2,Y2> f2;

public:
   cpmX2<Y1,Y2> operator()(const cpmX2<X1,X2>& x)const
   { return cpmX2<Y1,Y2>(f1(x.first),f2(x.second));}

   cart(const F<X1,Y1>& f1_,const F<X2,Y2>& f2_):f1(f1_),f2(f2_){}
};

// pairing of functions which are defined on the same class

////////////////////////// class cart1<> ///////////////////////////////

template <class X, class Y1, class Y2>
class cart1: public FncObj<X,cpmX2<Y1,Y2> >{

   const F<X,Y1> f1;
   const F<X,Y2> f2;
public:
   cpmX2<Y1,Y2> operator()(X const& x)const
   { return cpmX2<Y1,Y2>(f1(x),f2(x));}

   cart1(const F<X,Y1>& f1_,const F<X,Y2>& f2_):f1(f1_),f2(f2_){}
};

} // aux
```

```
//////////////////// function pairOf<> ////////////////////////////////

template <class X1, class X2, class Y1, class Y2>
F<cpmX2<X1,X2>,cpmX2<Y1,Y2> > pairOf(const F<X1,Y1>& f1,
   const F<X2,Y2>& f2)
   // Cartesian product of functions:
   // cpmX2(f1,f2)(x1,x2)=(f1(x1),f2(x2)).
   // name changed from pair to pairOf to avoid clash with std name
   // (99-5-31)
{
   using namespace aux;
   return F<cpmX2<X1,X2>,cpmX2<Y1,Y2> >(new cart<X1,X2,Y1,Y2>(f1,f2));
}

////////////////////////// operator&&<> ////////////////////////////////

template <class X, class Y1, class Y2>
F<X,cpmX2<Y1,Y2> > operator &&(const F<X,Y1>& f1,const F<X,Y2>& f2)
   // Cartesian product of functions which are defined on the same class.
    // (f1&&f2)(x)=(f1(x),f2(x)).
    // Elegant way for flexible definition of binary operators by
    // concatenation
    // with a mapping cpmX2<Y1,Y2> ---> Y3
{
   using namespace aux;
   return F<X,cpmX2<Y1,Y2> >(new cart1<X,Y1,Y2>(f1,f2));
}

// 'changing the number of variables from 2 to 1'

// Bind1 and Bind2 as classes of the same design as FuncPar.
// Assume:
// class X1;
// class X2;
// class Y;
// F<cpmX2<X1,X2>,Y> f(...);
// X1 x1=...;
// F<X2,Y> f1=Bind1<X1,X2,Y>(x1)(f);
// X2 x2=...;
// Y ya=f(cpmX2<X1,X2>(x1,x2));
// Y yb=f1(x2);
// then ya==yb

namespace aux{

template <class X1, class X2, class Y>
class bind1: public FncObj<X2,Y>{
   // bind1 stands for bind first variable
   const X1 x1;
```

```
   const F<cpmX2<X1,X2>,Y> f;
public:
   Y operator()(const X2& x2)const{ return f( cpmX2<X1,X2>(x1,x2));}
   bind1( const F<cpmX2<X1,X2>,Y> & f_, const X1& x1_):f(f_),x1(x1_){}
};

template <class X1, class X2, class Y>
class bind2: public FncObj<X1,Y>{
   // bind2 stands for bind second variable
   const X2 x2;
   const F< cpmX2<X1,X2>,Y>  f;
public:
   Y operator()(const X1& x1)const{ return f( cpmX2<X1,X2>(x1,x2));}
   bind2( const F< cpmX2<X1,X2>,Y>& f_, const X2& x2_):f(f_),x2(x2_){}
};

} // aux

////////////////////////// class Bind1<> ///////////////////////////////

template <class X1, class X2, class Y>
class Bind1 { // binding the first parameter
    const X1 x1;
public:
    Bind1(const X1& x1_):x1(x1_){}
    F<X2,Y> operator()(const F<cpmX2<X1,X2>,Y>& f)const
    { return F<X2,Y>(new aux::bind1<X1,X2,Y>(f,x1));}
};

////////////////////////// class Bind2<> ///////////////////////////////

template <class X1, class X2, class Y>
class Bind2 { // binding the second parameter
    const X2 x2;
public:
    Bind2(const X2& x2_):x2(x2_){}
    F<X1,Y> operator()(const F<cpmX2<X1,X2>,Y>& f)const
    { return F<X1,Y>(new aux::bind2<X1,X2,Y>(f,x2));}
};

// range and domain conversion

namespace aux{

template <class X1, class X2, class Y>
class convertDomain: public FncObj<X2,Y>{
   // means to transform a F<X,Y> in a F<X',Y> if there is an
   // automatic conversion from X to X'

   const F<X1,Y> f;
```

```
public:
   Y operator()(const X2& x)const{ return f(x);}
   convertDomain( const F<X1,Y>& f_):f(f_){}
};

template <class X, class Y1, class Y2>
class convertRange: public FncObj<X,Y2>{
   // means to transform a F<X,Y> in a F<X,Y'> if there is an
   // automatic conversion from Y to Y'

   const F<X,Y1> f;

public:
   Y2 operator()(X const& x)const{ return f(x);}
   convertRange( const F<X,Y1>& f_):f(f_){}
};

} // auxiliatry

////////////////////// class ConvertDomain<> //////////////////////////

template <class X1, class X2, class Y>
// usage e.g. :
//   F<R,C> f=.....
//   ConvertDomain<R,N,C> converter;
//   F<N,C> f_N=converter(f);
// The same converter object can be used for converting other functions
// of the same domain and range classes.
class ConvertDomain{ // converting the function domain
public:
   ConvertDomain(void){}
   F<X2,Y> operator()(const F<X1,Y>& f)const
      // transforms a F<X1,Y> in a F<X2,Y> if there is an
      // automatic conversion from X1 to X2.
   {
      return F<X2,Y>(new aux::convertDomain<X1,X2,Y>(f));
   }
};

////////////////////// class ConvertRange ////////////////////////////

template <class X, class Y1, class Y2>
// Usage:
//   F<R,C> f=.....
//   ConvertRange<R,C,Cd> converter;
//   F<R,Cd> f_Cd=converter(f);
// The same converter object can be used for converting other functions
// of the same domain and range classes.
class ConvertRange{ // converting the function range
public:
```

```
   ConvertRange(void){}
   F<X,Y2> operator()(const F<X,Y1>& f)const
      // transforms a F<X,Y1> in a F<X,Y2> if there is an
      // automatic conversion from Y1 to Y2.
   {
      return F<X,Y2>(new aux::convertRange<X,Y1,Y2>(f));
   }
};

/// functions of more than one variable

namespace aux{

template <class Y1, class Y2, class Y3>
class Arg2FncObj : public FncObj<cpmX2<Y1,Y2>, Y3>{

   const Y3 (*fp)(const Y1&, const Y2& );

public:
   Y3 operator()(const cpmX2<Y1,Y2>& x)const
   { return fp(x.c1(),x.c2());}

   Arg2FncObj( Y3 (*g)(const Y1&, const Y2&)):fp(g){}
};

} // aux

#ifdef CPM_Fn
//////////////////// class F_2<> ////////////////////////////////////////

template <class Y1, class Y2, class Y3>
class F_2{ // functions of two variables

   const F<cpmX2<Y1,Y2>,Y3> f;

public:

   F_2(Y3 (*f_)(const Y1&, const Y2&)):
      f(new aux::Arg2FncObj<Y1,Y2,Y3>(f_)){}
      // construction from function pointer

   F_2(const F<CpmArrays::X2<Y1,Y2>,Y3>& f_):f(f_){}
      // construction from 'smart function object'

   F<CpmArrays::X2<Y1,Y2>,Y3> operator()(void)const{ return f;}
      // evaluation to 'smart function object'

   Y3 operator()(const Y1& y1, const Y2& y2)const
      // evaluation to value
   { return f(CpmArrays::X2<Y1,Y2>(y1,y2));}
```

```
   F<Y2,Y3> c1(const Y1& y1)const
      // evaluation to a 'smart function object'
      // representing a function of one variable;
      // obtained by binding the first of the two variables
   { return Bind1<Y1,Y2,Y3>(y1)(f);}

    F<Y1,Y3> c2(const Y2& y2)const
     // ...  binding the second of the two variables
   { return Bind2<Y1,Y2,Y3>(y2)(f);}

};

#endif // CPM_Fn

// define a function by selecting one function value from two functions

namespace aux{

template <class X, class Y>
class sel: public FncObj<X,Y>{

   const F<X,Y>  f1;
   const F<X,Y>  f2;
   const F<X,Z>  s;

public:

   sel(const F<X,Y> & f1_, F<X,Y>  const& f2_, const F<X,Z>& s_):
      f1(f1_),f2(f2_),s(s_){}

   Y operator()(X const& )const;
};

template <class X, class Y>
Y sel<X,Y>::operator()(X const& x)const
{
   Z i=s(x);
   if (i==1) return f1(x);
   else if (i==2) return f2(x);
   else {
      cpmerror
         ("select::operator(): unvalid value of selection function");
      return f1(x); // never done
   }
}

//////////////////// class vselect<> ///////////////////////////////////

// define a function by selecting one function value from a sequence of
```

```
// functions

template <class X, class Y>
class vselect: public FncObj<X,Y>{

   const V< F<X,Y> > f;
   const F<X,Z>  sel;

public:

   vselect(const V< F<X,Y> >& f_, const F<X,Z> & sel_):
      f(f_),sel(sel_){}

   Y operator()(X const& )const;

};

template <class X, class Y>
Y vselect<X,Y>::operator()(X const& x)const
{
   Z i=sel(x);
   return (f[i])(x);
}

} // aux

////////////////////// function select<> ////////////////////////////////

template <class X, class Y>
F<X,Y> select(F<X,Y>  const& f, F<X,Y>  const& g, const F<X,Z>& h)
// The returned function res has the property res(x)==f(x) if h(x)==1,
// res(x)== g(x) if h(x)==2, cpmerror() else
{
   return F<X,Y>(new aux::sel<X,Y>(f,g,h));
}

template <class X, class Y>
F<X,Y> select(const V<F<X,Y> >& f, const F<X,Z>& h)
// The returned function res has the property res(x)=(f[g(x)])(x);
{
   return F<X,Y>(new aux::vselect<X,Y>(f,h));
}

} // namespace

#endif
```

13 cpmfa.h

```
//? cpmfa.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_FA_H_
#define CPM_FA_H_
////////////////////////////////////////////////////////////////////////
//
// Purpose: Define classes describing function-like objects with
//   arithmetic operations
//
////////////////////////////////// class FUOBJ<> ///////////////////////////
#include <cpmfo.h>
#include <cpmtypes.h>

namespace CpmFunctions{

  // using std::ostream;
  // using std::istream;
   using namespace CpmStd;
   using CpmRoot::Z;

   namespace auxiliary { // is within a Cpm... namespace !

//////////////////// tools for defining class Fa<X,Y> ///////////////////

// Advanced arithmetics of functions based on handles to FncObjs

// Arithmetic operations: Here it is assumed that Y is a ring i.e. that
// Y+Y, Y-Y, Y*Y, -Y are defined. Also division Y/Y is assumed to be
// defined. Also template functions/operators will be implemented which
// do not depend only on X and Y.

// class SUMFUO<>
```

```
template <class X, class Y>
class SUMFUO: public BINOPFUO<X,Y>{
   typedef SUMFUO<X,Y> Type;
   CPM_INVAR(SUMFUO)
public:
   Y operator()(const X& x)const{ return this->f1(x)+this->f2(x);}
   SUMFUO( const F<X,Y> & f1In, const F<X,Y>& f2In)
   :BINOPFUO<X,Y>(f1In,f2In){}
};

// class DIFFFUO<>

template <class X, class Y>
class DIFFFUO: public BINOPFUO<X,Y>{

public:
   Y operator()(const X& x)const{ return this->f1(x)-this->f2(x);}
   DIFFFUO( const F<X,Y> & f1In, const F<X,Y> & f2In)
   :BINOPFUO<X,Y>(f1In,f2In){}
};

// class NEGFUO<>

template <class X, class Y>
class NEGFUO: public MONOPFUO<X,Y>{

public:
   Y operator()(X const& x)const{ return -this->f1(x);}
   NEGFUO(const F<X,Y> & f1In):MONOPFUO<X,Y>(f1In){}
};

// class PRODFUO<>

template <class X, class Y>
class PRODFUO: public BINOPFUO<X,Y>{

public:
   Y operator()(const X& x)const{ return this->f1(x)*this->f2(x);}
   PRODFUO( const F<X,Y> & f1In, const F<X,Y> & f2In)
   :BINOPFUO<X,Y>(f1In,f2In){}
};

// class DIVIFUO<>

template <class X, class Y>
class DIVIFUO: public BINOPFUO<X,Y>{

public:
   Y operator()(const X& x)const{ return  this->f1(x)/this->f2(x);}
```

```
   DIVIFUO( const F<X,Y> & f1In, const F<X,Y> & f2In)
   :BINOPFUO<X,Y>(f1In,f2In){}
};

template <class X, class Y>
class INVFUO : public FncObj<X,Y>{
   const F<X,Y> f;
public:
   INVFUO(const F<X,Y>& f_):f(f_){}
   Y operator()(const X& x)const
   { return CpmRoot::invT<Y>(f(x));}
};

// treating operations which need more than 2 template arguments

template <class X1, class X2, class X3>
class TENSFUO: public FncObj< cpmX2<X1,X2>, X3 >{
   // TENS stands for Tensor Product

   const F<X1,X3>  f1;
   const F<X2,X3>  f2;

public:
   X3 operator()(const cpmX2<X1,X2>& x)const
      { return f1(x.get1())*f2(x.get2());}
   TENSFUO( const F<X1,X3> & f1In, const F<X2,X3> & f2In):
      f1(f1In),f2(f2In){}
};

template <class X1, class X2, class X12, class Y1, class Y2, class Y12>
class GENPRODFUO: public FncObj< X12,Y12 >{
   // GENPROD stands for general product. We assume that X12 is a class
   // which defines functions get1() and get2() such that for each x12 in
   // X12 the values of x12.get1() and x12.get2() allow unique automatic
   // conversion to X1 and X2 respectively. Further, we assume that for
   // each y1 in Y1 and y2 in Y2 the value of Y12(y1,y2) is an element in
   // Y12. Notice that the normal tensor product
   // TENSFUO<X1,X2,X3>(f1,f2)
   // would be obtained as
   // GENPRODFUO<X1,X2,cpmX2<X1,X2>,Y3,Y3,Triv<Y3> > (f1,f2)
   // where
   //    template <class X>
   //    class Triv: public X{
   //    public:
   //       Triv(const X& x):X(x){}
   //       Triv(const X& x1, const X& x2):X(x1*x2){}
   //   };
   //

   const F<X1,Y1>  f1;
```

```
   const F<X2,Y2>  f2;

public:
   Y12 operator()(const X12& x)const{ return Y12(f1(x.c1()),f2(x.c2()));}
   GENPRODFUO( const F<X1,Y1> & f1In, const F<X2,Y2>& f2In):
      f1(f1In),f2(f2In){}
};

template <class X, class Y, class S>
   // assumption S*Y defined, in typical situations, Y is a linear space
   // over S

class MULTFUO: public FncObj<X,Y>{

   const F<X,Y>  f1;
   const S s;

public:

   Y operator()(const X& x)const{ return s*(f1(x));}

   MULTFUO(const S& sIn, const F<X,Y> & f1In):f1(f1In),s(sIn){}

};

template <class X, class Y, class S>
   // assumption Y*S defined, in typical situations, Y is a linear space
   // over S

class MULTFUOR: public FncObj<X,Y>{

   const F<X,Y>  f1;
   const S s;

public:

   Y operator()(const X& x)const{ return (f1(x))*s;}

   MULTFUOR(const S& sIn, const F<X,Y> & f1In):f1(f1In),s(sIn){}

};

template <class X, class Y>
   // assumption Y+Y defined

class Y_SUMFUO: public FncObj<X,Y>{

   const F<X,Y>  f1;
   const Y y;
```

```
public:

   Y operator()(const X& x)const{ return (f1(x))+y;}

   Y_SUMFUO(const F<X,Y> & f1In, const Y& yIn):f1(f1In),y(yIn){}

};

template <class X, class Y>
   // assumption Y-Y defined

class Y_DIFFFUO: public FncObj<X,Y>{

   const F<X,Y> f1;
   const Y y;

public:

   Y operator()(const X& x)const{ return f1(x)-y;}

   Y_DIFFFUO(const F<X,Y> & f1In, const Y& yIn):f1(f1In),y(yIn){}

};

template <class X, class Y>
   // assumption Y*Y defined

class Y_PRODFUO: public FncObj<X,Y>{

   const F<X,Y>  f1;
   const Y y;

public:

   Y operator()(const X& x)const{ return f1(x)*y;}

   Y_PRODFUO(const F<X,Y> & f1In, const Y& yIn):f1(f1In),y(yIn){}

};

template <class X, class Y>

class Y_DIVIFUO: public FncObj<X,Y>{

protected:

   const F<X,Y>  f1;
   const Y y;

public:
```

```
   Y operator()(const X& x)const{ return f1(x)/y;}

   Y_DIVIFUO(const F<X,Y> & f1In,const Y& yIn):f1(f1In),y(yIn){}

};

} // auxiliary

////////////////////////////////// class Fa<X,Y> ////////////////////////

// Arithmetics and some additional functionality for Fo<X,Y>

// Now arithmetics is introduced. Just as in class Va, we assume that
// -Y, Y+Y, Y-Y, Y*Y, Y/Y , Y < Y, Y > Y, are defined
// Note 02-05-24:
// Notice that e.g. addition of functions is not implemented by really
// 'adding something together' but arranging pointers to the code
// of all functions in the sum in a manner that when the sum function is
// asked to be evaluated for some value of the argument, all functions
// needed in the sum are ready for evaluation and addition of the
// output-values. This is sometimes called 'deferred evaluation'.
// I found that code of type:
// Fa<X,Y> f;
// Z i,n=3300;
// V< Fa<X,Y> > vf(n);
// for (i=1;i<=n;i++) vf[i]=...;
// for (i=1;i<=n;i++) f+=vf[i];
// caused stack overflow in the last line

template <class X, class Y>
class Fa: public Fo<X,Y> { // version of Fo with arithmetics operations

public:

   typedef Fa<X,Y> Type;
   typedef Y ScalarType;

   Fa(void):Fo<X,Y>(){}
      // default constructor

   Fa(const F<X,Y>& g):Fo<X,Y>(g){}
      // downcast constructor

   Fa(const Fo<X,Y>& g):Fo<X,Y>(g){}
      // downcast constructor

   Fa(const Fa<X,Y>& g):Fo<X,Y>(g){}
      // copy constructor
```

```
   Fa(FncObj<X,Y>* fop):Fo<X,Y>(fop){}
      // constructor from pointers to FncObj

// Construction from function pointers

   Fa( Y (*f)(const X& )):Fo<X,Y>(f){}
   Fa( Y (*f)(X)):Fo<X,Y>(f){}

// Construtor for constant function

   explicit Fa(const Y& y1):Fo<X,Y>(y1){}

   virtual F<X,Y>* clone()const{ return new Fa(*this);}

   virtual Word nameOf()const
   {
      Word wi="Fa<";
      Word wx=CpmRoot::Name<X>()(X());
      Word wy=CpmRoot::Name<Y>()(Y());
      return wi&wx&","&wy&">";
   }

   CPM_SUM_C
   CPM_PRODUCT_C
   CPM_DIFFERENCE
   CPM_DIVISION
   CPM_SCALAR_C
   CPM_IO_V

   void scl_(X const& fac, X const& x0=X());
      //: scale
      // changes the function f into x|-->f(x0+(x-x0)*fac)
      // assumes that X defines addition, subtraction, and multiplication,
      // typically that X is a ring. Similar to function stretch_ in
      // class R_Func. For X=R this operation applied with fac>1
      // lets features of function *this (such as a peak width) shrink and
      // fac<1 lets them grow.
};

namespace{// anonymus
   template <class X, class Y>
   Y scaleFunc(X const& x, X const& x0, X const& fac, F<X,Y> const& f)
   { return f(x0+(x-x0)*fac);}
}// anonymus

template <class X, class Y>
void Fa<X,Y>::scl_(X const& fac, X const& x0)
// elegant implementation, avoids using operator new directly.
{
#ifdef CPM_Fn
```

```
   *this=F3<X,X,X,F<X,Y>,Y>(x0,fac,*this)(scaleFunc);
#else
   *this=F<X,Y>(bind(scaleFunc<X,Y>,_1,x0,fac,*this));
#endif
}

// functions(operators) outside this class. Here additional template
// arguments are allowed, which creates the flexibility needed for
// treating composition and tensor product.

template <class X1, class X2, class X3>

Fa<cpmX2<X1,X2>,X3> operator || (const Fa<X1,X3>& f13,
          const Fa<X2,X3>& f23)
   // Tensor product of functions: Assumption is that X3*X3 is defined.
   // This creates functions of two variables out of two functions of one
   // variable. Symbol || should remind to the old symbol )( for the
   // 'dyadic product' which is essentially the same construction
{
   using namespace auxiliary;
   return Fa<cpmX2<X1,X2>,X3>(new TENSFUO<X1,X2,X3>(f13,f23));
}

template <class X, class Y, class S>
   // assumption S*Y defined, in typical situations,
   // Y is a linear space over S

Fa<X,Y> mult(const S& sIn, const Fa<X,Y>& fIn)
   // Function values are multiplied by s, operator * instead of
   // mult results in compiler errors caused by mistaking this as
   // another operator *
{
   using namespace auxiliary;
   return Fa<X,Y>(new MULTFUO<X,Y,S>(sIn,fIn));
}

template <class X, class Y, class S>
   // assumption Y*S defined, in typical situations,
   // Y is a linear space over S
Fa<X,Y> multR(const S& sIn, const Fa<X,Y>& fIn)
   // Function values are multiplied by s, operator * instead of
   // mult results in compiler errors caused by mistaking this as
   // another operator *
{
   using namespace auxiliary;
   return Fa<X,Y>(new MULTFUOR<X,Y,S>(sIn,fIn));
}

template <class X, class Y>
Fa<X,Y> Fa<X,Y>::neg(void)const
```

```
{
   using namespace auxiliary;
   return Fa<X,Y>(new NEGFUO<X,Y>(*this));
}

template <class X, class Y>
Fa<X,Y> Fa<X,Y>::inv(void)const
{
   return Fa<X,Y>(new auxiliary::INVFUO<X,Y>(*this));
}

template <class X, class Y>
Fa<X,Y> Fa<X,Y>::net(CpmRoot::Z i)const
{
   Y y0;
   Y yn=CpmRoot::netT<Y>(y0,i);
   return Fa<X,Y>(yn);
}

template <class X, class Y>
Fa<X,Y> Fa<X,Y>::operator +(const Fa<X,Y>& f )const
{
   using namespace auxiliary;
   return Fa<X,Y>(new SUMFUO<X,Y>(*this,f));
}

template <class X, class Y>
Fa<X,Y> Fa<X,Y>::operator *(const Fa<X,Y>& f )const
{
   using namespace auxiliary;
   return Fa<X,Y>(new PRODFUO<X,Y>(*this,f));
}

template <class X, class Y>
Fa<X,Y> Fa<X,Y>::operator *(const Y& y )const
{
   using namespace auxiliary;
   return Fa<X,Y>(new Y_PRODFUO<X,Y>(*this,y));
}

template <class X, class Y>
Fa<X,Y> Fa<X,Y>::operator +(const Y& y )const
{
   using namespace auxiliary;
   return Fa<X,Y>(new Y_SUMFUO<X,Y>(*this,y));
}

template <class X, class Y>
Fa<X,Y> operator *( const Y& y, const Fa<X,Y>& f1 )
{
```

```
//   using namespace auxiliary;
   return (f1*y);
}

// I/O interface trivial. No assumtions on X,Y

template <class X, class Y>
bool Fa<X,Y>::prnOn(ostream& out)const
{
   return CpmRoot::writeTitle(" a function",out);
}

template <class X, class Y>
bool Fa<X,Y>::scanFrom(istream& in)
{
   in;
  // return (in!=0);
   if(!in) return false;
   else return true;
}

} // namespace

#endif
```

14 cpmfl.h

```
//? cpmfl.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_FL_H_
#define CPM_FL_H_
/*
Purpose: Define class F<X,Y> describing 'functions' from
   X to Y. Later a derived class Fa<X,Y> will define arithmetic
   operations. This description also has in mind this class.
   The classes have default constructor, copy constructor, and assignment
   operator. Thus these 'functions' behave like the most conventional
   arithmetic quantities (they have value semantics, moreover they
   satisfy the strict value interface, see preamble of cpmv.h)

   Example:

      F<R,R> f1(cos);
      F<R,R> f2(sin);
      F<R,R> f3=f1&f2;
      F<R,R> f4=f1+f2;
      R x=1.2345;
      R eps1=f3(x)-sin(cos(x));
      R eps2=f4(x)-sin(x)-cos(x);

   will result in eps1=eps2=0. I.e. the concatenation of functions sin()
   and cos() can be created as a F object simply by a binary operator
   acting on the F objects corresponding to sin() and cos() (notice the
   order convention for & which is not intuitive to everybody).
   F<>-instances are allowed as arguments and return values of functions:

      F<C,C> power3(const F<C,C>& g)
      {
         return g*g*g;
```

```
   }

creates the third power of a complex function, i.e.
   power3(g)(x)=g(x)*g(x)*g(x).

In a nutshell, these new functions behave just like normal variables.
They can also serve as data members in class definitions. This is
important in physics and engineering applications where e.g.
electrical fields now can be directly represented as functions instead
of discrete value samples. They also can form components of vectors.

For instance, a family of functions consisting of the first 101
Chebyshev polynomials
      T0(x),....T100(x)
can be defined by verbally coding the (numerically not optimal)
definition

      Tn(x):=cos(n*arccos(x)):
Z n=100;
IvZ iv(0,n);
V<Fa<R,R> > Chebyshev(iv);
Fa<R,R> ACos(acos);
Fa<R,R> Cos(cos);
for (Z i=0; i<=n; i++) Chebyshev[i]=(ACos*i)&Cos ;

The value of the 27-th Chebyshev polynomial for x=0.5 then will be
written as Chebyshev[27](0.5)

In defining new classes, data members of type F<X,Y> are allowed
and for the classes defined this way the default copy constructor
and default assignment operator are OK.

Compared to all other classes that I dealt with, F<X,Y> shows
the particularity that it is not possible to implement
CPM_IO and CPM_ORDER (see cpminterfaces.h) in a useful manner.
Later Fr<X,Y> tries to do the best one can in this respect.

Remarks on 'functions as values':

Consider

   R sum=0;
   for (Z i=1;i<=1000;++i) sum+=(result of some complicated
      computation with i);
   R x=sum;

and the similiar construct

   F<R,R> f;
   for (Z i=1;i<=1000;++i) f+=(Sin*i); // (*)
```

```
      F<R,R> g=f;

   There is a remarkable difference between the two:
   Whenever x will be used in subsequent parts of the program it is
   simply a number and the potential pain of getting to it is forgotten.
   On the contrary, when using g by evaluating it for specific
   values of the argument, e.g.
      R y=g(137)-g(1./137);
   each evaluation has to go through the huge expression which g is
   defined in (*) to be. What was achieved by defining g compactly
   was to compose the code defining the Sin*i into the code defining g.
   But this code was composed verbatim without any intent (and chance)
   of simplification or compactification. So the normal idea  associated
   with evaluation is misleading in this case.
   In a time-stepping simulation one could be tempted
   to replace in each step a data member F<X,X> f of the evolving system
   by f&g with some transformation g characteristic for the step under
   consideration. This would make f more expensive to compute with each
   step (actually we would run into stack overflow sometime).
   Here the right way is to parameterize the mappings and express their
   composition in terms of a computation done with these parameters.
   Sophus Lie has forseen the importance of this kind of computational
   treatment of the combination of transformations. With group theory,
   Lie groups of transformations, and their linear representations, there
   has a huge body of knowledge grown on this topic.
*/

#include <cpmuc.h>
   // for reference counting handle template CpmArrays::P<X>
#include <cpmword.h>
#include <functional>

//#undef CPM_Fn
//#define CPM_Fn CPM_Fn has to be set in cpmdefinitions.
// The next version of C+- will very probably have the form which
// corresponds here to #undef CPM_Fn.
// No longer so sure. The simple basic run of the my pala program
// first ran 1% faster with the old method. But a new run did fiffer only
// by 0.2 s of 95 s. So,there seems to be no significant speed difference.

namespace CpmFunctions{

   using CpmRoot::Z; // for order relation
   using CpmRoot::Word;
   using CpmArrays::P;

////////////////////////////// class FncObj<> ////////////////////////

template <class X, class Y>
class  FncObj{ // function objects
```

```
   // FncObj stands for function object
   // A class which is derived from some FncObj<X,Y> is said to be a
   // function class and all instances of function classes are called
   // function objects. A function object can do everything (and more)
   // than a function can do. Unlike a function, a function class can be
   // defined within a function bloc. Let G be a function class and g an
   // object belonging to it. The value of g at x can be written as
   // g(x) (see. operator()). If in deriving G from
   // FncObj<X,Y>, no further non-static data members are added (then
   // G is said to be a pure function class), G has just one instance
   // g; this is selected by a statement as simple as
   //      G g;
   // Despite this close relationship between function objects and their
   // classes, there is an important syntactic difference between these
   // entities. As already mentioned, function values are accessed via
   // instances, e.g. Y y=g(x). Function arithmetics can be implemented
   // via template classes, among the arguments of which are
   // function classes. We will use such templates only as a transitory
   // step in implementing value-like function objects in F<X,Y>,
   // Fa<X,Y>.

   // Notice that FncObj<X,Y> is an abstract class. This is a usefull
   // prevention against invalid attempts to re-define operator() (e.g.
   // by forgetting the final 'const' attribute.
   //
   // Example: function class describing the sin function:
   //
   //    class Sin: public FncObj<R,R>{
   //       public:
   //       R operator()(const R& x)const{ return sin(x);}
   //    };
   // classes derived from FncObj may implement a more flexible and
   // effective evaluation scheme than mere function: certain
   // preparatory actions (e.g. computation of auxiliary quantities that
   // are held as data members) can be done by constructor code, so that
   // evaluation of operator()(X const&) gets simpler.

   typedef FncObj<X,Y> Type;
   CPM_INVAR(FncObj)
      // makes sure that no client class will use FncObj in a way that
      // makes use of copy or assignment

public:

   FncObj(void){}

   virtual ~FncObj(void){}

   virtual   Y operator()(X const& x)const=0;
```

```
};

namespace aux{
// prevention against polluting namespace CpmFunctions
// The classes to be defined in this namespace aux are not intended
// for direct usage. They will be used internally for building
// classes which have a more expressive interface: the
// classes F<>,F1<>, .... F6<>
// and
// F1_1<>, F2_1<>, F2_2<>, F3_1<>, F3_2<>, F3_3<>, F4_1<>, F4_2<>,
// F4_3<>, F4_4<>

////////////////// ParFncObj ////////////////////////////////////////////

template <class X, class Y>
class ParFncObj : public FncObj<X,Y>{

   Y (* const fp)(X);
public:
   Y operator()(X const& x)const{ return (*fp)(x);}
   ParFncObj( Y (*g)(X)):fp(g){}
};

////////////////// Par0FncObj ///////////////////////////////////////////
template <class X, class Y>
class Par0FncObj : public FncObj<X,Y>{

   Y (* const fp)(X const&);

public:
   Y operator()(X const& x)const{ return (*fp)(x);}
   Par0FncObj( Y (*g)(X const&)):fp(g){}
};

////////////////// StdFunctionObj ///////////////////////////////////////
template <class X, class Y>
// Addition 2020-02-13. Means to use also std::function<Y(X)> objects
// as a source for F<X,Y>-objects. Requires C++11 or higher.
class StdFunctionObj : public FncObj<X,Y>{

   const std::function<Y(X)> f_;

public:
   Y operator()(X const& x)const{ return f_(x);}
   StdFunctionObj( std::function<Y(X)> f):f_(f){}
};

////////////////// ParcFncObj ///////////////////////////////////////////
// Addition 2010-01-13. Means to use also functions with
// constant return value as a source for F<X,Y>-objects.
```

```
template <class X, class Y>
class ParcFncObj : public FncObj<X,Y>{

   const Y (* const fp)(X const&); // This is the prototype of
   // many mathematical functions in mpreal have (unfortunately
   // up to an additional default argument)

public:
   Y operator()(X const& x)const{ return Y((*fp)(x));}
   ParcFncObj( const Y (*g)(X const&)):fp(g){}
};

//////////////////// Par1FncObj ////////////////////////////////////////
template <class X, class P1, class Y>
class Par1FncObj : public FncObj<X,Y>{

   const P1 p_;
   Y (* const f_)(X const&, P1 const& );

public:
   Y operator()(X const& x)const{ return (*f_)(x,p_);}
 //  P1 const& getP(void)const{return p_;} // experiment
   Par1FncObj( Y (*g)(X const&, P1 const&), P1 const& p):p_(p),f_(g){}
};

/////////////// Par1_1FncObj ///////////////////////////////////////
template <class X, class P1, class Y>
class Par1_1FncObj : public FncObj<X,Y>{

   const X x_;
   Y (* const f_)(X const&, P1 const& );

public:
   Y operator()(P1 const& p)const{ return (*f_)(x_,p);}
   Par1_1FncObj( Y (*g)(X const&, P1 const&), X const& x):x_(x),f_(g){}
};

#ifdef CPM_Fn
/////////////// Par2FncObj ///////////////////////////////////////
// See F2, F2_1, F2_2 for the reasons to introduce these class templates

template <class X, class P1, class P2, class Y>
class Par2FncObj : public FncObj<X,Y>{

   const P1 p1_;
   const P2 p2_;
   Y (* const f_)(X const&, P1 const&, P2 const& );

public:
   Y operator()(X const& x)const{ return (*f_)(x,p1_,p2_);}
```

```
   Par2FncObj(Y (*g)(X const&, P1 const&, P2 const&), P1 const& p1,
      P2 const& p2)
   :p1_(p1),p2_(p2),f_(g){}
};

/////////////// Par2_1FncObj ////////////////////////////////////////

template <class X, class P1, class P2, class Y>
class Par2_1FncObj : public FncObj<X,Y>{

   const X x_;
   const P2 p2_;
   Y (* const f_)(X const&, P1 const&, P2 const& );

public:
   Y operator()(P1 const& p1)const{ return (*f_)(x_,p1,p2_);}
   Par2_1FncObj(Y (*g)(X const&, P1 const&, P2 const&), X const& x,
      P2 const& p2)
   :x_(x),p2_(p2),f_(g){}
};

/////////////// Par2_2FncObj ////////////////////////////////////////

template <class X, class P1, class P2, class Y>
class Par2_2FncObj : public FncObj<X,Y>{

   const X x_;
   const P1 p1_;
   Y (* const f_)(X const&, P1 const&, P2 const& );

public:
   Y operator()(P2 const& p2)const{ return (*f_)(x_,p1_,p2);}
   Par2_2FncObj(Y (*g)(X const&, P1 const&, P2 const&), X const& x,
      P1 const& p1)
   :x_(x),p1_(p1),f_(g){}
};

/////////////// Par3FncObj ////////////////////////////////////////

template <class X, class P1, class P2, class P3, class Y>
class Par3FncObj : public FncObj<X,Y>{

   const P1 p1_;
   const P2 p2_;
   const P3 p3_;
   Y (* const f_)(X const&,P1 const&,P2 const&,P3 const&);

public:
   Y operator()(X const& x)const{ return (*f_)(x,p1_,p2_,p3_);}
   Par3FncObj(Y (*g)(X const&,P1 const&,P2 const&,P3 const&),
```

```
   P1 const& p1, P2 const& p2, P3 const& p3)
   :p1_(p1),p2_(p2),p3_(p3),f_(g){}
};

/////////////// Par3_1FncObj ////////////////////////////////////////

template <class X, class P1, class P2, class P3, class Y>
class Par3_1FncObj : public FncObj<X,Y>{

   const X x_;
   const P2 p2_;
   const P3 p3_;
   Y (* const f_)(X const&,P1 const&,P2 const&,P3 const&);

public:
   Y operator()(P1 const& p1)const{ return (*f_)(x_,p1,p2_,p3_);}
   Par3_1FncObj(Y (*g)(X const&,P1 const&,P2 const&,P3 const&),
   X const& x, P2 const& p2, P3 const& p3)
   :x_(x),p2_(p2),p3_(p3),f_(g){}
};

/////////////// Par3_2FncObj ////////////////////////////////////////

template <class X, class P1, class P2, class P3, class Y>
class Par3_2FncObj : public FncObj<X,Y>{

   const X x_;
   const P1 p1_;
   const P3 p3_;
   Y (* const f_)(X const&,P1 const&,P2 const&,P3 const&);

public:
   Y operator()(P2 const& p2)const{ return (*f_)(x_,p1_,p2,p3_);}
   Par3_2FncObj(Y (*g)(X const&,P1 const&,P2 const&,P3 const&),
   X const& x, P1 const& p1, P3 const& p3)
   :x_(x),p1_(p1),p3_(p3),f_(g){}
};

/////////////// Par3_3FncObj ////////////////////////////////////////

template <class X, class P1, class P2, class P3, class Y>
class Par3_3FncObj : public FncObj<X,Y>{

   const X x_;
   const P1 p1_;
   const P2 p2_;
   Y (* const f_)(X const&,P1 const&,P2 const&,P3 const&);

public:
   Y operator()(P3 const& p3)const{ return (*f_)(x_,p1_,p2_,p3);}
```

```
   Par3_3FncObj(Y (*g)(X const&,P1 const&,P2 const&,P3 const&),
   X const& x, P1 const& p1, P2 const& p2)
   :x_(x),p1_(p1),p2_(p2),f_(g){}
};

/////////////// Par4FncObj ////////////////////////////////////////

template <class X, class P1, class P2, class P3, class P4, class Y>
class Par4FncObj : public FncObj<X,Y>{

   const P1 p1_;
   const P2 p2_;
   const P3 p3_;
   const P4 p4_;
   Y (* const f_)(X const&, P1 const&, P2 const& ,P3 const& ,P4 const&);

public:
   Y operator()(X const& x)const{ return (*f_)(x,p1_,p2_,p3_,p4_);}
   Par4FncObj(Y (*g)(X const&,P1 const&,P2 const&,P3 const&,P4 const&),
   P1 const& p1,P2 const& p2,P3 const& p3,P4 const& p4)
   :p1_(p1),p2_(p2),p3_(p3),p4_(p4),f_(g){}
};

/////////////// Par4_1FncObj ////////////////////////////////////////

template <class X, class P1, class P2, class P3, class P4, class Y>
class Par4_1FncObj : public FncObj<X,Y>{

   const X x_;
   const P2 p2_;
   const P3 p3_;
   const P4 p4_;
   Y (* const f_)(X const&, P1 const&, P2 const& ,P3 const& ,P4 const&);

public:
   Y operator()(P1 const& p1)const{ return (*f_)(x_,p1,p2_,p3_,p4_);}
   Par4_1FncObj(Y (*g)(X const&,P1 const&,P2 const&,P3 const&,P4 const&),
   X const& x,P2 const& p2,P3 const& p3,P4 const& p4)
   :x_(x),p2_(p2),p3_(p3),p4_(p4),f_(g){}
};

/////////////// Par4_2FncObj ////////////////////////////////////////

template <class X, class P1, class P2, class P3, class P4, class Y>
class Par4_2FncObj : public FncObj<X,Y>{

   const X x_;
   const P1 p1_;
   const P3 p3_;
   const P4 p4_;
```

```
   Y (* const f_)(X const&, P1 const&, P2 const& ,P3 const& ,P4 const&);

public:
   Y operator()(P2 const& p2)const{ return (*f_)(x_,p1_,p2,p3_,p4_);}
   Par4_2FncObj(Y (*g)(X const&,P1 const&,P2 const&,P3 const&,P4 const&),
   X const& x,P1 const& p1,P3 const& p3,P4 const& p4)
   :x_(x),p1_(p1),p3_(p3),p4_(p4),f_(g){}
};

/////////////// Par4_3FncObj ////////////////////////////////////////

template <class X, class P1, class P2, class P3, class P4, class Y>
class Par4_3FncObj : public FncObj<X,Y>{

   const X x_;
   const P1 p1_;
   const P2 p2_;
   const P4 p4_;
   Y (* const f_)(X const&, P1 const&, P2 const& ,P3 const& ,P4 const&);

public:
   Y operator()(P3 const& p3)const{ return (*f_)(x_,p1_,p2_,p3,p4_);}
   Par4_3FncObj(Y (*g)(X const&,P1 const&,P2 const&,P3 const&,P4 const&),
   X const& x,P1 const& p1,P2 const& p2,P4 const& p4)
   :x_(x),p1_(p1),p2_(p2),p4_(p4),f_(g){}
};

/////////////// Par4_4FncObj ////////////////////////////////////////

template <class X, class P1, class P2, class P3, class P4, class Y>
class Par4_4FncObj : public FncObj<X,Y>{

   const X x_;
   const P1 p1_;
   const P2 p2_;
   const P3 p3_;
   Y (* const f_)(X const&, P1 const&, P2 const& ,P3 const& ,P4 const&);

public:
   Y operator()(P4 const& p4)const{ return (*f_)(x_,p1_,p2_,p3_,p4);}
   Par4_4FncObj(Y (*g)(X const&,P1 const&,P2 const&,P3 const&,P4 const&),
   X const& x,P1 const& p1,P2 const& p2,P3 const& p3)
   :x_(x),p1_(p1),p2_(p2),p3_(p3),f_(g){}
};

#endif

template <class X1, class X2, class X3> class comp;

} // aux
```

```
//////////////////////////// class F<X,Y> ////////////////////////////////

// Defining a template class implementing the value interface out of the
// 'poor' class FncObj<X,Y> via the handle template CpmArrays::P<>

// No assumption on X
// Y has to define Y()

template <class X, class Y>
class F: public P< FncObj<X,Y> >{ // functions as a class
   // F stands for function

   typedef F<X,Y> Type;

   typedef P< FncObj<X,Y> > base;

   static Y f_const(X const& x, const Y& y0){x; return y0;}
   static Y f_default(X const& x){ x; return Y();}

public:
   virtual F<X,Y>* clone()const{ return new F(*this);}
   F<X,Y> toClnBase()const{ return *this;}
   CPM_ORDER
      // order is here a simple bookkeeping device
      // (based on position in memory space not on
      // function values, so no order relations in X
      // or Y are needed. This is sufficient for the set
      // template  S< F<X,Y> > to work.
      // Operations that deal with order and equality in Y
      // will be introduced in Fo and Fr.
   CPM_IO_V

      // neeeded for using V< <F<X,Y> >

// destructor, and assignment inherited from P<...>
// constructors
   F(FncObj<X,Y>* fop):base(fop){}
      // constructor from pointers to FncObj's. All deeper constructions
      // utilize this way. F<X,Y>(new DC(...)) where DC is a class
      // derived from FncObj<X,Y> and (...) are suitable constructor
      // arguments, is thus a correct construction of a F<X,Y> object.
      // The class DC should be immutable, i.e. all its
      // data are declared const.
      // That this is also directly possible for classes derived from
      // F<X,Y> it is essential to have this as automatic conversion
      // i.e. without an 'explicit' in front

   F(base const& g):base(g){}
      // upcast constructor
```

```
      // conversion needed that F<X,Y> is just a new name for
      // P< FncObj<X,Y> >

   F(void):base(new aux::Par0FncObj<X,Y>(f_default)){}
      // default constructor

   explicit F(Y const& y1):base(
      new aux::Par1FncObj<X,Y,Y>(f_const,y1)){}
      // constructor for constant function

// Construction from function pointers

   explicit F( Y (*f)(X const& )):
   base(new aux::Par0FncObj<X,Y>(f)){}

   explicit F( const Y (*f)(X const& )):
   base(new aux::ParcFncObj<X,Y>(f)){}

   explicit F( Y (*f)(X)):
   base(new aux::ParFncObj<X,Y>(f)){}

   explicit F( std::function<Y(X)> f):
   base(new aux::StdFunctionObj<X,Y>(f)){}
      // This C++11-facility allows the definition of F<X,Y> instances
      // from functional expressions in many variables in a convenient
      // manner: consider
      // Y f(X1 const& x1, X2 const& x2, X3 const& x3)
      // we would like to define a function X3 --> Y, x3 |--> f(a,b,x3)
      // for some a in X1 and b in X2.
      // We set:
      // #include <functional>
      // using std::bind;
      // using std::function;
      // using namespace std::placeholders;
      // function<Y(X3)> g(bind(f,a,b,_1))
      // and F<X3,Y> f3(g) or less verbose F<X3,Y> f3(bind(f,a,b,_1)).
      // In expression bind(f,a,b,_1) the placeholder is _1 and not _3
      // since it becomes the single first argument of function f3.
      // Making use of this method, we never need the classes
      // F<>,F1<>, .... F6<>
      // and
      // F1_1<>, F2_1<>, F2_2<>, F3_1<>, F3_2<>, F3_3<>, F4_1<>, F4_2<>,
      // F4_3<>, F4_4<>. Actually, the macro CPM_Fn allows to purge those
      // from the active code. With my present system the larger generality
      // of the method based on std::function and std::bind results in
      // slightly longer compilation time (10%) and longer run time (1%-2%)
      // a single test with program pala.

   //F<X,Y>& operator=( F<X,Y> const& f){ return *this=f;}
      // should not be needed
```

```
   F<X,Y>& operator=( Y (*f)(X) ){ return *this=F<X,Y>(f);}
      // added 2005-04-16. Now we can write also
      // F<R,R> sinCpm; sinCpm=sin;
      // instead of F<R,R> sinCpm(sin);
      // The = syntax is particularly convenient in arrays:
      // V< F<R,R> > fList(3);
      // fList[1]=sin; fList[2]=cos; fList[3]=exp;
      // Beware of F<R,R> sinCpm=sin; which can't be valid in one's
      // right mind
   F<X,Y>& operator=( Y (*f)(X const&)){ return *this=F<X,Y>(f);}
      // see previous function

// enabling 'function value taking' via '()' for the underlying FncObj
   virtual Y operator()(X const& x)const
   {
      return (base::operator()())(x);
      // this complicated-looking syntax is the price for using
      // operator()(void) for 'de-referencing' in 'pointer'-class P<>,
      // since the function to be defined itself is operator(), a fully
      // scope resolved name is to be used for an expression which in a
      // different scope could be simply (*this)()(x)
   }

   Y evl(X const& x)const{ return (this->base::operator()())(x);}
      // evl stands for evaluation

// turning *this into a constant-valued approximation (the most simple
// easy-to-evaluate approximating function)

   void makeConstant(X const& x0){ Y y0=(*this)(x0);*this=F<X,Y>(y0);}
      // the modified object is a constant-valued function taking
      // everywhere the value y0, which the un-modified one takes at
      // x=x0.

// concatenation (composition) of functions.

   template< class T>
   F<X,T> operator&(F<Y,T> const& g)const
      // x |--> y=(*this)(x) |--> t=g(y)=g((*this)(x))
      // thus (f&g)(x)=g(f(x))
      // notice that f & g : x |--> f(x) |--> g(f(x))
      // first action by f, second action by g thus n o t
      // (f&g)(x) = f(g(x))    wrong  !!!!
      // but
      // (f&g)(x) = g(f(x))     right !!!!
      // If one would write x.f for f(x) one had
      // x.(f&g) = x.f.g which looks much better
      // In the literature one finds f;g for f&g
   { return F<X,T>(new aux::comp<X,Y,T>(*this,g));}
```

```
   F<X,Y>& operator&=(F<Y,Y> const& g)
   { *this=operator&(g); return *this;}
      // mutating form of &, but type should not change, thus
      // T == Y needed

   template< class T>
   F<T,Y> circ(F<T,X> const& g)const
      // t|-->x=g(t)|-->y=(*this)(x)=(*this)(g(t))
      // thus (f.circ(g))(x)=f(g(x))
      // \circ is the LATEX name for the usual function
      // composition symbol
   { return F<T,Y>(new aux::comp<T,X,Y>(g,*this));}

   F<X,Y>& circ_(F<X,X> const& g){ *this=circ(g); return *this;}
      // mutating form of circ, but type should not change, thus
      // T == X needed

   void leftCombine(F<X,X> const& fL){ *this=fL&(*this);}
      // *this is changed into fL & *this

   void rightCombine(F<Y,Y> const& fR){ *this=operator&(fR);}
      // *this is changed into *this & fR

   virtual Word nameOf()const
   {
      Word wi="F<";
      Word wx=CpmRoot::Name<X>()(X());
      Word wy=CpmRoot::Name<Y>()(Y());
      return wi&wx&","&wy&">";
   }
};

template <class X, class Y>
bool F<X,Y>::prnOn(std::ostream&)const{return false;}

template <class X, class Y>
bool F<X,Y>::scanFrom(std::istream&){return false;}

template <class X, class Y>
Z F<X,Y>::com(F<X,Y> const& f)const
{
   return base::com(f);
}

namespace aux{

template <class X1, class X2, class X3>
class comp: public FncObj<X1,X3>{
   const F<X1,X2> f1;
   const F<X2,X3> f2;
```

```
public:
   comp(const F<X1,X2>& f1_, const F<X2,X3>& f2_):f1(f1_),f2(f2_){}
   X3 operator ()(const X1& x1)const{ return f2(f1(x1));}
};

} // aux

#ifdef CPM_Fn
//////////////////// class F1<> ////////////////////////////////////

template <class X, class P1, class Y>
class F1{ // functions with one parameter
    const P1 p1_;
    typedef F1<X,P1,Y> Type;
    CPM_INVAR(F1)
public:
explicit F1( P1 const& p):p1_(p){}
    F<X,Y> operator()( Y (*f)(X const&, P1 const&) )const;
};

// We explain the usage of F2<X,P1,P2,Y> which is representative
// for F1,F2,F3,..., F6 (see cpmf.h for F5 and F6).
// F2<X,P1,P2,Y> is a tool for constructing instances of
// F<X,Y> in a flexible manner from classical C-functions
// of prototype Y (*f)(X const&, P1 const&, P2 const&). The
// syntax is as follows:
// Assume definitions
// P1 p1(...);
// P2 p2(...);
// Y fc(X const& x, P1 const& p1, P2 const& p2){....}
// Then
// F<X,Y> f=F2<X,P1,P2,Y>(p1,p2)(fc);
// ( or F<X,Y> f(F2<X,P1,P2,Y>(p1,p2)(fc));
// defines a function f for which
// f(x)==fc(x,p1,p2)
// Thus f is 'the function fc with the parameters of types
// P1 and P2 given fixed values'.
// If the fixed values p1, p2 of these parameters are to be used for
// several classical functions of identical prototype,
// say fc, gc, hc, the the following syntax is economic:
// F2<X,P1,P2,Y> f2(p1,p2);
// F<X,Y> f=f2(fc);
// F<X,Y> g=f2(gc);
// F<X,Y> h=f2(hc);

// Addition 2011-03-10
// Tools to let any of the parameter play the role of the function
// argument:
// Consider the function fc as defined above. Sometimes one needs to
// consider it as, say, a function of the parameter p1 for fixed values
```

```
// of parameters x and p2. In such a case it is inconvenient to be forced
// to define a new function which has p1 in the first argument slot such
// as by the definition
//    Y fc1(P1 const& p1, X const& x, P2 const& p2)
//    {
//       return fc(x,p1,p2);
//    }
// To avoid this need we define a class template F2_1 which considers the
// first parameter slot as providing the function argument. Of course, the
// truly first slot is taken as a parameter slot in turn. Further, we
// define a class template F2_2 which takes the second parameter slot as
// the argument slot, and the other slots as parameter slots.
// This should be usable as a pattern to explain also the class templates
// F3_1, F3_2, F3_3, F4_1, F4_2, F4_3, F4_4.
// We don't define F5_{1,...,5} and F6_{1,...,6} since already 4 parameters
// is more than I ever needed in a context in which the Fn_m were useful.

// Addition 2015-06-11
// Having in mind a general function
// Y f(X1 const& x1, ... Xn const& xn)
// one could be interested in partitioning the index set {1,...,n} in an
// arbitrary way into two disjoint sets S1 and S2 and take the xi indexed
// by the second set as parameters and those indexed by the first set as
// arguments of a 'function of many variables'. This turned out to be very
// and even after replacing my caresian products by std::tupel I found no
// solution. The problem is that in these heterotypic arrays one has no
// for-loops over all components. Only costant expressions are allowed
// to access the components. If all types X1,...Xn had a common base class
// one could use pointers for the loops and one would have no problem.
// Taking this difficulties into account, one can appreciate the fact that
// the special case card(S1)=1 which we solved  by means of the functions
// Fn_m can be treated in a clear and elementary manner.

template <class X, class P1, class Y>
F<X,Y> F1<X,P1,Y>::operator()( Y (*f)(X const&, P1 const&) )const
{
   return F<X,Y>(new aux::Par1FncObj<X,P1,Y>(f,p1_));
}

////////////////////// class F1_1<> ////////////////////////////////////

template <class X, class P1, class Y>
class F1_1{
// as F1, but the first parameter is the function argument
    const X x_;
    typedef F1_1<X,P1,Y> Type;
    CPM_INVAR(F1_1)
public:
explicit F1_1( X const& x):x_(x){}
    F<P1,Y> operator()( Y (*f)(X const&, P1 const&) )const;
```

```
};

template <class X, class P1, class Y>
F<P1,Y> F1_1<X,P1,Y>::operator()( Y (*f)(X const&, P1 const&) )const
{
   return F<P1,Y>(new aux::Par1_1FncObj<X,P1,Y>(f,x_));
}

////////////////////// class F2<> ////////////////////////////////////

template <class X, class P1, class P2, class Y>
class F2{ // functions with two parameters
    const P1 p1_;
    const P2 p2_;
    typedef F2<X,P1,P2,Y> Type;
    CPM_INVAR(F2)
public:
    F2( P1 const& p1, P2 const& p2):p1_(p1),p2_(p2){}
    F<X,Y> operator()(Y (*f)(X const&, P1 const&, P2 const&))const;
};

template <class X, class P1, class P2, class Y>
F<X,Y> F2<X,P1,P2,Y>::operator()(Y (*f)(X const&, P1 const&,
   P2 const&) )const
{
   using namespace aux;
   return F<X,Y>(new Par2FncObj<X,P1,P2,Y>(f,p1_,p2_));
}

////////////////////// class F2_1<> ////////////////////////////////////

template <class X, class P1, class P2, class Y>
class F2_1{
// as F2, but the first parameter is the function argument
    const X x_;
    const P2 p2_;
    typedef F2_1<X,P1,P2,Y> Type;
    CPM_INVAR(F2_1)
public:
    F2_1( X const& x, P2 const& p2):x_(x),p2_(p2){}
    F<P1,Y> operator()(Y (*f)(X const&, P1 const&, P2 const&))const;
};

template <class X, class P1, class P2, class Y>
F<P1,Y> F2_1<X,P1,P2,Y>::operator()(Y (*f)(X const&, P1 const&,
   P2 const&) )const
{
   using namespace aux;
   return F<P1,Y>(new Par2_1FncObj<X,P1,P2,Y>(f,x_,p2_));
}
```

```
////////////////////// class F2_2<> ////////////////////////////////////

template <class X, class P1, class P2, class Y>
class F2_2{
// as F2, but the second parameter is the function argument
    const X x_;
    const P1 p1_;
    typedef F2_2<X,P1,P2,Y> Type;
    CPM_INVAR(F2_2)
public:
    F2_2( X const& x, P1 const& p1):x_(x),p1_(p1){}
    F<P2,Y> operator()(Y (*f)(X const&, P1 const&, P2 const&))const;
};

template <class X, class P1, class P2, class Y>
F<P2,Y> F2_2<X,P1,P2,Y>::operator()(Y (*f)(X const&, P1 const&,
   P2 const&) )const
{
   using namespace aux;
   return F<P2,Y>(new Par2_2FncObj<X,P1,P2,Y>(f,x_,p1_));
}

////////////////////// class F3<> ////////////////////////////////////

template <class X, class P1, class P2, class P3, class Y>
class F3{ // functions with three parameters
    const P1 p1_;
    const P2 p2_;
    const P3 p3_;
    typedef F3<X,P1,P2,P3,Y> Type;
    CPM_INVAR(F3)
public:
    F3( P1 const& p1, P2 const& p2, P3 const& p3):
    p1_(p1),p2_(p2),p3_(p3){}
    F<X,Y> operator()(Y (*f)(X const&, P1 const&, P2 const&,
       P3 const&) )const;
};

template <class X, class P1, class P2, class P3, class Y>
F<X,Y> F3<X,P1,P2,P3,Y>::operator()(Y (*f)(X const&, P1 const&,
   P2 const&, P3 const&) )const
{
   using namespace aux;
   return F<X,Y>(new Par3FncObj<X,P1,P2,P3,Y>(f,p1_,p2_,p3_));
}
////////////////////// class F3_1<> ////////////////////////////////////

template <class X, class P1, class P2, class P3, class Y>
class F3_1{
```

```
// as F3, but the first parameter is the function argument
    const X x_;
    const P2 p2_;
    const P3 p3_;
    typedef F3_1<X,P1,P2,P3,Y> Type;
    CPM_INVAR(F3_1)
public:
    F3_1( X const& x, P2 const& p2, P3 const& p3):
    x_(x),p2_(p2),p3_(p3){}
    F<P1,Y> operator()(Y (*f)(X const&, P1 const&, P2 const&,
       P3 const&) )const;
};

template <class X, class P1, class P2, class P3, class Y>
F<P1,Y> F3_1<X,P1,P2,P3,Y>::operator()(Y (*f)(X const&, P1 const&,
   P2 const&, P3 const&) )const
{
   using namespace aux;
   return F<P1,Y>(new Par3_1FncObj<X,P1,P2,P3,Y>(f,x_,p2_,p3_));
}

////////////////////// class F3_2<> ////////////////////////////////////

template <class X, class P1, class P2, class P3, class Y>
class F3_2{
// as F3, but the second parameter is the function argument
    const X x_;
    const P1 p1_;
    const P3 p3_;
    typedef F3_2<X,P1,P2,P3,Y> Type;
    CPM_INVAR(F3_2)
public:
    F3_2( X const& x, P1 const& p1, P3 const& p3):
    x_(x),p1_(p1),p3_(p3){}
    F<P2,Y> operator()(Y (*f)(X const&, P1 const&, P2 const&,
       P3 const&) )const;
};

template <class X, class P1, class P2, class P3, class Y>
F<P2,Y> F3_2<X,P1,P2,P3,Y>::operator()(Y (*f)(X const&, P1 const&,
   P2 const&, P3 const&) )const
{
   using namespace aux;
   return F<P2,Y>(new Par3_2FncObj<X,P1,P2,P3,Y>(f,x_,p1_,p3_));
}

////////////////////// class F3_3<> ////////////////////////////////////

template <class X, class P1, class P2, class P3, class Y>
class F3_3{
```

```
// as F3, but the third parameter is the function argument
    const X x_;
    const P1 p1_;
    const P2 p2_;
    typedef F3_3<X,P1,P2,P3,Y> Type;
    CPM_INVAR(F3_3)
public:
    F3_3( X const& x, P1 const& p1, P2 const& p2):
    x_(x),p1_(p1),p2_(p2){}
    F<P3,Y> operator()(Y (*f)(X const&, P1 const&, P2 const&,
       P3 const&) )const;
};

template <class X, class P1, class P2, class P3, class Y>
F<P3,Y> F3_3<X,P1,P2,P3,Y>::operator()(Y (*f)(X const&, P1 const&,
   P2 const&, P3 const&) )const
{
   using namespace aux;
   return F<P3,Y>(new Par3_3FncObj<X,P1,P2,P3,Y>(f,x_,p1_,p2_));
}

////////////////////// class F4<> /////////////////////////////////////

template <class X, class P1, class P2, class P3, class P4, class Y>
class F4{ // functions with four parameters
    const P1 p1_;
    const P2 p2_;
    const P3 p3_;
    const P4 p4_;
    typedef F4<X,P1,P2,P3,P4,Y> Type;
    CPM_INVAR(F4)
public:
    F4( P1 const& p1, P2 const& p2, P3 const& p3, P4 const& p4):
    p1_(p1),p2_(p2),p3_(p3),p4_(p4){}
    F<X,Y> operator()(Y (*f)(X const&, P1 const&,
       P2 const&, P3 const&, P4 const&))const;
};

template <class X, class P1, class P2, class P3, class P4,
     class Y>

F<X,Y> F4<X,P1,P2,P3,P4,Y>::operator()(Y (*f)(X const&, P1 const&,
   P2 const&, P3 const&, P4 const&))const
{
   using namespace aux;
   return
   F<X,Y>(new Par4FncObj<X,P1,P2,P3,P4,Y>(f,p1_,p2_,p3_,p4_));
}

////////////////////// class F4_1<> /////////////////////////////////////
```

```
template <class X, class P1, class P2, class P3, class P4, class Y>
class F4_1{
// as F4, but the first parameter is the function argument
    const X x_;
    const P2 p2_;
    const P3 p3_;
    const P4 p4_;
    typedef F4_1<X,P1,P2,P3,P4,Y> Type;
    CPM_INVAR(F4_1)
public:
    F4_1( X const& x, P2 const& p2, P3 const& p3, P4 const& p4):
    x_(x),p2_(p2),p3_(p3),p4_(p4){}
    F<P1,Y> operator()(Y (*f)(X const&, P1 const&,
       P2 const&, P3 const&, P4 const&))const;
};

template <class X, class P1, class P2, class P3, class P4,
     class Y>

F<P1,Y> F4_1<X,P1,P2,P3,P4,Y>::operator()(Y (*f)(X const&, P1 const&,
   P2 const&, P3 const&, P4 const&))const
{
   using namespace aux;
   return
   F<P1,Y>(new Par4_1FncObj<X,P1,P2,P3,P4,Y>(f,x_,p2_,p3_,p4_));
}

////////////////////// class F4_2<> ////////////////////////////////////

template <class X, class P1, class P2, class P3, class P4, class Y>
class F4_2{
// as F4, but the second parameter is the function argument
    const X x_;
    const P1 p1_;
    const P3 p3_;
    const P4 p4_;
    typedef F4_2<X,P1,P2,P3,P4,Y> Type;
    CPM_INVAR(F4_2)
public:
    F4_2( X const& x, P1 const& p1, P3 const& p3, P4 const& p4):
    x_(x),p1_(p1),p3_(p3),p4_(p4){}
    F<P2,Y> operator()(Y (*f)(X const&, P1 const&,
       P2 const&, P3 const&, P4 const&))const;
};

template <class X, class P1, class P2, class P3, class P4,
     class Y>

F<P2,Y> F4_2<X,P1,P2,P3,P4,Y>::operator()(Y (*f)(X const&, P1 const&,
```

```
   P2 const&, P3 const&, P4 const&))const
{
   using namespace aux;
   return
   F<P2,Y>(new Par4_2FncObj<X,P1,P2,P3,P4,Y>(f,x_,p1_,p3_,p4_));
}

////////////////////// class F4_3<> ////////////////////////////////////

template <class X, class P1, class P2, class P3, class P4, class Y>
class F4_3{
// as F4, but the third parameter is the function argument
    const X x_;
    const P1 p1_;
    const P2 p2_;
    const P4 p4_;
    typedef F4_3<X,P1,P2,P3,P4,Y> Type;
    CPM_INVAR(F4_3)
public:
    F4_3( X const& x, P1 const& p1, P2 const& p2, P4 const& p4):
    x_(x),p1_(p1),p2_(p2),p4_(p4){}
    F<P3,Y> operator()(Y (*f)(X const&, P1 const&,
       P2 const&, P3 const&, P4 const&))const;
};

template <class X, class P1, class P2, class P3, class P4,
     class Y>

F<P3,Y> F4_3<X,P1,P2,P3,P4,Y>::operator()(Y (*f)(X const&, P1 const&,
   P2 const&, P3 const&, P4 const&))const
{
   using namespace aux;
   return
   F<P3,Y>(new Par4_3FncObj<X,P1,P2,P3,P4,Y>(f,x_,p1_,p2_,p4_));
}

////////////////////// class F4_4<> ////////////////////////////////////

template <class X, class P1, class P2, class P3, class P4, class Y>
class F4_4{
// as F4, but the fourth parameter is the function argument
    const X x_;
    const P1 p1_;
    const P2 p2_;
    const P3 p3_;
    typedef F4_4<X,P1,P2,P3,P4,Y> Type;
    CPM_INVAR(F4_4)
public:
    F4_4( X const& x, P1 const& p1, P2 const& p2, P3 const& p3):
    x_(x),p1_(p1),p2_(p2),p3_(p3){}
```

```
    F<P4,Y> operator()(Y (*f)(X const&, P1 const&,
       P2 const&, P3 const&, P4 const&))const;
};

template <class X, class P1, class P2, class P3, class P4,
     class Y>

F<P4,Y> F4_4<X,P1,P2,P3,P4,Y>::operator()(Y (*f)(X const&, P1 const&,
   P2 const&, P3 const&, P4 const&))const
{
   using namespace aux;
   return
   F<P4,Y>(new Par4_4FncObj<X,P1,P2,P3,P4,Y>(f,x_,p1_,p2_,p3_));
}

#endif // CPM_Fn

} // namespace

#endif
```

15 cpmfo.h

```
//? cpmfo.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_FO_H_
#define CPM_FO_H_
/*

Purpose: Define classes describing function-like objects with
   order operations

*/
#include <cpmf.h>

namespace CpmFunctions{

// Base class for classes describing binary operations

namespace auxiliary{

template <class X, class Y>
class BINOPFUO: public FncObj<X,Y>{ // BINOP stands for Binary
// Operation
protected:
   const F<X,Y>  f1;
   const F<X,Y>  f2;

public:
   BINOPFUO(  const F<X,Y> & f1In, const F<X,Y> & f2In ):
   f1(f1In),f2(f2In){}
};

// Base class for classes describing monary operations
```

```
template <class X, class Y>
class MONOPFUO: public FncObj<X,Y>{

protected:
   const F<X,Y>  f1;

public:

    MONOPFUO( const F<X,Y> &  f1In): f1(f1In){}
};

// order-related operations
// Now it will be assumed that Y is ordered ( Y < Y and Y > Y)

// class MINFUO<>

template <class X, class Y>
class MINFUO: public BINOPFUO<X,Y>{

public:
   Y operator()(const X& x)const
   {
      Y y1=f1(x);
      Y y2=f2(x);
      return (y1 < y2 ? y1 : y2);
   }

   MINFUO(const F<X,Y> & f1In, const F<X,Y> & f2In):
      BINOPFUO<X,Y>(f1In,f2In){}
};

// class MAXFUO<>

template <class X, class Y>
class MAXFUO: public BINOPFUO<X,Y>{

public:
   Y operator()(const X& x)const
   {
      Y y1=f1(x);
      Y y2=f2(x);
      return (y1 > y2 ? y1 : y2);
   }

   MAXFUO(const F<X,Y>& f1In, const F<X,Y>& f2In):
      BINOPFUO<X,Y>(f1In,f2In){}
};

// setting function values
```

```
template <class X, class Y>
class SETVALUEFUO : public FncObj<X,Y>{   // changing function values
   // assumes that '==' is defined in X

   const F<X,Y>  f1;
   const X xs;
   const Y ys;

public:
   Y operator()(const X& x)const
   {
      if (x==xs) return ys; else return f1(x);
   }

   SETVALUEFUO(const F<X,Y>& g1, const X& x1, const Y& y1):
   f1(g1),xs(x1),ys(y1){}
};

} // auxiliary

////////////////////////////// class Fo<X,Y> ////////////////////////////

template <class X, class Y>
class Fo: public F<X,Y> { // version of F with order-related operations

   typedef Fo<X,Y> Type;

public:

   Fo(const F<X,Y>& g):F<X,Y>(g){}
      // upcast constructor

   Fo(const Fo<X,Y>& g):F<X,Y>(g){}
      // copy constructor

   Fo(void):F<X,Y>(){}
      // default constructor

   Fo(FncObj<X,Y>* fop):F<X,Y>(fop){}
      // constructor from pointers to FncObj

// Construction from function pointers

   Fo( Y (*f)(const X& )):F<X,Y>(f){}
   Fo( Y (*f)(X)):F<X,Y>(f){}

// Construtor for constant function

   explicit Fo(const Y& y1):F<X,Y>(y1){}
      // Functionality:
```

```
      // Y y=...; F<X,Y> f(y); X x=...; f(x)==y;
      // last statement always evaluates to 1, independent of x

   void setValue(const X& x0, const Y& y0);
      // non-constant member function which modifies the function value
      // at x=x0 to become y0
   virtual F<X,Y>* clone()const{ return new Fo(*this);}
   virtual Word nameOf()const
   {
      Word wi="Fo<";
      Word wx=CpmRoot::Name<X>()(X());
      Word wy=CpmRoot::Name<Y>()(Y());
      return wi&wx&","&wy&">";
   }
   CPM_ORDER
};

template <class X, class Y>
   // assumption Y < Y defined

Fo<X,Y> inf( const Fo<X,Y>& f1, const Fo<X,Y>& f2)
{
   using namespace auxiliary;
   return Fo<X,Y>(new MINFUO<X,Y>(f1,f2));
}

template <class X, class Y>
   // assumption Y < Y defined

Fo<X,Y> sup( const Fo<X,Y>& f1, const Fo<X,Y>& f2)
{
   using namespace auxiliary;
   return Fo<X,Y>(new MAXFUO<X,Y>(f1,f2));
}

// Order interface consistent but in a sense trivial.
// No assumtions on X,Y
// Does not take into account the function values but
// only the storage position.

template <class X, class Y>
Z Fo<X,Y>::com(const Fo<X,Y>& f)const
{
   return F<X,Y>::com(f);
}

} // namespace

#endif
```

16 cpmfr.h

```
//? cpmfr.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_FR_H_
#define CPM_FR_H_
/*
   Purpose: Define classes describing function-like objects with
      arithmetic operations

*/

#include <cpmfa.h>
#include <cpmvo.h>

namespace CpmFunctions{

   using namespace CpmStd;

   using CpmRoot::R;
   using CpmRoot::Word;

template <class X, class Y>

class Fr: public Fa<X,Y> { // version of Fa with rich interface

   static Z imax;
   static Z cpl;
      // means for implementing order and equality
      // by comparison of function values ( actually all
      // values would be needed for being mathematically exact)
      // Another way of looking at the matter is that imax and cpl define
      // an equivalence relation on Fr<X,Y> and order and equality are
      // defined exactly on the set of equivalunce classes. The larger
```

```
      // imax and cpl are, the smaller the equivalence classes

public:

   typedef Fr<X,Y> Type;

   Fr(void):Fa<X,Y>(){}
      // default constructor

   Fr(const F<X,Y>& g):Fa<X,Y>(g){}
      // upcast constructor

   Fr(const Fo<X,Y>& g):Fa<X,Y>(g){}
      // upcast constructor

   Fr(const Fa<X,Y>& g):Fa<X,Y>(g){}
      // upcast constructor, note Fa = Fa

   Fr(const Fr<X,Y>& g):Fa<X,Y>(g){}
      // copy constructor

   Fr(FncObj<X,Y>* fop):Fa<X,Y>(fop){}
      // constructor from pointers to FncObj

// Construction from function pointers

   Fr( Y (*f)(const X& )):Fa<X,Y>(f){}
   Fr( Y (*f)(X)):Fa<X,Y>(f){}

// Construtor for constant function

   Fr(const Y& y1):Fa<X,Y>(y1){}
      // takes always the value y1, irrespective of x

   Fr(CpmArrays::V<X> const& xl, CpmArrays::V<Y> const& yl);
      // Construction from a list of x-values and a list of corresponding
      // y-values. Makes a staircase-function since real interpolation
      // assumes R-related operations in Y which we don't want to assme
      // here.

   virtual F<X,Y>* clone()const{ return new Fr(*this);}

   CPM_IO
   CPM_TEST_X
   CPM_DESCRIPTORS
   CPM_CONJUGATION

   Y operator()(X const& x)const;
      // redefinition with better error handling
```

```
   R compare(Word top, Fr<X,Y> const& rhs)const;

};

template <class X, class Y>
Y Fr<X,Y>::operator()(const X& x)const
      // notice the * operator from class P<>
{
   using CpmArrays::P;
   try{
      return (P< FncObj<X,Y> >::operator()())(x);
   }
   catch(...){
      CpmRoot::Word wx=CpmRoot::Name<X>()(X());
      Y y=Y();
      CpmRoot::Word wy=CpmRoot::Name<Y>()(y);
      cpmerror("trouble in Y Fa<X,Y>::operator()(const X& x)const\
      for X type "&wx&" and Y type "&wy);
      return y; // to avoid warning
   }
}

// helper classes for implementation

namespace aux{
template <class X, class Y>
class CONJFUO : public FncObj<X,Y>{
   const Fr<X,Y> f;
public:
   CONJFUO(const Fr<X,Y>& f_):f(f_){}
   Y operator()(const X& x)const
   { return CpmRoot::conT<Y>(f(x));}
};

template <class X, class Y>
class RANDFUO : public FncObj<X,Y>{
   const Z i;
   const Z tvs;
   const Z memory;
public:
   RANDFUO(const Z& i_, Z tvs_, Z mem_ ):i(i_),tvs(tvs_),
      memory(mem_){}
   Y operator()(X const& x)const
      // never invoke ran() since then the random functions
      // depend on history
   {
   // would make operator() a mutating operator
   // this would have rather opaque implications
      Z iAct=(i==0 ? memory : i);
      Y ys;
```

```
      ys=CpmRoot::testT<Y>(ys,tvs);
      ys=CpmRoot::ranT<Y>(ys,iAct);
      Z j=CpmRoot::hashT<X>(x);
      if (j==0)
         return ys;
      else
         return CpmRoot::ranT<Y>(ys,j);
         // notice that for j==0 ran behaves not as a function
         // (value depends on how often the routine was called)
   }
};

template <class X, class Y>
class IPOLFUO: public FncObj<X,Y>{
   // IPOL stands for interpolation. However, here we don't really
   // interpolate, we use staircase approximation instead.
   // This needs no assumptions concerning Y.
   const CpmArrays::V<X> xl;
   const CpmArrays::V<Y> yl;
public:
   Y operator()(const X& x)const;
   IPOLFUO( const CpmArrays::V<X>& xlIn, const CpmArrays::V<Y>& ylIn ):
      xl(xlIn),yl(ylIn){}
};

template <class X, class Y>
Y IPOLFUO<X,Y>::operator()(X const& x)const
{
   Z i,n=xl.dim();
   Word loc("IPOLFUO<X,Y>::operator()");
   cpmassert(yl.dim()>=n,loc);
   CpmArrays::Vo<R> dis(n);
   for (i=1;i<=n;i++){
      X xd=x-xl[i];
      dis[i]=CpmRoot::absT<X>(xd);
   }
   Z j=dis.indInf();
   return yl[j];
}

} // aux

// end of helper classes

template <class X, class Y>
Z Fr<X,Y>::imax=8;

template <class X, class Y>
Z Fr<X,Y>::cpl=8;
```

```
template <class X, class Y>
Fr<X,Y>::Fr(const CpmArrays::V<X>& xl,
   const CpmArrays::V<Y>& yl)
   // a rather crude interpolation based on weighted means
   // of function values from the list
{
   *this=Fr(new aux::IPOLFUO<X,Y>(xl,yl));
}

template <class X, class Y>
Fr<X,Y> Fr<X,Y>::con(void)const
{
   return Fr<X,Y>(new aux::CONJFUO<X,Y>(*this));
}

template <class X, class Y>
Fr<X,Y> Fr<X,Y>::ran(Z i)const
{
   static Z mem=37;
   if (i==0) mem++;
   return Fr<X,Y>(new aux::RANDFUO<X,Y>(i,cpl,mem));
}

template <class X, class Y>
Word Fr<X,Y>::nameOf(void)const
{
   Word wi="Fr<";
   X x; Y y;
   Word wx=CpmRoot::Name<X>()(x);
   Word wy=CpmRoot::Name<Y>()(y);
   return wi&wx&","&wy&">";
}

template <class X, class Y>
Word Fr<X,Y>::toWord(void)const
{
   ostringstream ost;
   prnOn(ost);
   return Word(ost.str());
}

template <class X, class Y>
Fr<X,Y> Fr<X,Y>::test(Z i)const
{
   Y y0;
   Y yt=CpmRoot::testT<Y>(y0,i);
   Fr<X,Y> res(yt);
   imax=i;
   cpl=i;
   return res;
```

```
}

template <class X, class Y>
bool Fr<X,Y>::prnOn(ostream& out)const
{
   out<<endl<<"// printing of "<<nameOf()<<endl;
   out<<endl<<"// Number of (x,y)-pairs: "<<endl;
   out<<imax<<endl;
   X x;
   Y y;
   X xs=CpmRoot::testT<X>(x,cpl);
   for (Z i=1;i<=imax;i++){
      x=CpmRoot::ranT<X>(xs,i);
      y=(*this)(x);
      bool b;
      out<<endl<<"// value pair number "<<i<<" of "<<imax;
      out<<endl<<"// x = "<<endl;
      b=CpmRoot::prnT<X>(x,out);
      if (!b) return false;
      out<<endl<<"// y = "<<endl;
      b=CpmRoot::prnT<Y>(y,out);
      if (!b) return false;
   }
   return true;
}

template <class X, class Y>
bool Fr<X,Y>::scanFrom(istream& in)
// reading value pairs and creating an interrpolated function
// from a file that may contain comments
{
   static const Z readLimit=512;
   Z nList;
   if (!CpmRoot::read(nList,in)) return false;
   cpmassert(nList>0,"Fr<X,Y>::scanFrom");
   cpmassert(nList<=readLimit,"Fr<X,Y>::scanFrom");
   CpmArrays::V<X> xl(nList);
   CpmArrays::V<Y> yl(nList);
   for (Z i=1;i<=nList;i++){
      if (!CpmRoot::scanT<X>(xl[i],in)) return false;
      if (!CpmRoot::scanT<Y>(yl[i],in)) return false;
   }
   *this=Fr<X,Y>(xl,yl);
   return true;
}

template <class X, class Y>
Z Fr<X,Y>::hash(void)const
// returns a key value to *this. This should be not
// expensive to calculate. Therfore the number of function
```

```
// evaluation is restricted by imax
{
   Z h=0;
   X x,xs;
   Y y;
   xs=CpmRoot::testT<X>(x,cpl);
   for (Z i=1;i<=imax;i++){
      x=CpmRoot::ranT<X>(xs,i);
      y=(*this)(x);
      h^=CpmRoot::hashT<Y>(y);
         // bit operation since += may give overflow
   }
   return h;
}

template <class X, class Y>
R Fr<X,Y>::absSqr(void)const
// returns as an absolute value a mean absolute value of function values
// where the arguments are chosen as a random sequence.
{
try{
   X x,xs;
   Y y;
   xs=CpmRoot::testT<X>(x,cpl);
   R res=0;
   for (Z i=1;i<=imax;i++){
      x=CpmRoot::ranT<X>(xs,i);
      y=(*this)(x);
      R a=CpmRoot::absT<Y>(y);
      res+=a*a;
   }
   cpmassert(imax>0,"Fr<X,Y>::absSqr");
   res/=imax;
   return cpmsqrt(res);
}
catch(...){
   cpmerror("trouble in R Fr<X,Y>::abs(void)const");
   return 0; // to avoid warning
}
}

template <class X, class Y>
R Fr<X,Y>::abs(void)const
{
   return cpmsqrt(absSqr());
}

template <class X, class Y>
R Fr<X,Y>::dis(const Fr<X,Y>& x)const
{
```

```
   Fr<X,Y> diffVector=*this;
   diffVector-=x;
   R da=diffVector.abs();
   R xa=x.abs();
   R ya=abs();
   return CpmRoot::disDefFun(xa,ya,da);
}

} // namespace

#endif
```

17 cpmgreg.h

```
//? cpmgreg.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_GREG_H_
#define CPM_GREG_H_

/*

    Functions for handling Gregorian calendar dates

*/

#include <cpmangle.h>

namespace CpmTime{ // Quantities related to calendar, date, time, clock

using namespace CpmStd;
//using CpmSystem::R_;
using CpmRoot::Z;
using CpmRoot::R;
using CpmRoot::Word;
using CpmGeo::Angle;

//////////////////////////// enum TimeScale ////////////////////////////////

enum TimeScale {TD, UT}; // presenly not used in this general-purpose
   // (non-astronomical) version of Greg in order not to need the more
   // astronomical files.

   // TD = Temps dynamique, UT = universal time

////////////////////////////// struct TimeStyle ////////////////////////////
```

```
struct TimeStyle{ // how to interprete time with respect to the globe
   TimeScale tsc;
   Angle lambdaOfTimeMeridian;
   TimeStyle(void):tsc(UT),lambdaOfTimeMeridian(Angle()){}
   TimeStyle(TimeScale s, const Angle& a):tsc(s),
      lambdaOfTimeMeridian(a){}
};

class Greg;
R operator -(const Greg& d1, const Greg& d2);

////////////////////////////// class Greg ////////////////////////////////

class Greg{  // time, date, and Gregorian Calendar
   // An instance of class Greg determines a point in time
   // coded according to the Gregorgian calendar (and according to the
   // Julian calendar for dates prior to the introduction of the
   // Gregorian calendar). No reference to a particular time scale (UT
   // or TD) or to a time zone (e.g. MEZ or MESZ ) is implied

private:

   static Z yearOffset;
      // The 'century' which is understood for 2 decimal year input
      // e.g. 2000 for dates in the 21th century
      // May be changed via setCentury(). Presently initialized as 2000.

   static const R mjdOffset;
      // JD=MJD+mjdOffset, mjdOffset=2400000.5

protected:

// primary (independent) information
   Z y;         // year
   Z m;         // month
   Z d;         // day
   Z h;         // hour
   Z min;       // minute
   R s;         // second

// secondary (dependent) information (function of the primary
// information)
   R mjd;         // analog to the MJD without reference to a concrete
               // time scale such as UT or TD
   void update(void);
      // updates dependent information (i.e. mjd)

   void makeDate(void);
      // adjusts the rest of data members to present value of mjd
```

```
public:

   typedef Greg Type;
   CPM_IO
   CPM_NAM_V(Greg)
   virtual Greg* clone()const{ return new Greg(*this);}

// parts of the day (the unit) which are often needed

   static const R hour;
   static const R minute;
   static const R second;

   static R mjd_(Z y, Z m, Z d);
      // Modified from Montenbruck, Pfleger: Astronomy on the Personal
      // Computer,
      // 4th edition, p. 15
      // Notice that mjd_==0 <==> y=1858 m=11 d=17

   static Z trunc(R x);
      // See O. Montenbruck, Grundlagen der Ephemeridenrechnung S. 49

   static void caldat(Z mjd, Z& yy, Z& mm, Z& dd);
      // From Montenbruck, Pfleger: Astronomie mit dem Personal Computer
      // p. 13
      // Translated from PASCAL to C

   void setCentury( Z century)
      // setCentury(20) sets the value to be added to a short
      // year, such as 95, to 1900, thus resulting in 1995.
      // Notice that this coincides with the conventional enumeration
      // of centuries.
      // Century should be positive, we are not interested in using
      // Gregorian Calendar in times much prior to its introduction !
      {
         yearOffset=(century-1)*100;
      }

   Greg(void);
      // Default constructor, creates 2000-1-1 0:0:0

   Greg(Z year, Z mm, Z dd);
      // mm has to be in the range [1,12]
      // dd is arbitrary (e.g. 94-12-31 = 95-1-0 = 95-2-(-31) )
      // if year<99 it will be understood as year+yearOffset.

   explicit Greg(R date){ setDate(date,false);}
     // date is assumed to be of the form year.mmdd
     // where year is the integer year number e.g. 2006 or
     // 1905 (no shortened numbers like 50 for 1950
```

```
     // or 6 for 2006 !) Only the first 4 decimal places get
     // used (no fractions of a day).

   Greg(R date, R watch){ setDate(date,false);setWatch(watch);}
      // Creates a Greg instance from date=YYYY.MMDD and
      // watch=HH.MMSSsss... (see setWatch() for details).

// getting calendar data, always clear from code

   Z getYear(void)const{ return y;}

   Z getMonth(void)const{ return m;}

   Z getDay(void)const{ return d;}

   Z getHour(void)const{ return h;}

   Z getMinute(void)const{ return min;}

   R getSecond(void)const{ return s;}

   R getMJD(void)const{ return mjd;}

   R getJD(void)const{ return mjd+mjdOffset;}

   Word getWeekDay(void)const;

   R yearsSince1900(void)const{ return (mjd-33281.)/R(365.2422)+50.;}
   R yearsSince2000(void)const{ return (mjd-51544.)/365.2422;}
   R years(void)const{ return yearsSince2000()+2000;}
      // Convenient unit for time axes which go over longer spans of time
      // and which are related to history.

   Word getDate(bool verbose=true)const;
      // "yyyy-mm-ddThh:mm:ssZ" ( ISO6801 time stamp )
      // a compact representation of a date in which the newline commands
      // of printOut() would cause trouble

   Word getCode(Z n=14)const;
      // returns a characterization of a point in time which
      // is compact enough to be used as an appendix to file names.
      // It consists of the n trailing characters of the string
      // yyyymmddhhmmss. So the default value for n means that
      // a condensed form of the date will be given. For
      // n=8 one gets a characterisation of the time within the
      // present day.

   Word getCodeWord(Z n, Z p=5);
      // returns a sequence of n digits which is determined by the
      // instance of call and may be used as random signature
```

```
      // method: multiply mjd by 10^p, form the integer part
      // and take the last n digits of the result. Thus, the larger
      // p, the faster the result changes with time. The frequency
      // of change is 10^p per day. Thus p=5 means 10^5 changes
      // in 86400 s, i.e. 1.157 changes per second. This is a
      // reasonable time resolution for differentiating between
      // susequent runs of a program.

// setting Greg according to numbered days input (Julian date and
//   modified Julian date

   void setMJD(R mjd){ this->mjd=mjd; makeDate();}
      // adjusts *this according to MJD input.
      // In constructor form, this function would enable automatic
      // conversion from Rd to Greg and finally would lead to
      // Greg + Greg being defined (since Greg + R is defined). This
      // is not desirable, only Greg - Greg is OK.

   void setMJD(R mjd, const TimeStyle& ts);
      // adjusts *this according to MJD input. The correspondence
      // between what the function does and the values
      // UT or TD of the component ts.tsc is logical if we
      // consider mjd as referring to TD. Then ts.tsc=UT means that
      // a deltaT subtraction has to be done to get a result in
      // UT or the civil time of a time zone specified by
      // ts.lambdaOfTimeMeridian

   void setJD(R jd){ mjd=jd-mjdOffset; makeDate();}
      // adjusts *this according to JD input.

   R toMJDTD(const TimeStyle& ts)const;
      // returns MJD (as TD) from the Greg instance *this where ts
      // tells how to interprete *this
      // (e.g. as civil time of a particular time zone)

   void now(void);
      // sets *this according to the computer's idea of Greenwich
      // mean time (Z, GMT0) at the moment of function call. Notice that
      // shifting this point in time according to needs can easily be done
      // by the arithmetics of Greg, so that no shift argument is needed
      // for this function.
      // e.g.
      // TimeStyle ts=...;
      // Greg g;
      // g.now();
      // g+=10*Greg::minute;
      // R t=g.toMJDTD(ts);
      // Earth e;
      // e.update(t);
      // will represent the state of the earth 10 minutes after the
```

```
      // function call if ts properly reflects the characteristics of the
      // computer clock

friend R operator -(const Greg& d1, const Greg& d2);
      // time span in days, notice that a corresponding + operator
        // makes no sense

   Greg& operator +=(R t);
      // shifting a point in time by t days

   Greg& nextMonth_();
      // shifting a point in time to the same day, minute, second
      // but one month in calander later (shift may be 28 days,
      // 30 days or 31 days according to the calender month we are
      // in). Name ends in '_' since this is a mutating function.

   Greg& nextDay_(){ mjd+=1.; makeDate(); return *this;}
      // shifting a point in time to by one day
      // in). Name ends in '_' since this is a mutating function.

   Greg& operator -=(R t);
      // shifting a point in time by -t days

   Greg operator +(R t){ Greg res=*this; return res+=t;}

   Greg operator -(R t){ Greg res=*this; return res-=t;}

   void setDate(R date, bool yShort=true);
      // Sets the values of y, m, d from date=Year.MMDD .
      // If yShort=true and Year<99 the integer number
      // yearOffset will be added to y.

   void setWatch(R watch);
      // Sets the values of h, min, s from watch=HH.MMSSsss...
      // where SS.sss... is a number of seconds possibly together with
      // decimal fractions of a second.

   bool readFormatted(istream& inStream);
   bool writeFormatted(ostream& outStream)const;
};

} // namespace

#endif
```

18 cpmgreg.cpp

```
//? cpmgreg.cpp
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


// Inserting diagnostic messages in function bodies of this file may causes
// problems.
// Functions may be called in a situation where the log file streams are
// not yet created.

#include <time.h>
#include <iomanip>

#if !defined(WIN32)
   #include <stdlib.h>
#endif

#include <cpmgreg.h>

using CpmRoot::Z;
using CpmRoot::R;
using CpmRoot::Word;
using CpmRoot::toWord;
using CpmRoot::toDouble;

using CpmTime::Greg;
using CpmTime::TimeStyle;

using namespace CpmStd;

R Greg::mjd_(Z y, Z m, Z d)
{
   if (m<=2){
      m+=12;
      y-=1;
```

```
   }
   R a=10000.*y+100.*m+d;
   Z b;
   if (a<=15821004.1){
      b=-2+(y+4716)/4-1179;
   }
   else{
      b=y/400-y/100+y/4;
   }
   a=365.0*y-679004.0;
   R res=a+b+floor(30.6001*(m+1))+d;
   return res;
}

Z Greg::trunc(R x)
{
   return CpmRoot::toZ(x,true);
}

void Greg::caldat(Z mjd, Z& yy, Z& mm, Z& dd)
{
   Z b,d,f;
   R jd,jd0,c,e;
   jd=2400000.5+mjd;
   jd0=trunc(jd+0.5);
   if (jd0<2299161.0){
      b=0;
      c=jd0+1524.0;
   }
   else{
      b=trunc((jd0-1867216.25)/36524.25);
      c=jd0+(b-trunc(b/4))+1525.0;
   }
   d=trunc((c-122.1)/365.25);
   e=365.0*d+trunc(d/4);
   f=trunc((c-e)/30.6001);
   dd=trunc(c-e+0.5)-trunc(30.6001*f);
   mm=f-1-12*trunc(f/14);
   yy=d-4715-trunc((7+mm)/10);
}

Z Greg::yearOffset=2000;

const R Greg::mjdOffset=2400000.5;

const R Greg::hour=1./24;

const R Greg::minute=Greg::hour/60;

const R Greg::second=Greg::minute/60;
```

```
void Greg::update(void)
{
   mjd=mjd_(y,m,d)+R(h)*hour+R(min)*minute+s*second;
}

void Greg::makeDate(void)
{
   Z mjdInt=cpmtoz(mjd);
   R rest=mjd-mjdInt;
   caldat(mjdInt,y,m,d); // now y,m,d are valid
   R h1=rest*24;
   h=trunc(h1);
   R min1=(h1-h)*60;
   min=trunc(min1);
   s=(min1-min)*60;
}

Greg::Greg(void)
{
   y=2000;
   m=1;
   d=1;
   h=0;
   min=0;
   s=0.;
   mjd=51544;
}

Greg::Greg(Z yy, Z mm, Z dd):y(yy),m(mm),d(dd),h(0),min(0),s(0)
{
   if (m<1 || m>12) cpmerror("Greg","month must be 1,...,12");
   update();
   if (d<1 || d>28) makeDate();
}

Greg& Greg::nextMonth_()
{
   m++;
   if (m>12){
      y++;
      m=1;
   }
   update();
   if (d>28) makeDate(); // this makes a valid date even when called
   // e.g. for March 31, where April 31 would not be a correct
   // date.
   return *this;
}
```

```
bool Greg::prnOn(ostream& str)const
{
   cpmwat;
   cpmp(y);
   cpmp(m);
   cpmp(d);
   cpmp(h);
   cpmp(min);
   cpmp(s);
   return true;
}

bool Greg::scanFrom(istream& str)
{
   cpms(y);
   cpms(m);
   cpms(d);
   cpms(h);
   cpms(min);
   cpms(s);
   update();
   return true;
}

Word Greg::getDate(bool verbose)const
{
   using std::setw;
   using std::setfill;
   ostringstream ost;
   if (verbose) ost<<"yyyy-mm-ddThh:mm:ss ";
   ost<<setw(4)<<setfill('0')<<y<<"-"
      <<setw(2)<<setfill('0')<<m<<"-"
      <<setw(2)<<setfill('0')<<d<<"T"
      <<setw(2)<<setfill('0')<<h<<":"
      <<setw(2)<<setfill('0')<<min<<":"
      <<setw(2)<<setfill('0')<<toDouble(s)<<"Z";
   return Word(ost.str());
}

Word Greg::getCode(Z n)const
{
   using std::setw;
   using std::setfill;
   ostringstream ost;
   ost<<setw(4)<<setfill('0')<<y
      <<setw(2)<<setfill('0')<<m
      <<setw(2)<<setfill('0')<<d
      <<setw(2)<<setfill('0')<<h
      <<setw(2)<<setfill('0')<<min
      <<setw(2)<<setfill('0')<<toDouble(s);
```

```
   return Word(ost.str()).tail(n);
}

bool Greg::readFormatted(istream& inStream)
{
   cout<<endl<<
   "Entering a point in time in terms of the Gregorian calendar";
   R d,h;
   cout<<"\n Enter the date in the format yy.mmdd or yyyy.mmdd:  ";
   inStream>>d;
   cout<<"\n Enter the time in the format hh.mmss:  "; inStream>>h;
   setDate(d);
   setWatch(h);
   update();
   return true;
}

bool Greg::writeFormatted(ostream& str)const
{
   Word yS=Word::write(y,4);
   Word mS=Word::write(m,2);
   Word dS=Word::write(d,2);
   Word GregS=yS&"-"&mS&"-"&dS&" ";
   Word hS=Word::write(h,2);
   Word minS=Word::write(min,2);
   Word sS=toWord(s,"%2.1f");
   Word watchS=hS&":"&minS&":"&sS&" Z";
   Word res=GregS&watchS;
   str<<res;
   //return (str!=0);
   if (!str) return false;
   else return true;
}

R CpmTime::operator - (const Greg& g1, const Greg& g2)
{
   return g1.mjd-g2.mjd;
}

Greg& Greg::operator +=(R t)
{
   mjd+=t;
   makeDate();
   return *this;
}

Greg& Greg::operator -=(R t)
{
   mjd-=t;
   makeDate();
```

```
   return *this;
}

void Greg::setDate(R date, bool yShort)
{
   const R eps=0.0001;
   Z sig=(date >=0. ? 1 : -1);
   date*=sig; // now date is positive
   y=cpmtoz(date);
   date-=y; // fractional part of the absolute value of date
   y*=sig; // now the year carries its sign
   if (yShort){
      if (y<=99) y+=yearOffset;
   }
   date*=100;
   m=cpmtoz(date+eps);
   date-=m;
   date*=100;
   d=cpmtoz(date+eps);
   h=0;
   min=0;
   s=0.;
   update(); // added 99-11-27
}

void Greg::setWatch(R watch)
{
   const R eps=0.0001;
   if (watch<0) cpmerror("Greg::setWatch(R_): argument should be >=0");
   h=cpmtoz(watch+eps);
   watch-=h; // fractional part of the absolute value of watch
   watch*=100;
   min=cpmtoz(watch+eps);
   watch-=min;
   watch*=100;
   s=watch;
   update(); // added 99-11-27
}

Word Greg::getWeekDay(void)const
{
   Z weekDay=cpmtoz(getMJD());
   weekDay=weekDay%7;
   Word res;
   switch (weekDay){
      case 0: res="Wendsday";break;
      case 1: res="Thursday";break;
      case 2: res="Friday";break;
      case 3: res="Saturday";break;
      case 4: res="Sunday";break;
```

```
      case 5: res="Monday";break;
      case 6: res="Tuesday";break;
   }
   return res;
}

void Greg::now( void)
{
   static bool firstrun=true;
   if (firstrun){
   #if defined(WIN32)
      _putenv("TZ=GMT0");
      _tzset();
   #else
      char tz[7]={'T','Z','=','G','M','T','0'};
      putenv(tz);
      tzset(); // needs stdlib.h
   #endif
      firstrun=false;
   }
   time_t t;
   time(&t);
   tm* all;
   all=gmtime(&t);
   // due to putenv() this gives the normal conversion
      // without a time zone shift
   s=all->tm_sec;
   min=all->tm_min;
   h=all->tm_hour;
   d=all->tm_mday;
   m=1+all->tm_mon;
   y=all->tm_year; // comes out as 99 for 1999, and 100 for 2000

   if (y>=90 /* && y<=99 */) y+=1900; // Windows NT gives 100 in 2000
      // this means that the rule to add 1900 is not changed
    update();
    if (cpmverbose>=4){
        cout<<endl<<"s="<<s;
        cout<<endl<<"min="<<min;
        cout<<endl<<"h="<<h;
        cout<<endl<<"d="<<d;
        cout<<endl<<"m="<<m;
        cout<<endl<<"y="<<y;
        cout<<endl<<"mjd="<<mjd;
    }
}

R Greg::toMJDTD(const TimeStyle& ts)const
{
```

```
   R shift=ts.lambdaOfTimeMeridian.toDeg(0);
    shift/=360.; // now it is in days
    R res=mjd;
    res+=shift; // now res can be compared with MJD of UT or TD
        // i.e. there is no longer a time zone contribution
    return res;
}

void Greg::setMJD(R mjdI, const TimeStyle& ts)
{
    mjd=mjdI;

   R shift=ts.lambdaOfTimeMeridian.toDeg(0);
    shift/=360.; // now shift is in days
    mjd-=shift; // now referring to the local meridian
    makeDate();
}

Word Greg::getCodeWord(Z n, Z p)
{
   now();
   R f=pow(10.,p);
   Z cod=cpmtoz(mjd*f);
   Word res=cpmwrite(cod);
   return res.tail(n);
}
```

19 cpminterfaces.h

```
//? cpminterfaces.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_INTERFACES_H_
#define CPM_INTERFACES_H_
/*

   Description: Declaration macros for interfaces for
      CPM classes. No macro argument, assuming that the
      client class will typedef her name as Type; some macros
      assume that the client class also defines ScalarType
      and CloneBase

*/
#include <cpmnumbers.h>
   // includes cpmdefinitions.h
#include <cpmbasicinterfaces.h>
#include <cpmmpi.h>
   // contains valid (though trivial) declarations even for
   // CPM_USE_MPI not defined

// Notice that using types Z and R in macros is only safe as
// CpmRoot::Z and CpmRoot::R since they might be used in a context
// where a corresponding using directive is not available.

// defining member functions send and rec(ieve) for communication between
// processes if the message passing interface library is being used
// (for parallel programming)
//using CpmMPI::Com;
// Definition of send builds on function prnOn, definition of rec builds
// on function scanFrom. The macros CPM_MPI_... will always be used after
// macros which declare these functions prnOn and scanFrom.
```

```
#if defined(CPM_USE_MPI)

   #define CPM_MPI_O\
   bool send(Z dest, CpmRoot::Z tag, CpmMPI::Com comm=CpmMPI::Com()\
   )const\
      { \
         std::ostringstream ost;\
         CpmRoot::Z wrtPrcMem=CpmRoot::wrtPrc;\
         CpmRoot::wrtPrc=18;\
         bool b=prnOn(ost);\
         CpmRoot::wrtPrc=wrtPrcMem;\
         return b ? comm.sendStr(ost.str(),dest,tag): false;\
      }

   #define CPM_MPI_I\
      bool rec(CpmRoot::Z source, CpmRoot::Z tag, CpmMPI::Com\
       comm=CpmMPI::Com())\
      {\
         std::string s;\
         bool res=comm.recStr(s,source,tag);\
         std::istringstream ist(s);\
         bool b=scanFrom(ist);\
         return b ? res : false;\
      }

#else // !defined(CPM_USE_MPI)

   #define CPM_MPI_O\
   bool send(CpmRoot::Z dest, CpmRoot::Z tag, CpmMPI::Com comm=\
   CpmMPI::Com())const{ dest; tag; comm; return true;}

   #define CPM_MPI_I\
      bool rec(CpmRoot::Z source, CpmRoot::Z tag, CpmMPI::Com\
       comm=CpmMPI::Com()){ source; tag; comm; return true;}

#endif

// input and output functions; the function read is the common interface
// to both built-in types and Cpm classes to read from commented files
// ( which does not work for R and Z via >>)

// all IMPL-MACROS get completed by their corresponding
// MPI-functionality.
// Then it is guaranteed that all classes define proper inter-process
// communication.
// Notice that for non-class types like Z and R, there is no re-definition
// of operators >> an <<.

#define CPM_IO_IMPL\
   friend std::ostream& operator<<(std::ostream& out, Type const& x)\
```

```
      { CpmRoot::IO<Type>().o(x,out); return out;}\
   friend std::istream& operator>>(std::istream& in, Type& x)\
      { CpmRoot::IO<Type>().i(x,in); return in;}\
      CPM_MPI_O\
      CPM_MPI_I

// declaration macros without public qualifications
// with and without virtual qualification.

#define CPM_IO_V\
   virtual bool prnOn(std::ostream&)const;\
   virtual bool scanFrom(std::istream&);\
   CPM_IO_IMPL

#define CPM_IO\
   bool prnOn(std::ostream&)const;\
   bool scanFrom(std::istream&);\
   CPM_IO_IMPL

// sometimes one needs the O-part (= output part) in isolation:

#define CPM_O_IMPL\
   friend std::ostream& operator<<(std::ostream& out, Type const& x)\
      { CpmRoot::IO<Type>().o(x,out); return out;}\
   CPM_MPI_O

#define CPM_O_V\
   virtual bool prnOn(std::ostream&)const;\
   CPM_O_IMPL

#define CPM_O\
   bool prnOn(std::ostream&)const;\
   CPM_O_IMPL

// help to unify defining member functions
//    bool prnOn(ostream& str)const;
//    bool scanFrom(istream& str);
// in classes with many data members that define prnOn and scanFrom.
// Notice that the ; is missing in the definition of cpmp and cpms,
// so that the call of those macros has to provide it.
// These lests these macros behave just as if they were functions.
// E.g.
/*
   bool prnOn(ostream& str)const
   {
      cpmp(x1);
      cpmp(x2);
      cpmp(x3);
      return true;
   }
```

```
   bool scanFrom(istream& str)
   {
      cpms(x1);
      cpms(x2);
      cpms(x3);
      return true;
   }
*/
// Defining the nameOf function conveniently, no ';' at the end.
#define CPM_NAM(X) Word nameOf()const{ return Word(#X);}
#define CPM_NAM_V(X) virtual Word nameOf()const{ return Word(#X);}
// Declarations that need a ';' behind. The rule is that any CPM_...
// needs no ; behind it, but cpm... needs it.
#define cpmwt(X) if (!CpmRoot::writeTitle((X),str)) return false
   // wt for 'write title'
#define cpmwat if (!CpmRoot::writeTitle(nameOf().str(),str)) return false
   // wt for 'write automatic title', needs no argument.
#define cpmwbt if (!CpmRoot::writeTitle((nameOf()&" begin").str(),str))\
 return false
   // wbt for 'write begin title'
#define cpmwet if (!CpmRoot::writeTitle((nameOf()&" end").str(),str))\
 return false
   // wet for 'write end title', needs no argument.
#define cpmp(X) if (!CpmRoot::prnT((X),str)) return false
   // p for print
#define cpms(X) if (!CpmRoot::scanT((X),str)) return false
   // s for scan
#define cpmrh(X) rch.read(sec,#X,X)
   // Regularly occurring idiom for reading from a RecordHandler
   // read handler. Stops the program if the quantity cannot be
   // read.
#define cpmrhf(X) rch.read(sec,#X,X,1)
   // Regularly occurring idiom for reading from a RecordHandler
   // read handler, option pedantic = 1, so that only a warning
   // gets issued if the quantity cannot be read.
   // 'f' for 'floppy'
#define cpmr(X) rc.get(sec,#X,X,1)
   // Regularly occurring idiom for reading from a read handler
   // of type Record. Never stops the program if the quantity
   // cannot be read. Issues a warning however.
   // Introduced 2010-08-30.
#define cpmrf(X) rc.get(sec,#X,X,0)
   // Regularly occurring idiom for reading from a read handler
   // of type Record. Never stops the program if the quantity cannot be
   // read. Does not even warn. Introduced 2010-08-30.
#define cpmwh(X) rch.write(sec,#X,X)
   // Regularly occurring idiom for writing to a RecordHandler
   // write handler.
#define cpmrc(X) rc_.get(sec,#X,X,1)
```

```
   // regularly occurring idiom for reading from the attribute
   // Recordable::rc_

#define cpmc(X) if (X < obj.X) return 1;\
 if (X > obj.X) return -1
   // c for compare, helps to implement member function com
   // declared in CPM_ORDER (2009-10-02). As in the macros cpmp and
   // cpms the name 'str' for the stream argument was implied, we
   // require here the argument to be named obj'. Works fine and
   // simplifies the implementation of CPM_ORDER considerably.

#define CPM_WORD\
   CpmRoot::Word toWord()const\
   { std::ostringstream ost; prnOn(ost);\
   return CpmRoot::Word(ost.str());}

// order-related stuff now in basicinterfaces

// multiplicative structure with respect to scalars of type ScalarType.
// version where mutating member functions (actually operator *=)  are
// primary
#define CPM_PRODUCT_SCALAR_M\
   Type& operator *=(ScalarType const& s);\
   Type& operator /=(ScalarType const& s)\
   {return operator *=(CpmRoot::invT<ScalarType>(s));}\
   Type operator *(ScalarType const& s)const\
   { Type res=*this; return res*=s;}\
   friend Type operator *(ScalarType const& s, Type const& x)\
   { Type res=x; return res*=s;}\
   Type operator /(ScalarType const& s)const\
   { Type res=*this; return res/=s;}

// version where constant member functions (actually operators *) are
// primary
#define CPM_PRODUCT_SCALAR_C\
   Type operator *(ScalarType const& s)const;\
   Type operator /(ScalarType const& s)const\
   { return *this*CpmRoot::invT<ScalarType>(s);}\
   friend Type operator *(ScalarType const& s, Type const& x)\
    { return x*s;}\
   Type& operator *=(ScalarType const& s){ return *this=*this*s;}\
   Type& operator /=(ScalarType const& s){ return *this=*this/s;}

// linear structure with respect to scalars of type ScalarType.
// version where mutating member functions (actually operator *=, +=) are
// primary
#define CPM_SCALAR_M\
   Type& operator *=(ScalarType const& s);\
   Type& operator +=(ScalarType const& s);\
   Type& operator -=(ScalarType const& s){return operator +=(-s);}\
```

```
   Type& operator /=(ScalarType const& s)\
   {return operator *=(CpmRoot::invT<ScalarType>(s));}\
   Type operator *(ScalarType const& s)const\
   { Type res=*this; return res*=s;}\
   friend Type operator *(ScalarType const& s, Type const& x)\
   { Type res=x; return res*=s;}\
   Type operator +(ScalarType const& s)const\
   { Type res=*this; return res+=s;}\
   Type operator -(ScalarType const& s)const\
   { Type res=*this; return res-=s;}\
   Type operator /(ScalarType const& s)const\
   { Type res=*this; return res/=s;}

// version where constant member functions (actually operators *,+) are
// primary
#define CPM_SCALAR_C\
   Type operator *(ScalarType const& s)const;\
   Type operator +(ScalarType const& s)const;\
   Type operator -(ScalarType const& s)const{ return *this+(-s);}\
   Type operator /(ScalarType const& s)const\
   { return *this*CpmRoot::invT<ScalarType>(s);}\
   Type& operator *=(ScalarType const& s){ return *this=*this*s;}\
   Type& operator +=(ScalarType const& s){ return *this=*this+s;}\
   Type& operator -=(ScalarType const& s){ return *this=*this-s;}\
   Type& operator /=(ScalarType const& s){ return *this=*this/s;}

// product, where the mutating member function *= is primary
#define CPM_PRODUCT_M\
   Type& operator *=(Type const& x);\
   Type operator * (Type const& x)const\
   { Type res=*this; return res*=x;}

// product, where the constant member function operator * is primary
#define CPM_PRODUCT_C\
   Type operator * (Type const& x)const;\
   Type& operator *=(Type const& x){ return *this=(*this)*x;}

// inversion
// lean inversion
#define CPM_INVERSION\
   Type operator !(void)const;\
   Type& operator /=(Type const& x){ return operator*=(!x);}\
   Type operator /(Type const& x)const\
   { return *this*!x;}

// more complex version of inversion which also brings the zero into the
// game (which in a group needs not to be a defined concept)
#define CPM_DIVISION\
   Type net(CpmRoot::Z i=0)const;\
   Type inv(void)const;\
```

```
   Type operator !(void)const{ return inv();}\
   Type& operator /=(Type const& x){ return operator*=(x.inv());}\
   Type operator /(Type const& x)const\
   { return *this*x.inv();}\
   friend Type net(Type const& x, CpmRoot::Z i=0)\
   { return x.net(i);}\
   friend Type inv(Type const& x){ return x.inv();}

// inversion and netral elements; no multiplication involved.
#define CPM_INV\
   Type net(CpmRoot::Z i=0)const;\
   Type inv(void)const;\
   Type operator !(void)const{ return inv();}\
   friend Type net(Type const& x, CpmRoot::Z i=0)\
   { return x.net(i);}\
   friend Type inv(Type const& x){ return x.inv();}

// minimum infra structure for programming unbiased test for the validity
// of expected mathematical identities

#define CPM_TEST\
   Type ran(CpmRoot::Z j=0)const;\
   Type test(CpmRoot::Z)const;\
   CpmRoot::Z hash(void)const;

#define CPM_TEST_X\
   CPM_TEST\
   CpmRoot::R dis(Type const&)const;\
   CpmRoot::R abs(void)const;\
   CpmRoot::R absSqr(void)const;\
   CpmRoot::R abs2(void)const{ return absSqr();}

// CPM_TEST_ALL, comprises all infra-structure functions
// which are defined in cpmnumbers.h for the bb-types.
#define CPM_TEST_ALL\
   CPM_TEST_X\
   CPM_INV\
   Type con()const;

// CPM_TEST_MOD is a modification of CPM_TEST which turned out
// to be needed in a proper implementation of class Q where it would
// be unnatural to have abs R-valued instead of Q-valued. Also
// implementing abs in terms of absSqure would not be suitable in
// this case.
#define CPM_TEST_MOD\
   CPM_TEST\
   CpmRoot::R dis(Type const&)const;\
   Type abs(void)const;\
   Type absSqr(void)const;
```

```
// descriptors
#define CPM_DESCRIPTORS\
   virtual Word nameOf(void) const;\
   virtual Word toWord(void)const;

// arithmetic
#define CPM_DIFFERENCE\
   Type neg(void)const;\
   Type& operator -=(Type const& x){ return operator+=(x.neg());}\
   Type operator -(void)const{ return neg();}\
   Type operator - (Type const& x)const\
   { return *this+x.neg();}

#define CPM_SUM_PLAIN\
   Type& operator +=(Type const& x);\
   Type operator + (Type const& x)const;

#define CPM_DIFFERENCE_PLAIN\
   Type& operator -=(Type const& x);\
   Type operator -(void)const;\
   Type operator - (Type const& x)const;

// sum, where the mutating member function += is primary
#define CPM_SUM_M\
   Type& operator +=(Type const& x);\
   Type operator + (Type const& x)const\
   { Type res=*this; return res+=x;}

// sum, where the constant member function + is primary
#define CPM_SUM_C\
   Type operator + (Type const& x2)const;\
   Type& operator +=(Type const& x){ return *this=(*this)+x;}

// sum, where the constant member function + is primary
#define CPM_DIFF_C\
   Type operator - (Type const& x2)const;\
   Type& operator -=(Type const& x){ return *this=(*this)-x;}

// complex conjugation
#define CPM_CONJUGATION\
   Type con(void)const;\
   friend Type con(Type const& x){ return x.con();}\
   Type operator~(void)const{ return con();}

// lean complex conjugation without friend function
// (friend functions are better avoided since they cause
// problems with todays MS compilers when templates are
// heavily involved)
#define CPM_CONJ\
   Type con(void)const;\
```

```
   Type operator~(void)const{ return con();}

// Scalar Product and real valued dependents

#define CPM_DOT_PRODUCT_LEAN\
   ScalarType operator |(Type const&)const;\
   CpmRoot::R absSqr()const\
   { return CpmRoot::absT<ScalarType>(*this|*this);}\
   CpmRoot::R abs()const\
   { return cpmsqrt(absSqr());}

#define CPM_DOT_PRODUCT\
   CPM_DOT_PRODUCT_LEAN\
   CpmRoot::R dis(Type const& x)const\
   {\
      CpmRoot::R a=abs();\
      CpmRoot::R b=x.abs();\
      CpmRoot::R d=(*this - x).abs();\
      return CpmRoot::disDefFun(a,b,d);\
   }

#define CPM_NORMALIZE \
CpmRoot::R nor_()\
{\
   CpmRoot::R nOrig=abs();\
   if (nOrig>0){\
      CpmRoot::R nInv=R(1.)/nOrig;\
      operator*=(ScalarType(nInv));\
   }\
   return nOrig;\
}

// linear structure with respect to scalars of type ScalarType.
// Inversion in ScalarType is not considered; has to be added in a class
// where this is needed (e.g in C)

#define CPM_LINEAR\
   CpmRoot::Z dim()const;\
   Type& operator +=(Type const& a);\
   Type& operator -=(Type const& a);\
   Type& operator *=(ScalarType const& r);\
   Type& neg(void);\
   friend CpmRoot::Z dim(Type const& x){ return x.dim();}\
   friend Type operator +(Type const& x1, Type const& x2)\
      { Type res=x1; return res+=x2;}\
   friend Type operator -(Type const& x1, Type const& x2)\
      { Type res=x1; return res-=x2;}\
   friend Type operator -(Type const& x)\
      { Type res=x; return res.neg();}\
   friend Type operator *(Type const& x, ScalarType const& s)\
```

```
      { Type res=x; return res*=s;}\
   friend Type operator *(ScalarType const& s, Type const& x)\
      { Type res=x; return res*=s;}

// a lean and collection of arithmetic operations for linear spaces with
// scalar product. This is now the basic interface of
// VVa<>, VVVa<>, and VVVVa<>.
#define CPM_LIN\
   CPM_SUM_C\
   CPM_DIFF_C\
   Type operator-(void)const;\
   Type operator*(ScalarType const& r)const;\
   Type& operator *=(ScalarType const& r)\
      { return *this=*this*r;}\
   CPM_DOT_PRODUCT\
   CPM_CONJ
#endif
```

20 cpmm.h

```
//? cpmm.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_M_H_
#define CPM_M_H_
/*

   Description: Defines a template class M<X,Y> of mappings
      X-->Y comparable to std::map<X,Y>
*/

#include <cpms.h>

////////////////////// class M<X,Y> ////////////////////////////////////

namespace CpmArrays{

   using namespace CpmStd;
   using CpmFunctions::F;
   using CpmArrays::S;

namespace aux{ // a technical construct

template <class X, class Y>
class  Pair01{  // Pair with order according to first element only.
   // Especially all pairs differing in only the second component
   // are equal by ==. This allows an elegant access to the x-values
   // of (x,y) pairs held in M<X,Y>::rep_. Since this is a 'heavy-duty
   // technical class' it is no longer implemented as derived from
   // X2<X,Y>.
   X x_;
   Y y_;
   typedef Pair01<X,Y> Type;
```

```
public:
   Word nameOf()const
   {
      return Word("Pair01< "&
          CpmRoot::Name<X>()(X())&
          ","&
          CpmRoot::Name<Y>()(Y())&
          " >");
   }
   CPM_ORDER
   CPM_IO
   Pair01(X const& x, Y const& y):x_(x),y_(y){}
   explicit Pair01(X const& x):x_(x),y_(){}
   Pair01():x_(),y_(){}
   X& c1(void){ return x_;}
   Y& c2(void){ return y_;}
   X const& c1(void)const{ return x_;}
   Y const& c2(void)const{ return y_;}
};

template <class X, class Y>
Z Pair01<X,Y>::com(Pair01<X,Y> const& s)const
{
   if (x_<s.x_) return 1;
   else if (x_>s.x_) return -1;
   else return 0;
}

template <class X, class Y>
bool Pair01<X,Y>::prnOn(ostream& str)const
{
   cpmp(x_);
   cpmp(y_);
   return true;
}

template <class X, class Y>
bool Pair01<X,Y>::scanFrom(istream& str)
{
   cpms(x_);
   cpms(y_);
   return true;
}

} // aux

template <class X, class Y>
   class M{ // map, associative array, dictionary, hash
   // it is assumed that X<X and X>X is defined
   Y y0_;
```

```
      // default Y, allows to define
      // Y const& operator[](const X const& x)const;
      // which STL is omitting
   S< aux::Pair01<X,Y> > rep_;
   Z locate(X const& x)const{ return rep_.locate(aux::Pair01<X,Y>(x));}

public:
   typedef M<X,Y> Type;
   CPM_IO
   CPM_ORDER
   M(void):y0_(),rep_(){}
      // constructor for the void set of X,Y pairs.
      // Non-trivial instances can be obtained by applying
      // mutating methods, such as function set(...) on the
      // default intance

   M(S<X> const& sx, F<X,Y> const& f);
      // Setting the values by an algorithm.

   M(S<X> const& sx, Y const& y);
      // Setting a single value y for all elements of sx.

   void merge(V<Type> const& mapList, bool reverse=false);
      // combining *this and a list of maps into a single one.
      // This constructs a map that is formed out of all
      // (x,y)-pairs of *this and the components of mapList.
      // If for a x, there are more pairs (x,y), (x,y'), ...
      // The pair coming from mapList[i] with highest i
      // will actually be taken (if we would take more than one
      // we would not get a map). Thus we work through the
      // mapList in ascending order of their indexes and use it
      // in an overwriting mode. If the second argument differs
      // from the default value, we use reversed order so that
      // mapList[i] has highest priority for smallest i.

   Y operator()(X const& x)const;
      // in analogy to F<X,Y>; returns the 'value of *this at x'

   S<Y> operator()(S<X> const& sx)const;
      // The common extension of functions from single argument values
      // to sets of argument values

   Y& operator[](X const& x);
      // setting values in array-style (which also is the STL-style)
      // for M<X,Y> f, X x, Y y the statement
      // f[x]=y has the same effect as f.set(x,y)

   Y const& operator[](X const& x)const;
      // getting to the values as references.
      // Let Z g(const& Y), M<X,Y> f, X x, Y y . Then
```

```
      // g(f[x]) does the same as
      // Y y1; f.get(x,y1); g(y1);
      // So, operator[] provides slightly more convenient grammar than
      // get(...) in this case.
      // This is the most efficient non-mutating access function

   void set(X const& x, Y const& y){(*this)[x]=y;}
      //: set
      //  M<X,Y> f, X x, Y y ==>
      // ( f.set(x,y) ==> (y==f(x) evaluates to true)). Interesting:
      // it is possible to express this mixture of logic notation
      // and C++ much clearer in pure C++:
      // template <class X, class Y>
      // bool setTest(X x, Y y, M<X,Y> f){ f.set(x,y); return y==f(x);}
      // returns true for all arguments.

   bool get(X const& x, Y& y)const;
      //: get
      // template <class X, class Y>
      // bool getTest(X x, M<X,Y> f){ Y y; f.get(x,y); return y==f(x);}
      // returns true for all arguments.

// cardinality access: number of (x,y) pairs stored in *this
   Z car()const{ return rep_.dim();}
      //: cardinality
   Z dim()const{ return rep_.dim();} // for uniformity with other array
      //: dimension
   // types
   Z size()const{ return rep_.dim();} // for uniformity with STL code
   bool isVoid(void)const{ return rep_.isVoid();}
      //: is void
   Z b()const{ return rep_.b();}
      //. begin
   Z e()const{ return rep_.e();}
      //. end

// accessing arguments, values, and (x,y)-pairs by an index ranging from
// 1 to car()
   X x(Z i)const{ return rep_(i).c1();}
   Y y(Z i)const{ return rep_(i).c2();}
   X2<X,Y> xy(Z i)const{ return X2<X,Y>(x(i),y(i));}
      // template <class X, class Y>
      // bool xyTest(Z i, M<X,Y> f){ return y(i)==f(x(i));}
      // returns true for all arguments.
   bool defined(X const& x)const{ return locate(x)!=0;}
      // f.defined(x) iff before a statement f[x]=... or f.set(x,...)
      // happened.
   void clear_(X const& x){ Z i=locate(x); if (i>0) rep_.eliminate(i);}
      // after f.clear_(x), we have f.defined(x)==false
```

```
   S<X> dom()const;
      //: domain
      // domain of the function X-->Y associated with object *this
      // returns {x \in X | defined(x)==true}

   S<Y> ran()const;
      //: range
      // range of the function X-->Y associated with object *this
      // returns {y \in Y | y==operator[](x) for some x \in X}

   M<X,Y> select(F<X,bool> const& sel)const;
      // returns the M<X,Y> objects consisting of all those
      // (x,y)-pairs of *this for which sel(x)==true

   template <class T>
   M<X,T> operator()(F<Y,T> const& g)const;

   template <class T>
   M<X,T> operator&(M<Y,T> const& m)const;

   template< class T>
   M<T,Y> circ(M<T,X> const& m)const{ return m&(*this);}
      //: circ (LaTeX-command)
      // concatenation of mappings

   Word toWord()const;
      //: to word

   Word nameOf()const;
      //: name of
};

////////////////////////// Implementation ////////////////////////////////

template <class X, class Y>
template<class T>
M<X,T> M<X,Y>::operator()(F<Y,T> const& g)const
{
   M<X,T> res();
   for (Z i=b();i<=e();++i){
      X2<X,Y> xyi=xy(i);
      res.set(xyi.c1(),g(xyi.c2()));
   }
   return res;
}

template <class X, class Y>
template <class T>
M<X,T> M<X,Y>::operator&(M<Y,T> const& m)const
{
```

```
   M<X,T> res;
   for (Z i=b();i<=e();++i){
      X2<X,Y> xyi=xy(i);
      bool bi=m.defined(xyi.c2());
      if (bi) res.set(xyi.c1(),m[xyi.c2()]);
   }
   return res;
}

template <class X, class Y>
Word M<X,Y>::nameOf()const
{
   Word wi="M<";
   Word wx=CpmRoot::Name<X>()(X());
   Word wy=CpmRoot::Name<Y>()(Y());
   return wi&wx&","&wy&">";
}

template <class X, class Y>
Word M<X,Y>::toWord()const
{
   Word res="{ ";
   for (Z i=b();i<=e();++i){
      X2<X,Y> xyi=xy(i);
      Word wxi=CpmRoot::toWord<X>(xyi.c1());
      Word wyi=CpmRoot::toWord<Y>(xyi.c2());
      res&=" ( "&wxi&" , "&wyi&" ) ";
      if (i!=e()) res&=" , ";
   }
   res&=" }";
   return res;
}

template <class X, class Y>
M<X,Y>::M(S<X> const& sx, F<X,Y> const& f):y0_(),rep_()
{
   for (Z i=sx.b();i<=sx.e();++i){
      X xi=sx[i];
      set(xi,f(xi));
   }
}

template <class X, class Y>
M<X,Y>::M(S<X> const& sx, Y const& y):y0_(),rep_()
{
   for (Z i=sx.b();i<=sx.e();++i){
      X xi=sx[i];
      set(xi,y);
   }
}
```

```
template <class X, class Y>
Y M<X,Y>::operator()(X const& x)const
{
   Z i=locate(x);
   if (i==0) return Y();
   else return rep_[i].c2();
}

template <class X, class Y>
Y& M<X,Y>::operator[](X const& x)
{
   aux::PairO1<X,Y> fx(x);
   Z i=rep_.locate(fx);
   if (i==0){
      rep_.add(fx);
   }
   i=rep_.locate(fx);
   if (i<=0) cpmerror("Y& M<X,Y>::operator[](X const& x): i<=0");
   return rep_.ref(i).c2();
      // since the ordering of rep_ only relies on rep_.ref(i).c1()
      // there is no danger in exposing rep_.ref(i).c2() to
      // changes from outside
}

template <class X, class Y>
Y const& M<X,Y>::operator[](X const& x)const
{
   aux::PairO1<X,Y> fx(x);
   Z i=rep_.locate(fx);
   if (i<=0){
      return y0_;
   }
   else{
      return rep_[i].c2();
   }
}

template <class X, class Y>
bool M<X,Y>::get(X const& x, Y& y)const
{
   Z i=locate(x);
   if (i==0){
      return false;
   }
   else{
      y=rep_[i].c2();
      return true;
   }
}
```

```
template <class X, class Y>
S<Y> M<X,Y>::operator()(S<X> const& sx)const
{
   Z n=sx.car();
   V<Y> val(n);
   for (Z i=1;i<=n;i++){
      val[i]=y((sx[i]));
   }
   return S<Y>(val);
}

template <class X, class Y>
void M<X,Y>::merge(V<Type> const& mapList, bool reversed)
{
   Z ni,i,j,d=mapList.dim();
   if (!reversed){
      for (i=1;i<=d;i++){
         ni=mapList[i].car();
         for (j=1;j<=ni;j++){
            X x=mapList[i].x(j);
            Y y=mapList[i].y(j);
            set(x,y);
         }
      }
   }
   else{
      for (i=d;i>=1;i--){
         ni=mapList[i].car();
         for (j=1;j<=ni;j++){
            X x=mapList[i].x(j);
            Y y=mapList[i].y(j);
            set(x,y);
         }
      }
   }
}

template <class X, class Y>
S<X> M<X,Y>::dom()const
{
   S<X> res;
   for (Z i=1;i<=dim();++i) res.add(x(i));
   return res;
}

template <class X, class Y>
S<Y> M<X,Y>::ran()const
{
   S<Y> res;
```

```
   for (Z i=1;i<=dim();++i) res.add(y(i));
   return res;
}

template <class X, class Y>
M<X,Y>  M<X,Y>::select(F<X,bool> const& sel)const
{
   M<X,Y> res;
   for (Z i=1;i<=dim();++i){
      X2<X,Y> xyi=xy(i);
      X xi=xyi.c1();
      if (sel(xi)) res[xi]=xyi.c2();
   }
   return res;
}

template <class X, class Y>
bool M<X,Y>::prnOn(ostream& str)const
{
   Z mL=3;
   static Word loc("Mio<>::prnOn(ostream&)");
   CPM_MA
   Z n=car();
   cpmwat;
   cpmp(n);
   for (Z i=1;i<=n;i++){
      if (CpmRoot::wrtTit){
         Word wi("// i=");
         wi&=cpm(i);
         bool bi=wi.prnOn(str);
         cpmassert(bi==true,loc);
      }
      X2<X,Y> val(x(i),y(i));
      if (!val.prnOn(str)){
         CPM_MZ
         return false;
      }
   }
   cpmwet;
   CPM_MZ
   return true;
}

template <class X, class Y>
bool M<X,Y>::scanFrom(istream& str)
{
   Z mL=3;
   Z mL2=5;
   static Word loc("M<>::scanFrom(istream&)");
   CPM_MA
```

```
   Z n;
   if (!CpmRoot::scanT<Z>(n,str)){
      cpmwarning(loc&": can't read dimension");
      CPM_MZ
      return false;
   }
   else if (n<0){
      cpmwarning(loc&": negative dimension: n="&cpm(n));
      CPM_MZ
      return false;
   }
   else if (n>dimMax)
      cpmwarning(loc&": dimension read as "&cpm(n)&
      ": probably too large");
   else if (n==0){
      cpmwarning(loc&" dimension read as 0");
      if (cpmverbose>mL2){ // copying the stream beyond the dubious point
         // can be useful
         char c;
         while (str.get(c)) cpmdata.put(c);
      }
   }
   else{
      cpmmessage(mL,loc&": dimension read as "&cpm(n));
   }
   M<X,Y> res;
   X2<X,Y> val;
   for (Z i=1;i<=n;i++){
      if (!val.scanFrom(str)){
         cpmwarning(loc&": reading failed for i="&cpm(i)&" of "&cpm(n));
         CPM_MZ
         return false;
      }
      res.set(val.c1(),val.c2());
   }
   *this=res;
   CPM_MZ
   return true;
}

template <class X, class Y >
Z M<X,Y>::com(M<X,Y> const& s)const
{
   Z d1=dim(), d2=s.dim();
   if (d1<d2) return 1;
   else if (d1>d2) return -1;
   else{ // then d1==d2
      for (Z i=1;i<=d1;i++){
         if (xy(i)<s.xy(i))   return 1;
         if (xy(i)>s.xy(i))   return -1;
```

```
      }
      return 0;
   }
}

} // CpmArrays

#endif
```

21 cpmmacros.h

```
//? cpmmacros.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_MACROS_H_
#define CPM_MACROS_H_
/*

   Description: macros for uniform implementation of
      functions. Too often I wrote the starting cpmmessage
      and the closing one in function blocks. The abbreviation by macros
      is not impressive in this case. For the R-macro the savings are
      more convincing.

*/

////////////////////////// CPM_MAZ ///////////////////////////////////////

// M stands for messaging
//              -
// A for beginning, Z for end (from the
// role of A and Z in the alphabet)

// needs
// Z mL=...;
// Word loc(".....");
// see the example following CPM_RAZ

#define CPM_MA\
   cpmmessage(mL,loc&" started");

#define CPM_MZ\
   cpmmessage(mL,loc&" done");
```

```
////////////////////// CPM_RAZ /////////////////////////////////////

// R stands for report
//              -
// reports the computation time for the code between CPM_RA and
// CPM_RZ. If loadMes (load measure) is assigned a positive value
// also the computation time divided by that quantity is returned
// under the name of 'normalized computation time'

// needs
// Word loc(".....");

#define CPM_RA\
   R tcStart=cpmtime();\
   R loadMes=0;

#define CPM_RZ\
   R tcFinal=cpmtime();\
   R tcTotal=tcFinal-tcStart;\
   std::ostringstream ost;\
   ost<<std::endl<<"Performance data for "<<loc.toStr()<<std::endl;\
   ost<<"computing time ="<<tcTotal;\
   if (loadMes>0){\
      R tcNorm=tcTotal/loadMes;\
      ost<<std::endl<<"normalized computing time="<<tcNorm;\
   }\
   cpmmessage(1,Word(ost.str()),-1);
      // not writing to the status bar due to '-1'

////////////////////// usage of CPM_MAZ and CPM_RAZ //////////////////////
/*
   a typical usage is

  void aFunction()
  {
      Z mL=2;
      Word loc("aFunction()");
      CPM_MA
      CPM_RA
      ...
      loadMes=...;
      ...
      CPM_RZ
      CPM_MZ
   }

   for a function that notifies start, end, computation time,
   and an efficiency measure on cpmcerr.txt. Notice that A and Z
   resemble opening and closing brackets.
*/
```

```
#endif
```

22 cpmmpi.h

```
//? cpmmpi.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_MPI_H_
#define CPM_MPI_H_

/*
   Description: Functions to send and receive objects of C++
   type string via MPI functions. The length of the strings
   is handled automatically. This header file will be included in
   cpminterfaces.h and, therefore, should not include files like
   cpmsystem.h, cpmtypes.h, cpmword.h
*/

#include <cpmdefinitions.h>
   // only to make known CPM_USE_MPI
   // no additional C+- stuff needed here

#include <string>

#if defined(CPM_USE_MPI)
   #include <mpi.h>
      // MPIPro
      // including within the namespace caused trouble!
#endif

namespace CpmMPI{

using std::string;

extern int mpiverbose;

#if defined(CPM_USE_MPI)
```

```
////////////////////////////// class Com ////////////////////////////////

class Com{
// class version of MPI's communicator concept for parallel computing
   // for which rank varies from 1 to size in order to cooperate with
   // CpmArrays. Note that, due to their template character, the
   // main communication functions can be used to send arround objects
   // of any type which implements the basic C+- marshalling scheme
   // provided by macro CPM_IO.

protected:
   MPI_Comm mc;
   int size;
   int rank; // starts with 1
public:
   static int tagFromToStatic(int f, int t,int size_)
   // returned simply f*t till 2004-07-07. The footnote to p. 254 of GLS
   // (Gropp,Lusk,Skjellum: using MPI) indicates that using the same tag
   // several times in succession might be not so desastrous as one could
   // expect it to be. Nevertheless, one should ensure that the function
   // is injective (appart from tagFromToStatic(f, t, size_)==
   // tagFromToStatic(t, f, size_) )
   {
      int i,j;
      if (f<=t){ i=f-1; j=t;}
      else{ i=t-1; j=f;}
      return size_*i+j;
   }
   int tagFromTo(int f, int t)const{ return tagFromToStatic(f, t, size);}
   Com(MPI_Comm comm=MPI_COMM_WORLD);
   int getSize()const{ return size;}
      // number of processes between which one can communicate via
      // sendStr and recStr
   int getRank()const{ return rank;}
   bool sendStr(const string& s, int dest, int tag)const;
      // sending C++-strings without having to input their length
      // return value true means success.
      // I use string as a universal data type singe the C++ class
      // stringstream offers efficient read and write.
      // 1<=dest<=size
   bool bcastStr(string& s)const;
      // broadcasts a string from the process with rank 1 to
      // all other processes
   bool recStr(string& s, int source, int tag)const;
      // receiving strings
      // 1<=source<=size
   int cyc(int i)const;
      // the return value k is equal i modulo size
      // also 1<=k<=size. Useful tool although one could argue that this
```

```
      // function belongs to the topic of 'communicator topologies'
      // and not to the communicator itself

// communication patterns:

// The aim of the following functions is to ensure a pairing
// of send and receive activities that avoids deadlock. Although
// it is not difficult to gain an intuitive insight saying that
// the combinations as implemented in these functions are safe against
// deadlock, I have no formal proof for this so far.
// Supporting points are
// 1. empirical: no exception found so far
// 2. philosophical: no simpler expression conceivable except
//    of obviously wrong ones.

// template member functions are very helpful here.
   template <class T, class V>
      // T has to define send() and rec(), V has to have
      // the basic properties of V<T> (see cpmv.h)
   void sendAndRec(const T& t, V& v)const
/*
   Typical application in a SPMD (single program, multiple data )
   situation: (using C+- classes)
   Com com;
   Z size=com.getSize();
   T t(...); // the contribution of com.getRank()
   V<T> v(size); // container for the contributions of others
   com.sendAndRec(t,v); // communication
   R sum=0; // starting to use the communicated stuff
   for (Z i=1;i<=size;i++) sum+=g(v[i]); // R g(const T&)
       // A cummulative effect of all processes is thus available for
       // each process.
*/
   {
      for (int i=1;i<=size;i++){
         if (i==rank){ // OK for size==1, rank==1
            v[i]=t;
         }
         else if (rank<i){
            t.send(i,tagFromTo(rank,i));
            v[i].rec(i,tagFromTo(i,rank));
         }
         else{
            v[i].rec(i,tagFromTo(i,rank));
            t.send(i,tagFromTo(rank,i));
         }
      }
   }

   template <class V>
```

```
      // V has to have the basic properties of V<T>, where
      // T defines send() and rec().
   void exchange(V const& s, V& v)const
/*
   Typical application in a SPMD (single program, multiple data )
   situation: (using C+- classes)
   Com com;
   int size=com.getSize();
   int rank=com.getRank();
// the contribution of com.getRank()
   V<T> s(size);
   for (int i=1;i<=size;i++){
      for (i==rank) continue;
         // there is nothing to exchange from rank to rank
      s[i]=....; // this is what rank has to say to i
   }
   V<T> v(size); // container for the messages from others
   com.exchange(s,v); // communication
   R sum=0; // starting to use the communicated stuff
   for (int i=1;i<=size;i++) sum+=g(v[i]); // R g(const T&)
       // Process rank has available the messages from its
       // partners all at once
*/
   {
      for (int i=1;i<=size;i++){
         if (i==rank){
            v[i]=s[i]; // was continue till 2003-06-10
         }
         else if (rank<i){
            s[i].send(i,tagFromTo(rank,i));
            v[i].rec(i,tagFromTo(i,rank));
         }
         else{
            v[i].rec(i,tagFromTo(i,rank));
            s[i].send(i,tagFromTo(rank,i));
         }
      }
   }

//   export // not working with MS cl
   template <class T>
      // see sendAndRec() for the assumed properties of T
   void bcast(T& t)const
      // 'broadcast'
      // Workhorse method for distributing data from the master to
      // the public. See CpmRootX::RecordHandler::ini_ for the typical
      // usage.
      // Call to bcast has to be preceeded by initialization of t in rank
      // 1. In tis respect rank 1 plays a special role; the call to
      // bcast shoult not be spoilt by conditions on rank!
```

```
   // After call for each rank t is assured to have the value which
   // rank 1 had provided. Execution of this function involves each
   // rank with at least one send or receive activity. Since rank 1
   // sends to all processes, and send is not considered done before
   // data are received, rank 1 can only continue after all receives
   // are done in all other processes. Any other process also can
   // only continue after it received. Who received first may hurry
   // away not waiting till others are also ready with receiving. So
   // this is not a strict synchronization mechanism.
   // However, each process that hurries away must have completed its
   // receive and this can only be achieved if all messages sent
   // earlier have been received to. So the function should force
   // completion of all communication tasks that  were pending at the
   // moment of calling bcast. Similar considerations apply to the
   // other collective communication functions.

   // A definition of this template providing diagnostics would be
   // desirable. In order to have cpmmessage available one could in
   // principle include  cpmsystems.h into the present file and
   // implement messages much like those of sendStr and recStr. In our
   // special situation this is not working since cpmmpi.h has to be
   // included in cpminterfaces.h where cpmsystem.h should not yet be
   // known. So one had to move the implementation to cpmmpi.cpp which,
   // however, asks for the additional qualification 'export'
   // (see Vandevoorde, Josuttis: C++ Templates, p. 68) which MS cl
   // does not understand.
{
   if (size==1) return;
   if (rank==1){
      for (int i=2; i<=size;i++) t.send(i,tagFromTo(1,i));
   }
   else  t.rec(1,tagFromTo(1,rank));
}

template <class T, class V>
void gather(const T& t, V& v)const
// The out-commented messages made me aware of the actual timing of
// processes in my emulating MPICH runs: Writing of moviefiles for some
// processes may be in progress long before function step() is done for
// all tasks.
{
   if (size==1){ v[1]=t; return;}
   if (rank!=1){
      t.send(1,tagFromTo(rank,1));
   }
   else{
      v[1]=t;
      for (int i=2;i<=size;i++) v[i].rec(i,tagFromTo(i,1));
   }
}
```

```
   template <class T, class V>
      // see sendAndRec() for the assumed properties of T and V
   void gatherXL(const T& t, V& v, int n)const
      // clear from code
   {
      if (size==1){ v[1]=t; return;}
      if (n<2) {
         gather(t,v);
         return;
      }
      if (rank!=1){
         t.sendXL(1,tagFromTo(rank,1),n);
      }
      else{
         v[1]=t;
         for (int i=2;i<=size;i++) v[i].rec(i,tagFromTo(i,1));
      }
   }

   template <class T, class V>
      // see sendAndRec() for the assumed properties of T and V
   void scatter(T& t, const V& v)const
      // clear from code, not yet used or tested
   {
      if (rank==1){ // directly OK for size==1
         t=v[1];
         for (int i=2;i<=size;i++) v[i].send(i,tagFromTo(1,i));
      }
      else{
         t.rec(1,tagFromTo(1,rank));
      }
   }

   template <class T, class V>
   void scatterXL(T& t, const V& v, int n)const
      // clear from code, not yet used or tested
   {
      if (rank==1){
         t=v[1];
         for (int i=2;i<=size;i++) v[i].sendXL(i,tagFromTo(1,i),n);
      }
      else{
         t.rec(1,tagFromTo(1,rank));
      }
   }
};

#else // not defined(CPM_USE_MPI)
```

```
class Com{ //trivial implementation of Com

public:
   static int tagFromToStatic(int f, int t,int size_)
   {f;t;size_;return 0;}
   static int tagFromTo(int f, int t){ f;t;return 0;}
   Com(){}
   int getSize()const{ return 1;}
   int getRank()const{ return 1;}
   bool sendStr(const string& s, int dest, int tag)const
   {s;dest;tag; return true;}
   bool bcastStr(string& s)const{s; return true;}
   bool recStr(string& s, int source, int tag)const
      {s; source; tag; return true;}
   int cyc(int i)const{ i;return 1;}

   template <class T, class V>
      void sendAndRec(const T& t, V& v)const{t;v;}

   template <class V>
      void exchange(const V& s, V& v)const
      { v[1]=s[1];}

   template <class T>
      void bcast(T& t)const{t;}

   template <class T, class V>
      void gather(const T& t, V& v)const{t;v;}

   template <class T, class V>
      void gatherXL(const T& t, V& v, int n)const{t;v;n;}

   template <class T, class V>
      void scatter(T& t, const V& v)const{t;v;}

   template <class T, class V>
      void scatterXL(T& t, const V& v, int n)const{t;v;n;}
};

#endif // #if defined(CPM_USE_MPI)

extern CpmMPI::Com Cpm_com;
void initialize(int* a, char*** b);
void finalize();

} // namespace

#endif // #if defined(CPM_MPI_H)
```

23 cpmnumbers.h

```
//? cpmnumbers.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_NUMBERS_H_
#define CPM_NUMBERS_H_
/*
   Description: The C+- class system needs some basic types from C++.
      These are in the first place: (long) int, unsigned (long) int,
      unsigned char, bool, string, and (long) double.
      With the availability of convenient and efficient C++ types for
      floating point numbers of 'multiple precision' a new degree of
      freedom can be added to the C+- class system. We therefore add
      the multiple precision class mpfr::mpreal (by Pavel Holoborodko)
      which is based on the multiple precision libraries gmp and mpfr.
      This class can be used if the macro CPM_MP is defined in the header
      file cpmdefinitions.h.

      Introduces a common infrastructure for some basic types.
      There are two topics of interest here:
      1. It is a common situation that a class is to be defined in a way
      that some (or all) its data members have types which belong to these
      basic types. Then the methods should be in place which allow to
      carry over some basic infra structure from the members to the
      class to be defined. Most frequently used in present C+- code is
      the implementation of stream I/O ('Marshalling') according to the
      following example:

      class X{
         B1 x1; // B1,B2,...,Bn basic types
         B2 x2;
         ...
         Bn xn;
         typedef X Type;
```

```
public:
   CPM_IO
      // declaration macro for Input/Output
   Word nameOf()const{ return "X";}
      //: name of
      ...
};

bool X::prnOn(str)const
{
   cpmwat; // uses the nameOf - function to write an
      // automatic 'title'
   cpmp(x1); // 'p' for 'print'
   cpmp(x2);
   ...
   cpmp(xn);
   return true;
}

bool X::scanFrom(str)
{
   cpms(x1); // 's' for 'scan'
   cpms(x2);
   ...
   cpms(xn);
   return true;
}

The declaration macro CPM_IO and the implementation macros
cpmwat, cpmp, cpms are defined in cpminterfaces.h.
The main achievement is the recursive nature of the scheme:
If X is the type of a member of a class Y the same mechanism
works:

class Y{
   X x1;
   ...
   typedef Y Type;
public:
   CPM_IO
   Word nameOf()const{ return "Y";}
   ...
};

bool Y::prnOn(str)const
{
   cpmwat; // uses the nameOf - function to write an
      // automatic 'title'
   cpmp(x1); // 'p' for 'print'. According to the
      // definition of X this makes sense as a part of
```

```
            // the implementation of Y's stream interface.
            ...
         return true;
      }

      bool Y::scanFrom(str)
      {
         cpms(x1); // 's' for 'scan'
           ...
         return true;
      }

      The details of the scheme will be given later in this file.

      2. Functions which enable unbiased testing of mathematical
      properties especially of class templates.

         ... not yet finished

      Z, N, R, L, bool, string (summerized as bb-types:
      basic built-in types). Thus no longer restricted to numbers.
      Provides most of the content of CpmRoot, cpmword adds the
      rest.

      We try to avoid the unsigned number types and thus don't introduce
      Nh, Nd as in earlier versions of this class library.
      (See. Bjarne Stroustroup: The C++ Programming Language,
      Third Edition, Addison-Wesley p. 70-71 last and first few lines.
      This book will referred to as BS3, the second edition as BS2.
      At present we have a fourth edition BS4.
      Certainly it is highly desirably to have the basic number
      types defined as classes. These seems however be possible only by
      reducing speed by factor 8, or so, in numerics intensive
      programs. This is not considered acceptable here.
      More recent experiments (in 2010) gave a speed loss of only 20% upon
      speed-optimized compilation. So the previous judgement seems no
      longer to have a very firm basis.

      We do not yet make use of the class CpmRoot::Word in this file.
*/

// making available the basic C++ standard library facilities
#include <string>
#include <iostream>
#include <fstream>  // for ifstream, ofstream
#include <sstream>
#include <cmath>
   // all math functions in namespace std ???
   // They can be also used as if they were in the global namespace.
   // This is important that we don't need a directive
```

```
   // 'using namespace std' in this file (as I had prior to 2016-08-29
   // and got a clash with a probably newly introduced function
   // std::ignore.
#include <limits>
//#include <regex>
#include <functional>

#include <cpmbasictypes.h>
   // Defines integer types Z, N , L.
   // Includes cpmdefinitions.h from which defines (or does not
   // define) the macro CPM_MP which controls the usage of
   // 'multiple precision' floating-point numbers.

#if defined(CPM_MP)
   #if defined(CPM_USE_MPREAL)
      #include <mpreal.h> // Declares and defines class mpfr::mpreal.
         // No mpreal.cpp needed any more.
         // The definition relies on
         // gmp.lib, mpfr.lib. See
         // http://www.holoborodko.com/pavel/
   #else
      //#include <boost/multiprecision/cpp_dec_float.hpp>
      //#include <boost/multiprecision/mpfr.hpp>
      #include <boost/multiprecision/gmp.hpp>
      #include <boost/math/special_functions/round.hpp>
      #include <boost/math/constants/constants.hpp>
         // relies on the boost library
   #endif
#endif

namespace CpmStd{
// This is the part of the standard library which we use frequently.
// With C++11 we replace F1,F2,... with the replace of std::bind and
// namespace std::placeholders.
   using std::string;
   using std::ostream;
   using std::istream;
   using std::iostream;
   using std::ofstream;
   using std::ifstream;
   using std::fstream;
   using std::ostringstream;
   using std::istringstream;
   using std::stringstream;
   using std::endl;
   using std::cout;
   using std::cin;
   using std::bind;
   using namespace std::placeholders;
 } // CpmStd
```

```
namespace CpmRoot{
   using namespace CpmStd;

// Defining type R, the real type with which present C+- code uses
// in all places where floating point values are needed. No longer
// I use floating point types of multiple and fixed precision (such
// as R together with Rh or R_) in parallel.
#if defined(CPM_MP) // MP stands here for 'multiple precision'.
   // a typical line to appear in cpmdefinitions.h is
   // #define CPM_MP 32
   // where the number gives the number of decimal digits.
   #if defined(CPM_USE_MPREAL) // use class from Pavel Holoborodko
      // A very convenient feature of this selection is the following:
      // In projects which use cpmapplication.cpp this number CPM_MP
      // can be overridden in cpmconfig.ini without causing a need for
      // recompilation. The expected entry there is of the form:
      //    numerical precision     // section
      //    Z val=64                // value
      // mpfr::mpreal is a wrapper class by Pavel Holoborodko to the
      // MPFR function system. See
      // http://www.holoborodko.com/pavel/
      // This class has an interface to other
      // C++ types which is very similar to that of type double.
      // So an algorithm coded in conventional style will still work
      // when type double is replaced by class mpfr::mpreal. Since the
      // information content (number of digits) can be set 'arbitrarily'
      // large, there is no longer a fundamental difference between
      // class mpfr::mpreal and the mathematical concept of the real
      // number field. See the section 'motivation and rational'
      // in the documentation to class AppMath::R in
      // http://www.ulrichmutze.de/rubystuff/doc_rnum/index.html
      // for more motivation concerning this point of view.
      // mpfr::sin(r) is the full name of the sine
      // function for argument r \in mpfr::mpreal. This is then a clear
      // distinction from e.g. CpmRoot::sin(C const&).
   typedef mpfr::mpreal R;
      using namespace mpfr;
    #else
      // Then we use the tools from boost/multiprecision. These interface
      // well with boost/math but show strange behavior with respect to
      // C++ conversion operators and create a lot of warnings which at
      // the end of building are reported as errors by the Code::blocks
      // IDE. Nevertheless the result of building is valid and works.
      // Standard functions such as sqrt are also in this namespace.
    using namespace boost::multiprecision;
    using namespace boost::math;
      // typedef number<cpp_dec_float<CPM_MP>, et_off > R;
         // works, but is by a factor ~ 1.5 slower than the gmp version
         // which is chossen in the present code.
```

```
      //typedef number<mpfr_float<CPM_MP>, et_off > R;
         // mpfr does not work here with my present mpfr imlementation

    typedef number<gmp_float<CPM_MP>, et_off > R;
 //typedef number<gmp_float<CPM_MP>, et_on > R;
      // Notice the et_off-switch which disables expression-templates.
      // If these would left active compilation of nearly all of my C+-
      // code basis would fail. Since the Eigen library is said to build
      // heavly on expression templates disabling expression templates
      // probably compromises the efficiency of Eigen. Should be clarified!
      // Notice here numerical precision is set at compile time and can
      // not be changed by reading input from cpmconfig.ini.
   #endif
#else
   // Then we don't need the multiple precision libraries and get
   // more speed. This was the only mode of C+- till March 2009. We have
   // the choice between double (64 bit) and long double (80 bit on most
   // systems) as a definition of type R.
   #if defined(CPM_LONG) // then: typedef long int Z
      typedef long double R;
   #else
      typedef double R;
   #endif
  // using namespace std; // experiment
#endif

// Remark on using CPM_LONG:
// C+- often overloads functions on the basis of argument type R or Z.
// As an example consider the following constructors in cpmviewport.h:
// CpmGraphics::rgb(Z r_, Z g_, Z b_, bool norm=true);
// CpmGraphics::rgb(R r_, R g_, R b_, bool norm=true);
// For R=double one can distinguish the two by calling e.g.
// rgb c1(255.,128.,10.); for the R-version, and
// rgb c2(0,0,255); for the Z-version.
// For R=long double the code will become ambiguous since interpreting
// 255 as long int is not considered better than interpreting it as long
// double. Better one avoids numerical literals as function arguments
// and writes
// R r=255, g=128, b=10; Z z0=0,z1=255;
// rgb c1(r,g,b); rgb c2(z0,z0,z1);
// This gives all quantities their matching type and is as readable as
// rgb c1(R(255),R(128),R(10)); rgb c2(Z(0),Z(0),Z(255));

// Two implementation tools:

inline R disDefFun(R a, R b, R d)
   //: distance defining function
   // Function for implementing function dis.
   // Needs also be known to cpminterfaces.h.
   // Continuous function, but not reasonable from a geometrical
```

```
   // point of view, since the distance dis(a,b):=disDefFun(|a|,|b|,|a-b|)
   // between points based naturally on this function is not translation
   // invariant.
{
   R s=a+b;
   if (s==0.) return R(0.);
   R d1=d/s;
   return (d < d1 ? d  : d1);
}

template <class T>
T posVal(T t){ return t<T() ? -t : t;}
   //: positive value
   // added 2005-09-10. Similar to abs but not necessarily
   // R-valued. Not intended to be used for aggregated data types

// Defining the basic R-related functions as class-less
// functions in namespace CpmRoot.

void prec(Z p);
   // Setting precision; a typical numerical error is ~ 10^(-p).
   // Thus p is a number of decimal digits.
   // Always defined, but active only if
   //    defined(CPM_MP) && defined(CPM_USE_MPREAL).
   // Notice that for !defined(CPM_USE_MPREAL) the precision is set
   // at compile time
Z getPrec(void);
   // Returns the active value of precision.
   // If !defined(CPM_MP) the corresponding value for type
   // double or long double (i.e. the actual type of R) is returned.
R inv(R const& r);
   // returns r==0. ? 0 : 1./r with warnings in the case r==0.
R randomR(Z j=0);
   // Returns uniformely distributed random values in [0,1) if
   // no argument (or argument 0) is used. For j != 0 the return
   // value is a 'chaotic' function of j, so that for any sequence
   // of argument values we get a random sequence. Since the value
   // is determined by j, one gets reproducible results which sometimes
   // is useful. In March 2013 I tested {0,1}-valued sequences of length
   // 1000000 with the NIST Random Number Test Suite and found all tests
   // passed convincingly. I used the NIST suite as translated to
   // Mathematica by Ilja Gerhardt in 2010 and I translated randomR also
   // to Mathematica.
R test(R const& r, Z i);
   // Implementation tool for class Test<>, see this file.
R ran(R const& r, Z i=0);
   // Implementation tool for class Ran<>, see this file.
   // The return value x satisfies -|r| <= x < |r|
Z hash(R const& r);
   // Implementation tool for class Hash<>, see this file.
```

```
R dis(R const& r1, R const& r2);
   // Implementation tool for class Dis<>, see this file.
Z getPrec(R const& r);
   // Returns the number of decimal digits used to hold the value of r.
bool isVal(R const& r);
   // Returns 'false' if r is not a regular number (infinite or NaN).
Z toZ(R const& r, bool toZero=false);
   // For positive r we return floor(r) as a Z instead of a R.
   // For negative r, the second argument determines whether the next
   // integer in minus-direction (toZero=false) or in plus-direction
   // (toZero=true) is returned. So, for toZero==false, the result is
   // floor(r) also for negative r.
// functions with inline implementation
inline Z sgn(R const& r)
{
   if (r>0.) return 1;
   if (r<0.) return -1;
   return 0;
}

inline R hypot(R const& x, R const& y)
// hypotenuse, a numerically careful version of sqrt(x*x + y*y).
{
   R tiny=1.e-12;
   R xa=posVal<R>(x),ya=posVal<R>(y),xya=xa+ya;
   if (xya==R(0.)) return R(0.);
   xya=(xya<tiny ? tiny : xya);
   R xyaInv=CpmRoot::inv(xya);
   R xr=xa*xyaInv, yr=ya*xyaInv;
   return R(xya*sqrt(xr*xr+yr*yr));
}

inline R con(R const& r){ return r;}
   // Implementation tool for class Conj<>, see this file.

inline R net(R const& r, Z i){ if (i==1) return R(1.); else return R(0.);}
   // Implementation tool for class Neutrals<>, see this file.

inline Z rnd(R const& r)
   // Returns the nearest integer value (rounding to integer).
{
#if defined(CPM_MP)
   #if defined(CPM_USE_MPREAL)
      return (floor(r+0.5)).toLong();
   #else
      return lround(r);
   #endif
#else
   return (Z)floor(r+0.5);
#endif
```

```
}

inline bool isZero(R const& r){ return r==R(0.);}
   // Returns 'true' if r==0, and 'false' else.

inline double toDouble(R const& r){
   // to double, helps to program uniformly formatted screen output
#if defined(CPM_MP)
   #if defined(CPM_USE_MPREAL)
      return r.toDouble();
   #else
      return static_cast<double>(r);
   #endif
#else
   //return (double)r;
   return static_cast<double>(r);
#endif
}

inline R absSqr(R const& r){ return r*r;}
   //: absolute (value) squared
   // Returns the square of |r| (which happens to be the square of r)

inline R arg(R const& x, R const& y)
   //: argument
   // Returns the polar angle (in (-Pi,Pi]) of point (x,y).
   { return atan2(y,x);}

// Split implementation for circle related matters.
#if !defined(CPM_MP)
   const R Pi=asin(0.5)*6.0;
      // Number pi=3.14159..., pi = 30*deg * 6, sin(30*deg) = 0.5.
      // Faster convergence expected then for Pi = atan(1.)*4.
      // This is certainly not an important issue in this place.
   const R Pi2=Pi*2;
      // 2 * pi
   const R AngDeg= Pi/180.;
      //: angular degree, pi/180
#endif
} // CpmRoot closed, will be re-opened soon

// Convenient abbreviations for constants and service functions.
// See also corresponding abbreviations
// cpmnam, cpmtow in cpmword.h and
// cpmswp, cpmord, cpminf, cpmsup, cpmtin, cpmhug in cpmtypes.h.

#if defined CPM_MP
   #if defined(CPM_USE_MPREAL)
   // Since precision may change, the pi-related macros
   // define function calls and not the reading of values from storage.
```

```
      #define cpmpi     mpfr::const_pi()
      #define cpmpiInv  CpmRoot::R(1.)/mpfr::const_pi()
      #define cpm2pi    mpfr::const_pi()*2
      #define cpmdeg    mpfr::const_pi()/180.
      #define cpmrho    mpfr::hypot
   #else
      #define cpmpi     R(boost::math::constants::pi<CpmRoot::R>())
      #define cpmpiInv  R(R(1.)/boost::math::constants::pi<CpmRoot::R>())
      #define cpm2pi    R(boost::math::constants::pi<CpmRoot::R>()*R(2.))
      #define cpmdeg    R(boost::math::constants::pi<CpmRoot::R>()/R(180.))
      #define cpmrho    CpmRoot::hypot
   #endif
#else
   #define cpmpi     CpmRoot::Pi
   #define cpmpiInv  1./CpmRoot::Pi
   #define cpm2pi    CpmRoot::Pi2
   #define cpmdeg    CpmRoot::AngDeg
   #define cpmrho    CpmRoot::hypot
#endif

// easy access to C+- utility functions
// Notice that we are here not in CpmRoot.
#define cpmrnd    CpmRoot::rnd
#define cpmtoz    CpmRoot::toZ
#define cpmtod    CpmRoot::toDouble
#define cpmpos    CpmRoot::posVal
#define cpmprn    CpmRoot::prnOn
#define cpmscn    CpmRoot::scanFrom
#define cpmsgn    CpmRoot::sgn
#define cpmcom    CpmRoot::com
#define cpmcon    CpmRoot::con
#define cpmdis    CpmRoot::dis
#define cpmran    CpmRoot::ran
#define cpmtes    CpmRoot::test
#define cpmhas    CpmRoot::hash
#define cpmnet    CpmRoot::net
#define cpmiva    CpmRoot::isVal
#define cpminv    CpmRoot::inv
#define cpmab2    CpmRoot::absSqr
#define cpmarg    CpmRoot::arg
#define cpmprec    CpmRoot::prec
#define cpmgetprec CpmRoot::getPrec

// Easy access to mathematical functions of a real argument.
// CPM_ is either std or mpfr or boost::multiprecision depending on CPM_MP
// and CPM_USE_MPREAL.
// Since the function-names begin with 'cpm' it should not harm that they
// are defined here in the global namespace.
// Since these function names are defined already in namespace std it would
// be not convenient to define them first in CpmRoot and that use, for
```

```
// instance,
// #define cpmsin    CpmRoot::sin
// Instead, we define a function with name cpmsin.

#if defined(CPM_MP)
   #if defined(CPM_USE_MPREAL)
      inline CpmRoot::R cpmabs(CpmRoot::R const& x)
         { return mpfr::abs(x);}
      inline CpmRoot::R cpmfloor(CpmRoot::R const& x)
         { return mpfr::floor(x);}
      inline CpmRoot::R cpmceil(CpmRoot::R const& x)
         { return mpfr::ceil(x);}
      inline CpmRoot::R cpmsin(CpmRoot::R const& x)
         { return mpfr::sin(x);}
      inline CpmRoot::R cpmasin(CpmRoot::R const& x)
         { return mpfr::asin(x);}
      inline CpmRoot::R cpmsinh(CpmRoot::R const& x)
         { return mpfr::sinh(x);}
      inline CpmRoot::R cpmcos(CpmRoot::R const& x)
         { return mpfr::cos(x);}
      inline CpmRoot::R cpmacos(CpmRoot::R const& x)
         { return mpfr::acos(x);}
      inline CpmRoot::R cpmcosh(CpmRoot::R const& x)
         { return mpfr::cosh(x);}
      inline CpmRoot::R cpmtan(CpmRoot::R const& x)
         { return mpfr::tan(x);}
      inline CpmRoot::R cpmtanh(CpmRoot::R const& x)
         { return mpfr::tanh(x);}
      inline CpmRoot::R cpmatan(CpmRoot::R const& x)
         { return atan(x);}
      inline CpmRoot::R cpmatan2(CpmRoot::R const& y, CpmRoot::R const& x)
         { return mpfr::atan2(y,x);}
      inline CpmRoot::R cpmexp(CpmRoot::R const& x)
         { return mpfr::exp(x);}
      inline CpmRoot::R cpmsqrt(CpmRoot::R const& x)
         { return mpfr::sqrt(x);}
      inline CpmRoot::R cpmlog(CpmRoot::R const& x)
         { return mpfr::log(x);}
      inline CpmRoot::R cpmlog10(CpmRoot::R const& x)
         { return mpfr::log10(x);}
      inline CpmRoot::R cpmpow(CpmRoot::R const& a, CpmRoot::R const& b)
         { return mpfr::pow(a,b);}
   #else
      inline CpmRoot::R cpmabs(CpmRoot::R const& x)
         { return boost::multiprecision::abs(x);}
      inline CpmRoot::R cpmfloor(CpmRoot::R const& x)
         { return boost::multiprecision::floor(x);}
      inline CpmRoot::R cpmceil(CpmRoot::R const& x)
         { return boost::multiprecision::ceil(x);}
      inline CpmRoot::R cpmsin(CpmRoot::R const& x)
```

```
         { return boost::multiprecision::sin(x);}
      inline CpmRoot::R cpmasin(CpmRoot::R const& x)
         { return boost::multiprecision::asin(x);}
      inline CpmRoot::R cpmsinh(CpmRoot::R const& x)
         { return boost::multiprecision::sinh(x);}
      inline CpmRoot::R cpmcos(CpmRoot::R const& x)
         { return boost::multiprecision::cos(x);}
      inline CpmRoot::R cpmacos(CpmRoot::R const& x)
         { return boost::multiprecision::acos(x);}
      inline CpmRoot::R cpmcosh(CpmRoot::R const& x)
         { return boost::multiprecision::cosh(x);}
      inline CpmRoot::R cpmtan(CpmRoot::R const& x)
         { return boost::multiprecision::tan(x);}
      inline CpmRoot::R cpmtanh(CpmRoot::R const& x)
         { return boost::multiprecision::tanh(x);}
      inline CpmRoot::R cpmatan(CpmRoot::R const& x)
         { return atan(x);}
      inline CpmRoot::R cpmatan2(CpmRoot::R const& y, CpmRoot::R const& x)
         { return boost::multiprecision::atan2(y,x);}
      inline CpmRoot::R cpmexp(CpmRoot::R const& x)
         { return boost::multiprecision::exp(x);}
      inline CpmRoot::R cpmsqrt(CpmRoot::R const& x)
         { return boost::multiprecision::sqrt(x);}
      inline CpmRoot::R cpmlog(CpmRoot::R const& x)
         { return boost::multiprecision::log(x);}
      inline CpmRoot::R cpmlog10(CpmRoot::R const& x)
         { return boost::multiprecision::log10(x);}
      inline CpmRoot::R cpmpow(CpmRoot::R const& a, CpmRoot::R const& b)
         { return boost::multiprecision::pow(a,b);}
   #endif
#else
      inline CpmRoot::R cpmabs(CpmRoot::R const& x)
         { return std::abs(x);}
      inline CpmRoot::R cpmfloor(CpmRoot::R const& x)
         { return std::floor(x);}
      inline CpmRoot::R cpmceil(CpmRoot::R const& x)
         { return std::ceil(x);}
      inline CpmRoot::R cpmsin(CpmRoot::R const& x)
         { return std::sin(x);}
      inline CpmRoot::R cpmasin(CpmRoot::R const& x)
         { return std::asin(x);}
      inline CpmRoot::R cpmsinh(CpmRoot::R const& x)
         { return std::sinh(x);}
      inline CpmRoot::R cpmcos(CpmRoot::R const& x)
         { return std::cos(x);}
      inline CpmRoot::R cpmacos(CpmRoot::R const& x)
         { return std::acos(x);}
      inline CpmRoot::R cpmcosh(CpmRoot::R const& x)
         { return std::cosh(x);}
      inline CpmRoot::R cpmtan(CpmRoot::R const& x)
```

```
         { return std::tan(x);}
      inline CpmRoot::R cpmtanh(CpmRoot::R const& x)
         { return std::tanh(x);}
      inline CpmRoot::R cpmatan(CpmRoot::R const& x)
         { return atan(x);}
      inline CpmRoot::R cpmatan2(CpmRoot::R const& y, CpmRoot::R const& x)
         { return std::atan2(y,x);}
      inline CpmRoot::R cpmexp(CpmRoot::R const& x)
         { return std::exp(x);}
      inline CpmRoot::R cpmsqrt(CpmRoot::R const& x)
         { return std::sqrt(x);}
      inline CpmRoot::R cpmlog(CpmRoot::R const& x)
         { return std::log(x);}
      inline CpmRoot::R cpmlog10(CpmRoot::R const& x)
         { return std::log10(x);}
      inline CpmRoot::R cpmpow(CpmRoot::R const& a, CpmRoot::R const& b)
         { return std::pow(a,b);}
#endif

// namespace re-opened
namespace CpmRoot{
// Numerical precision matters.
   extern Z numPrc;
      //: numerical precision
      // value for decimals used for the internal representation
      // of real numbers of type R. This has an effect only if
      // compilation is done with defined(CPM_USE_MPREAL).

// Formatting matters.

   extern Z wrtPrc;
      //: writing precision
      // value for decimals written to strings for real
      // numbers of type R. Notice that all data communicated
      // between processes in my MPI-based parallel computing
      // functions use writing on streams; so numerical errors in
      // communication are ignorable only if this number is large
      // e.g. 16. Now these functions set wrtPrc high and reset
      // it afterwards to a normal value (automatically!)

   extern bool wrtTit;
      //: write title
      // if this is true, all but the elementary tyes get written on
      // stream with a title preceeded by a comment indicator such as
      // '//'

   bool writeTitle(const string& , ostream&);
      // write title
      // Uses the string argument to write a title which is precceced
      // by a comment line indicator. Is under the control of wrtTit.
```

```
   bool startsComment(char c);
      // starts comment
      // Returns true if c is a character which gives a line the
      // meaning of a comment if it is its first non-whitespace
      // character.
      // Presently startsComment(c)==true for
      // c \in { ';','/','#','*'}

   bool eatComments(istream& in);
      // eat comments
      // Returns false if no character not belonging to a comment
      // was found. If return is true, the stream is now in a
      // state that the next reading access will find something
      // not belonging to a comment.

// Unified interface to types Z,N,R,L,bool,string,
// which we refer to as bb-types (basic built-in types). Notice that the
// types Z, N, and L are defined via typedef and thus are no classes.
// R may be a class - depending on the macro CPM_MP.
// The types bool and string from standard C++ are useful as they stand.
// Nevertheless they will be wrapped into classes - CpmRootX::B in
// cpmtypes.h and class CpmRoot::Word in file cpmword.h.
// --------

   bool strVal(istream& str);
      //: stream validity
      // returns stream status

   bool strVal(ostream& str);
      //: stream validity
      // returns stream status

// Topic 1: reading instances of b-types from streams that may contain
// comments.

   template <class T>
   bool read(T& t, istream& in)
   //: read
   // reads a T from a istream; in doing so it looks out for characters
   // having the meaning of 'comment indicators'. After having identified
   // such a character, the reading action will jump to the beginning of
   // the next line (thus not reading the rest of the previous line,
   // which is supposed to be a comment not containing data for reading).
   // Of course, a comment indicator there, will result in skipping that
   // line too. See function startsComment for the characters that are
   // considered comment indicators.
   // If no T can be read the function returns
   // false otherwise true. Providing a program with numerical data from
   // a text file created by the programmer is useful only if the file is
```

```
   // structured by comments explaining the meaning of the data. Thus
   // reading from files containing comment lines is crucial.
   // On ENUMERATIONS:
   // Not only quantities of type Z but also enumerations will be
   // read by this function. Then the syntax of writing and reading
   // should be as follows:
/*
   enum Corners{LL,UL,LR,UR};
   Corners c;
   write((Z)c,cout);
   ...
   Z temp;
   read(temp,cin);
   Corners c2=(Corners)temp;
*/
   {
      if  (!eatComments(in)) return false;
      if (in>>t) return true; else return false;
   }

   template <>
   bool read(L& x, istream& in);

   bool readLine(string& str, istream&);
   // reads a non-comment line from a istream into str. Trailing
   // whitespace such as the most annoying such species CR (carriage
   // return) will not find a way to spoil str. Leading whitespace
   // will not be ignored here since it is essential for copying
   // texts in a readable form by reading and printing lines

// Topic 2: writing instances of b-types to streams
// in a similar syntax, to make large prnOn and scanFrom functions of
// classes more alike

   template <class T>
   bool write(T const& t, ostream& out)
   {
      out<<t<<endl; return strVal(out);
   }

   template <>
   bool write(L const& t, ostream& out);

   template <>
   bool write(R const& t, ostream& out);
      // special treatment to avoid problems with small
      // numbers (which should not exist)

// Miscellanea
```

```
   bool isVal(L x);
   bool isVal(Z x);
   bool isVal(N x);
      // No modes of invalidity of types L, bool, or string are
      // known to me. The real types are already treated.

   L getByte(Z z, Z n);
      // z is an (typically 32 bit)-integer and n varies over 1,2,3,4
      // returned is the first, second, third, forth byte (counted from
      // left to right)

   string skipLeadingWhitespace(string const& str);
      // clear from name

   string skipTrailingWhitespace(string const& str);
      // clear from name

///////////////// interface service classes //////////////////////////////
// Unified interface to the bb-types
//    Z, N, R, L, bool, string
// and most of the other C+- classes.
// This uniform interface is given by the interface service class
// templates
//    IO<T>, Comp<T>, Inv<T>, AbsSqr<T>, Abs<T>, Dis<T>,
//    Test<T>, Ran<T>, Hash<T>, Conj<T>, Neutrals<T>,
//    ToWord<T>, Name<T> (these two to be added later in cpmword.h)
// For a class T to implement this interface means to define the following
// member functions ( here an T as argument may also mean T const&):
//    bool prnOn(ostream& str)const;         // print on
//    bool scanFrom(istream& str);           // scan from
//    Z com(T)const;                         // compare
//    T inv()const;                          // inverse
//    R absSqr()const;                       // absolute (value) squared
//    R abs()const;                          // absolute value
//    R dis(T)const;                         // distance value
//    T test(Z)const;                        // test value
//    T ran(Z)const;                         // random value
//    Z hash()const;                         // hash value
//    T con()const;                          // conjugate
//    Z net(Z)const;                         // neutrals ( = 0 and 1)
//    Word nameOf()const;                   // class name (see cpmword.h)
//    Word toWord()const;                   // string representation of
                                            // the value  (see cpmword.h)
// For the non-class b-types, the interface service class templates are
// subsequently defined as specializations.

////////////////////////// class IO<> ////////////////////////////////////
// Here we describe for the important example of stream IO
// (marshalling) the method by which C+- succeeds to treat
// the non-class b-types and C+- classes on equal
```

```
// footing when these are used as template arguments of
// C+- array classes.
// For a C+- class T it is normal to define the member functions
// (1) bool T::prnOn(ostream& str)const;
// (2) bool T::scanFrom(istream& str);
// Their declaration (together with the definition of some
// related non-member functions) is provided by using the
// macro CPM_IO after a statement 'typedef T Type;'
// For a C+- class template, e.g. CpmArrays::V<T> it is  n o t
// natural to assume that T defines the member functions
// (1) and (2) since T could be Z or R and thus not a class at all.
// Instead, all C+- class templates which implement
// interaction with streams assume that the template argument T
// defines functions
// (3) bool CpmRoot::IO<T>().o(T const& t, ostream& str)const;
// (4) bool CmRoot::IO<T>().i(T& t, istream& str)const;
// There are two possibilities for these functions to get defined:
// (i) It could be that functions (1),(2) are defined for T
//    e.g. since T is a C+- class implementing CPM_IO.
// Then the following un-specialized version of the class template IO<T>
// does the job.
// If, however, T is not a C+- class and cannot be modified by
// adding member functions (1),(2) then
// (ii) a specialization of the class template
//    IO has to make IO<T> defined: Such specializations
//    are given in the following for the  bb-types.
//    Partial specializations (in which T is a class template)
//    can be defined where needed. An example is the definition
//    of IO< std::vector<T> > in cpmv.h.
// Finally there are class-less function templates (near the end of the
// file)
// bool prnT(T const& t, ostream& str)
// bool scanT(T& t, istream& str)
// which will be called in the covenient read and write macros
// #define cpms(X) if (!CpmRoot::scanT((X),str)) return false
// #define cpmp(X) if (!CpmRoot::prnT((X),str)) return false
// fom file cpminterfaces.h. These functions can also be used in
// template code instead of the more clumpsy functions (3),(4).
// It should be noted that none of the interface service classes
// IO, Comp, Inv, AbsSqr, Abs, Dis, Test, Ran,
// Hash, Conj, Neutrals contains its own data, so that no transport
// of data is involved in their use. Notice that there are two more
// such service classes: Name and ToWord in cpmword.h.
// On Name the C+- type naming system is based, and ToName provides
// convenient short output string representation of values.
// The send and receive functions for MPI usage are based on the
// present IO-class (see macros CPM_MPI_O and #define CPM_MPI_I
// in cpminterfaces.h).
// For all such service classes except IO there is only one member
// function (not two, namely i and o, in IO) and this is chosen to
```

```
// be operator().

template<class T>
class IO{ // input output class template
public:
   IO(){}
   bool o(T const& t, ostream& str)const
      { return t.prnOn(str);}
   bool i(T& t, istream& str)const
      { return t.scanFrom(str);}
};

template<>
class IO<R>{ // specialization of IO
public:
   IO(){}
   bool o(R const& t, ostream& str)const
      { return CpmRoot::write(t,str);}
   bool i(R& t, istream& str)const
      { return CpmRoot::read(t,str);}
};

template<>
class IO<Z>{ // specialization of IO
public:
   IO(){}
   bool o(Z const& t, ostream& str)const
      { return CpmRoot::write(t,str);}
   bool i(Z& t, istream& str)const
      { return CpmRoot::read(t,str);}
};

template<>
class IO<N>{ // specialization of IO
public:
   IO(){}
   bool o(N const& t, ostream& str)const
      { return CpmRoot::write(t,str);}
   bool i(N& t, istream& str)const
      { return CpmRoot::read(t,str);}
};

template<>
class IO<L>{ // specialization of IO
public:
   IO(){}
   bool o(L const& t, ostream& str)const
      { return CpmRoot::write(t,str);}
   bool i(L& t, istream& str)const
      { return CpmRoot::read(t,str);}
```

```
};

template<>
class IO<bool>{ // specialization of IO
public:
   IO(){}
   bool o(bool const& t, ostream& str)const
      { return CpmRoot::write(t,str);}
   bool i(bool& t, istream& str)const
      { return CpmRoot::read(t,str);}
};

template<>
class IO<string>{ // specialization of IO
public:
   IO(){}
   bool o(string const& t, ostream& str)const
      { return CpmRoot::write(t,str);}
   bool i(string& t, istream& str)const
      { return CpmRoot::read(t,str);}
};

////////////////////////// class Comp<> ////////////////////////////////////
// class is named Comp instead of Com in order to avoid confusion
// with CpmMPI::Com
// Ruby style comparison
// com = 'compared' corresponds to the negative of
// Ruby's intelligent operator <=> (only the symbol '<=>' together
// with the ordered list -1,0,1 suggests Ruby's choice; if one is not
// bound by a symbol it is more natural to associate a<b (odered) with
// a.com(b)==1 and a>b (converse order) with a.com(b)==-1.
// Thus our normation:
// a.com(b) ==  0  for  a == b
// a.com(b) ==  1  for  a <  b
// a.com(b) == -1  for  a >  b

template<class T>
class Comp{ // compare
public:
   Comp(){}
   Z operator()(T const& x, T const& y){ return x.com(y);}
};

// Specializations building on > and < :
#define CPM_SC\
 if (x<y) return 1; if (x>y) return -1; return 0

template<>
class Comp<R>{
public:
```

```
   Comp(){}
   Z operator()(R const& x, R const& y){CPM_SC;}
};

template<>
class Comp<Z>{
public:
   Comp(){}
   Z operator()(Z const& x, Z const& y){CPM_SC;}
};

template<>
class Comp<N>{
public:
   Comp(){}
   Z operator()(N const& x, N const& y){CPM_SC;}
};

template<>
class Comp<L>{
public:
   Comp(){}
   Z operator()(L const& x, L const& y){CPM_SC;}
};

template<>
class Comp<bool>{
public:
   Comp(){}
   Z operator()(bool const& x, bool const& y){CPM_SC;}
};

template<>
class Comp<string>{
public:
   Comp(){}
   Z operator()(string const& x, string const& y){CPM_SC;}
};

#undef CPM_SC

/////////////////////////// class Inv<> ////////////////////////////////////

template<class T>
class Inv{ // inverse
public:
   Inv(){}
   T operator()(T const& t)const{ return t.inv();}
};
```

```
template<>
class Inv<R>{ // inverse
public:
   Inv(){}
   R operator()(R const& t)const{ return cpminv(t);}
};

template<>
class Inv<Z>{ // inverse
public:
   Inv(){}
   Z operator()(Z const& t)const{ t; return Z(0);}
};

template<>
class Inv<N>{ // inverse
public:
   Inv(){}
   N operator()(N const& t)const{ t; return N(0);}
};

template<>
class Inv<L>{ // inverse, defined in an arbitrary manner
public:
   Inv(){}
   L operator()(L const& t)const{ return t;}
};

template<>
class Inv<bool>{ // inverse
public:
   Inv(){}
   bool operator()(bool const& t)const{ return !t;}
};

template<>
class Inv<string>{ // reverse string
public:
   Inv(){}
   string operator()(string const& t)const
   {
      string::const_reverse_iterator p1=t.rbegin(), p2=t.rend();
      return string(p1,p2);
   }
};

////////////////////////// class Test<> ///////////////////////////////
// The intented usage is as follows:
//    using CpmRoot;
//    T t;
```

```
//    Z complexity=3; // for example
//    T tc=Test<T>()(t,complexity);
//    T t1=Ran<T>()(tc,1);
//    T t2=Ran<T>()(tc,2);
//    T t3=Ran<T>()(tc,3);
// which is an unbiased way to create an arbitrarily large set of T-values.

template<class T>
class Test{ // test value
public:
   Test(){}
   T operator()(T const& t, Z complexity)const
   { return t.test(complexity);}
};

template<>
class Test<R>{
public:
   Test(){}
   R operator()(R const& x, Z complexity)const
   { return cpmtes(x,complexity);}
};

template<>
class Test<Z>{
public:
   Test(){}
   Z operator()(Z const& x, Z complexity)const
   { x; return complexity;}
};

template<>
class Test<N>{
public:
   Test(){}
   L operator()(N const& x, Z complexity)const
   { x; Z cp=posVal(complexity); return N(cp);}
};

template<>
class Test<L>{
public:
   Test(){}
   L operator()(L const& x, Z complexity)const
   { x; Z cp=posVal(complexity); return L(cp%256);}
};

template<>
class Test<bool>{
public:
```

```
   Test(){}
   bool operator()(bool const& x, Z complexity)const
   { x; Z cp=posVal(complexity); return cp%2==1;}
};

template<>
class Test<string>{
public:
   Test(){}
   string operator()(string const& x, Z complexity)const
   {
      x;
      if (complexity<=1)
         return "az";
      else if (complexity==2)
         return "abcdefghijklmnopqrstuvwxyz";
      else if (complexity==3)
         return "abcdefghijklmnopqrstuvwxyz0123456789";
      else return
         "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
   }
};

////////////////////// class Ran<> ////////////////////////////////

Z randomZ(Z n, Z j=0);
   // returns uniformly distributed integers in the natural
   // index range [1,n]. Thus stops with error if n<1.
   // Function IvZ::ran allows to set lower limit and upper limit
   // arbitrarly.

template<class T>
class Ran{ // random value
public:
   Ran(){}
   T operator()(T const& t, Z j=0)const
   { return t.ran(j);}
};

template<>
class Ran<R>{
public:
   Ran(){}
   R operator()(R const& x, Z j=0)const
   {
     return cpmran(x,j);
   }
};

template<>
```

```
class Ran<Z>{
public:
   Ran(){}
   Z operator()(Z const& i, Z j=0)const
   {
      R y=randomR(j);
      Z ia=(i>=0 ? i : -i);
      if (ia==0) ia=1;
      // without this modification we would get 0,0,0,... for i=0. This
      // 'random' sequence will probably never be needed in a serious
      // context.
      // More probably i itself will result from a random process and
      // nevertheless ran(i,i2) will be expected to be random in i2.
      // Thus we increase ia in order to get the random sequence with
      // values in {-1,0,1}.
      Z n=ia*2+1;
      y*=n; // belongs to [0,2|i|+1)
      Z res=cpmtoz(y);
      if (res==n) res--; // should not happen
      return res-ia;
   }
};

template<>
class Ran<N>{
public:
   Ran(){}
   N operator()(N const& i, Z j=0)const
   {
      R y=randomR(j);
      Z ip=(i==0 ? 1 : i);
      // without this modification we would get 0,0,0,... for i=0. This
      // 'random' sequence will probably never be needed in a serious
      // context.
      // More probably i itself will result from a random process and
      // nevertheless ran(i,i2) will be expected to be random in i2.
      // Thus we increase ip in order to get the random sequence with
      // values in {0,1}.
      y*=ip; // belongs to [0,ip)
      Z res=cpmtoz(y);
      if (res==ip) res--; // should not happen
      return N(res); // belongs to {0,1} for i==0 and to {0,...i-1} else.
   }
};

template<>
class Ran<L>{
public:
   Ran(){}
   L operator()(L const& x, Z j=0)const
```

```
   {
      x;
      R y=randomR(j);
      y*=256; // belongs to [0,256)
      Z zy=cpmtoz(y); // belongs to {0,1,..., 255}=
      if (zy==256) zy--; // should not happen
      return (L)zy;
   }
};

template<>
class Ran<bool>{
public:
   Ran(){}
   bool operator()(bool const& b, Z j=0)const
   {
      R y=randomR(j);
      if (y<=0.5) return b; else return !b;
   }
};

// Here, all random strings have only printable ascii characters. See the
// definition of myChars for details.
template<>
class Ran<string>{
public:
   Ran(){}
   string operator()(string const& a, Z i=0)const
   {
      string res=a;
      Z l=i;
      string::const_iterator j;
      string::iterator k;
      const char myChars[] {'a','b','c','d','e','f','g','h','i','j','k',
      'l','m','n','o','p','q','r','s','t','u','v','w','x','y','z',
      'A','B','C','D','E','F','G','H','I','J','K',
      'L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z',
      '0','1','2','3','4','5','6','7','8','9','_','.'};
      const int nc=64;
      for (j=a.begin(),k=res.begin();j!=a.end(),k!=res.end();++j,++k){
         int z = *j;
         int r = z%nc + 1;
         int ii = i;
         if (ii!=0){
            l++;
            ii+=l;
         }
         int q = randomZ(nc,ii) + r;
         *k = myChars[q%nc];
      }
```

```
      return res;
   }
};

//////////////////// class Hash<> ////////////////////////////////////
// If for T t1,t2 one has hash(t1)==hash(t2) there
// should be a overwhelming probability that t1 is equal
// to t2 (unless t2 is obtained from t1 by only a minor
// modification, or by a modification which is aware
// of the algorithm for hash).

template<class T>
class Hash{ // hash value
public:
   Hash(){}
   Z operator()(T const& t)const
   { return t.hash();}
};

template<>
class Hash<R>{
public:
   Hash(){}
   Z operator()(R const& x)const
   {
      return cpmhas(x);
   }
};

template<>
class Hash<Z>{
public:
   Hash(){}
   Z operator()(Z const& x)const
   {  return x-x%137+(x/13)*7;}
};

template<>
class Hash<N>{
public:
   Hash(){}
   N operator()(N const& x)const
   {  return x+x%137+(x/13)*7;}
};

template<>
class Hash<L>{
public:
   Hash(){}
   Z operator()(L const& x)const
```

```
   {
      Z y=(Z)x+117;
      return Hash<Z>()(y);
   }
};

template<>
class Hash<bool>{
public:
   Hash(){}
   Z operator()(bool const& x)const
   { return x ? 137 : 13;}
};

template<>
class Hash<string>{
public:
   Hash(){}
   Z operator()(string const& x)const
   {
      Z sum=137;
      string::const_iterator i;
      for (i=x.begin();i!=x.end();++i) sum+=(Z)(*i);
      return Hash<Z>()(sum);
   }
};

//////////////////// class Conj<> ////////////////////////////////////////
// This describes a concept that does not convincingly arrises
// from the types under present consideration. It will
// represent complex conjugation of complex numbers, transposition
// of real matrices, and Hermitean conjugation of complex
// matrices.

template<class T>
class Conj{ // conjugation
public:
   Conj(){}
   T operator()(T const& t)const
   { return t.con();}
};

template<>
class Conj<R>{
public:
   Conj(){}
   R operator()(R const& t)const
   { return cpmcon(t);}
};
```

```
template<>
class Conj<Z>{
public:
   Conj(){}
   Z operator()(Z const& t)const
   { return t;}
};

template<>
class Conj<N>{
public:
   Conj(){}
   N operator()(N const& t)const
   { return t;}
};

template<>
class Conj<L>{
public:
   Conj(){}
   L operator()(L const& t)const
   { return t;}
};

template<>
class Conj<bool>{
public:
   Conj(){}
   bool operator()(bool const& t)const
   { return t;}
};

template<>
class Conj<string>{
public:
   Conj(){}
   string operator()(string const& t)const
   { return t;}
};

//////////////////// class Neutrals<> ////////////////////////////////////
// for T t; the quantity Neutrals<T>(t,0) is the
// instance of T that is zero or comes as close as possible
// to realising the concept of a zero of type T.
// Correspondingly Neutrals<T>(t,1) with zero
// replaced by unity. The name comes from the fact that
// zero is a neutral element for addition and
// unity is a neutral element for multiplication.
// This renders the concept ambiguous for T==string since
// here we do refer only to addition (concatenation)
```

```
// but not to multiplication.

template<class T>
class Neutrals{ // neutral elements
public:
   Neutrals(){}
   T operator()(T const& t, Z i)const
   { return t.net(i);}
};

template<>
class Neutrals<R>{
public:
   Neutrals(){}
   R operator()(R const& t, Z i)const
   { return cpmnet(t,i);}
};

template<>
class Neutrals<Z>{
public:
   Neutrals(){}
   Z operator()(Z const& t, Z i)const
   { t; return i==1 ? 1 : 0;}
};

template<>
class Neutrals<N>{
public:
   Neutrals(){}
   N operator()(N const& t, Z i)const
   { t; return i==1 ? N(1) : N(0);}
};

template<>
class Neutrals<L>{
public:
   Neutrals(){}
   L operator()(L const& t, Z i)const
   { t; return i==1 ? L(1) : L(0);}
};

template<>
class Neutrals<bool>{
public:
   Neutrals(){}
   bool operator()(bool const& t, Z i)const
   { t; return i==1 ? true : false;}
};
```

```
template<>
class Neutrals<string>{
public:
   Neutrals(){}
   string operator()(string const& t, Z i)const
   { t; return i==1 ? "1" : "";}
};

// Now some classes will be defined which use in their interface
// a common real type, which is chosen as R. This type appears here
// as a return value of operator() of classes
// AbsSqr<>, Abs<>, and Dis<>. The definition of class Root<> in
// cpmword.h puts this role into perspective.
////////////////// class AbsSqr<> ////////////////////////////////////////

template<class T>
class AbsSqr{ // absolute (value) squared
public:
   AbsSqr(){}
   R operator()(T const& t)const{ return t.absSqr();}
};

template<>
class AbsSqr<R>{ // absolute (value) squared
public:
   AbsSqr(){}
   R operator()(R const& x)const{ return cpmab2(x);}
};

template<>
class AbsSqr<Z>{ // absolute (value) squared
public:
   AbsSqr(){}
   R operator()(Z const& x)const{ return R(x)*R(x);}
};

template<>
class AbsSqr<N>{ // absolute (value) squared
public:
   AbsSqr(){}
   R operator()(N const& x)const{ return R(x)*R(x);}
};

template<>
class AbsSqr<L>{ // absolute (value) squared
public:
   AbsSqr(){}
   R operator()(L const& x)const{
      Z y = static_cast<Z>(x);
      R r = y;
```

```
      return r * r;
   }
};

template<>
class AbsSqr<bool>{ // absolute (value) squared
public:
   AbsSqr(){}
   R operator()(bool const& x)const
      { return x==false ? R(0) : R(1);}
};

template<>
class AbsSqr<string>{
   // here string is interpreted as an array of L's
public:
   AbsSqr(){}
   R operator()(string const& x)const
   { // string components are cased to type L
      R res=0;
      string::const_iterator i;
      for (i=x.begin();i!=x.end();++i) res+=AbsSqr<L>()((L)(*i));
      return res;
   }
};

////////////////// class Abs<> ////////////////////////////////////////

template<class T>
class Abs{ // absolute value
public:
   Abs(){}
   R operator()(T const& t)const{ return t.abs();}
};

template<>
class Abs<R>{ // absolute value
public:
   Abs(){}
   R operator()(R const& x)const{ return cpmabs(x);}
};

template<>
class Abs<Z>{ // absolute value
public:
   Abs(){}
   R operator()(Z const& x)const{ return x<0 ? R(-x) : R(x) ;}
};

template<>
```

```
class Abs<N>{ // absolute value
public:
   Abs(){}
   R operator()(N const& x)const{ return R(x);}
};

template<>
class Abs<L>{ // absolute value
public:
   Abs(){}
   R operator()(L const& x)const{ return R(x);}
};

template<>
class Abs<bool>{ // absolute value
public:
   Abs(){}
   R operator()(bool const& x)const{ return x==false ? R(0) : R(1) ;}
};

template<>
class Abs<string>{
   // here string is interpreted as an array of L's
public:
   Abs(){}
   R operator()(string const& x)const
   { return sqrt(AbsSqr<string>()(x)) ;}
};

////////////////////// class Dis<> ////////////////////////////////////
// For T t1,t2 the real number Dis<T>()(t1,t2) belongs to [0,1]
// and tries to assess whether t1 and t2 are significantly different
// or differ only by numerical noise. Thus for discrete types we
// set 0 for t1==t2 and 1 else. For 'continuous types' we use
// the natural construct dis(x,y)=Min( |x-y|, |x-y|/(|x|+|y|) )
// = distFunc(|x|,|y|,|x-y|)
// based on the concepts of absolute value and difference.

template<class T>
class Dis{
public:
   Dis(){}
   R operator()(T const& x, T const& y){ return x.dis(y);}
};

template<>
class Dis<R>{
public:
   Dis(){}
   R operator()(R const& x, R const& y)
```

```
   { return cpmdis(x,y);}
};

template<>
class Dis<Z>{
public:
   Dis(){}
   R operator()(Z const& x, Z const& y)
   { return x!=y ?  1. : 0.;}
};

template<>
class Dis<N>{
public:
   Dis(){}
   R operator()(N const& x, N const& y)
   { return x!=y ?  1. : 0.;}
};

template<>
class Dis<L>{
public:
   Dis(){}
   R operator()(L const& x, L const& y)
   { return x!=y ?  1. : 0.;}
};

template<>
class Dis<bool>{
public:
   Dis(){}
   R operator()(bool const& x, bool const& y)
   { return x!=y ?  1. : 0.;}
};

template<>
class Dis<string>{
public:
   Dis(){}
   R operator()(string const& x, string const& y)
   { return x!=y ?  1. : 0.;}
};

bool apprInt(R x, R tol);
   // returns true if the distance between x and the nearest
   // integer is <= tol (if tol<0, this is never the case)

////////////////////////// class Conv ////////////////////////////////////
template<class Ti, class Tf>
class Conv{
```

```
// conversion
// Allows to define conversion between basic types in a manner
// which differes from that provided by the compiler.
   Ti x_;
public:
   Conv(Ti const& t):x_(t){}
   Tf operator()(void){ return (Tf)x_;}
};

template<>
class Conv<R,L>{
   // The image classes Image<T> and MultiImage<T> have to convert
   // between T and R in some functions. By far the most important
   // case is T=R so that no real conversion is involved. However
   // also T = L occurs. This case is handled here.
   R x_;
public:
   Conv(R const& t):x_(t){}
   L operator()(void){ return (L)cpmrnd(x_);}
};

template<>
class Conv<L,R>{
   L x_;
public:
   Conv(L const& t):x_(t){}
   R operator()(void){ return (R)(Z)x_;}
};

////////////////////// class-less function templates ////////////////////////
// This is a convenient form for using the service functions:
// The final T in the name is to suggest "template" and avoids name
// clashes with functions already defined for T = R with names inv, abs,
// hash, ran, ...
// See the comments on template <class T> class Root at the end of this
// file. Calling these functions is possible without mentioning the type
// of the argument, e.g. prnT(t,cout) which makes them usable in macros
// such as cpmp(X) and cpms(X).

// implementation of CPM_IO
template<class T>
bool prnT(T const& t, ostream& str)
   // print on
{ return CpmRoot::IO<T>().o(t,str);}

template<class T>
bool scanT(T& t, istream& str)
   // scan from
{ return CpmRoot::IO<T>().i(t,str);}
```

```
// implementation of CPM_ORDER
template<class T>
Z comT(T const& t1, T const& t2){ return Comp<T>()(t1,t2);}
   //: compare

// implementation of CPM_TEST_ALL
template<class T>
T invT(T const& t){ return Inv<T>()(t);}
   //: inverse

template<class T>
R absSqrT(T const& x){ return AbsSqr<T>()(x);}
   // absolute (value) squared

template<class T>
R absT(T const& x){ return Abs<T>()(x);}
   // absolute value

template<class T>
R disT(T const& t1, T const& t2){ return Dis<T>()(t1,t2);}
   // distance value

template<class T>
T testT(T const& t, Z complexity){ return Test<T>()(t,complexity);}
   // test value

template<class T>
T ranT(T const& t, Z j=0){ return Ran<T>()(t,j);}
   // random value

template<class T>
Z hashT(T const& t){ return Hash<T>()(t);}
   //: hash value

template<class T>
T conT(T const& t){ return Conj<T>()(t);}
   //: conjugate

template<class T>
T netT(T const& t, Z i){ return Neutrals<T>()(t,i);}
   // neutrals
} // CpmRoot

// A probably more convenient collection of these service functions
// is given by the class template CpmRoot::Root<> defined in cpmword.h.
// The class Root<T> has for each of the previous class-less functions
// a member function of the same functionality. For instance:
// T t1 = ... ;
// T t2 = ... ;
// R d = dis(t1,t2); ( = cpmdis(t1,t2), see the defines above )
```

```
// could also be written
// R d = Root<T>(t1).dis(t2);
// Here the Root-template needs the template argument T, which for the
// function template is optional (we could also write
// R d = disT<T>(t1,t2);).
// This explicit indication of the template argument helps more the reader
// than the compiler (who should know what to do anyway).
// Notice that definition of the I/O macro cpmp(X) and cpms(X) is
// easily based on functions prnOnT and scanFromT. As defined in
// cpminterfaces.h:
// #define cpmp(X) if (!prnOnT((X),str)) return false
// Probably also
// Root<typeid((X))>((X)).prnOn(str) would work. But I want to avoid
// the overhead incured by using typeid.

#endif
```

24 cpmnumbers.cpp

```
//? cpmnumbers.cpp
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#include <cpmnumbers.h>
#include <cpmsystem.h>
#include <cpmsystemdependencies.h>
   // defines wrtPrc, wrtTit
#include <cpmmacros.h>
   // CPM_MA, CPM_MZ

using namespace CpmStd;
using CpmRoot::R;
using CpmRoot::Z;
using CpmRoot::N;
using CpmRoot::L;
using CpmRoot::Word;

#if defined(CPM_MP)
   Z CpmRoot::numPrc=CPM_MP;
#else
   Z CpmRoot::numPrc=0;
#endif

Z tenToTwo(Z p){ return (Z)(p*3.3219281 + 0.5);}
Z twoToTen(Z d){ return (Z)(d/3.3219281 + 0.5);}

void CpmRoot::prec(Z p)
{
#if defined(CPM_MP)&&defined(CPM_USE_MPREAL)
   R::set_default_prec(tenToTwo(p));
#endif
   // set_default_prec(...) defined in mpreal.h
}
```

```
Z CpmRoot::getPrec()
{
#if defined(CPM_MP)
   #if defined(CPM_USE_MPREAL)
      return twoToTen(R::get_default_prec());
         // get_default_prec(...) defined in mpreal.h
   #else
      return CPM_MP;
   #endif
#else
   return twoToTen(8*sizeof(R));
#endif
}

Z CpmRoot::getPrec(R const& r)
{
#if defined(CPM_MP)
   #if defined(CPM_USE_MPREAL)
      return twoToTen(r.get_prec());
   #else
      return CPM_MP;
   #endif
#else
   return twoToTen(8*sizeof(R));
#endif
}

namespace{
// template tool for implementation
template <class T>
inline bool isValFunc(T x) // inline is needed
{
   if (cpmdbg>0){ // so calling this function can be switched
   // to idle from a central place.
      ostringstream ost;
      ost<<x;
      string s=ost.str();
      bool b1=s.find("I")==string::npos; // true if not found!
      bool b2=s.find("N")==string::npos;
      bool b3=s.find("n")==string::npos;
      //return b1||b2||b3; // corrected 2014-01-13
      return b1&&b2&&b3; // needs to give true for all options
   }
   else return true;
}

} // namespace

bool CpmRoot::isVal(L x){return true;}
```

```
bool CpmRoot::isVal(Z x){return isValFunc(x);}
bool CpmRoot::isVal(N x){return isValFunc(x);}

bool CpmRoot::isVal(R const& x)
{
#if defined(CPM_MP)
   if (cpmdbg>1){
      return !(isnan(x)||isinf(x));
   }
   else return true;
#else
   return isValFunc(x);
#endif
}

R CpmRoot::inv(R const& t)
{
   const Z maxMes=100;
   static Z mes=1;
   R z(0.);
   Word loc("inv(R): argument is 0");
   if (t==z){
      if (mes==maxMes){
         mes++;
         cpmmessage(loc&" ... messages discontinued");
      }
      if (mes<maxMes){
         mes++;
         cpmwarning(loc&", 0 returned");
      }
      return z;
   }
   R res=R(1.)/t;
   if (isVal(res)) return res;
   else {
      loc="inv(R): 1/argument is not valid";
      if (mes==maxMes){
         mes++;
         cpmmessage(loc&" ... messages discontinued");
      }
      if (mes<maxMes){
         mes++;
         cpmwarning(loc&", 0 returned");
      }
      return z;
   }
}

R CpmRoot::randomR(Z j)
{
```

```
   static N j0=137;
   static const R ranFac=1e6;
   R x = j!=0 ? (R)(j) : (R)(++j0); // no overflow possible due to
      // using unsigned integers as N
   R y = cpmsin(x)*ranFac;
   return y -= cpmfloor(y);
}

R CpmRoot::ran(R const& r, Z j)
{
   R fac = randomR(j);
   return r*(fac*2.0 - 1.);
}

Z CpmRoot::hash(R const& r)
{
   const R reg=100000;
   R a=cpmabs(r);
   return cpmrnd(r+reg/(a*reg+1));
}

R CpmRoot::test(R const& r, Z complexity)
{ return R(1.123456789)*complexity;}

Z CpmRoot::toZ(R const& r, bool toZero)
{
   if (toZero){
      Z s = cpmsgn(r);
      R x = cpmfloor(cpmabs(r));
      Z xi;
#if defined(CPM_MP)
   #if defined(CPM_USE_MPREAL)
      xi=x.toLong();
   #else
      xi=lround(x);
   #endif
#else
      xi=(Z)x;
#endif
      return s*xi;
   }
   else{
#if defined(CPM_MP)
      #if defined(CPM_USE_MPREAL)
         return (Z)cpmfloor(r).toLong();
      #else
         return (Z)lround(cpmfloor(r));
      #endif
#else
      return (Z)cpmfloor(r);
```

```
#endif
   }
}

R CpmRoot::dis(R const& r1, R const& r2)
{
   R a1 = cpmabs(r1);
   R a2 = cpmabs(r2);
   R a3 = cpmabs(r1-r2);
   return disDefFun(a1,a2,a3);
}

Z CpmRoot::randomZ(Z n, Z j)
{
   if (n<1) cpmerror("n<1 in CpmRoot::randomZ(Z n, Z j)");
   R y=n*randomR(j);
   Z res=toZ(y);
   return ((res==n) /* should not happen */ ? n : res+1);
}

bool CpmRoot::apprInt(R x, R tol)
{
   R d=x-rnd(x);
   return ( d>=0 ? d<=tol : -d<=tol);
}

namespace{
   L getbits(N x, N p, unsigned n)
   // Kernighan Ritchie: Programming in C, section 2.9
   { return (L)((x>>(p+1-n))& ~(~0<<n));}
}

L CpmRoot::getByte(Z z, Z n)
{
   N x=(N)z;
   return getbits(x, (n-1)*4,4);
}

#if defined(CPM_WRITE_PRECISION)
   Z CpmRoot::wrtPrc=CPM_WRITE_PRECISION;
#else
   Z CpmRoot::wrtPrc=10;
#endif

#if defined(CPM_WRITE_TITLE)
   bool CpmRoot::wrtTit=true;
#else
   bool CpmRoot::wrtTit=false;
#endif
```

```
bool CpmRoot::writeTitle(string const& s , ostream& out)
{
   if (wrtTit) CpmRoot::write(string("//")+s,out);
   //return (out!=0);
   return !out ? false : true;
}

string CpmRoot::skipLeadingWhitespace(string const& str)
{
   Z n=(Z)str.size();
   if (n==0) return str;
   char cTest=str[0];
   if ( !isspace(cTest)) return str;
      // fast part of the process, no leading whitespace there
// now the first character is space
   Z i=0;
   while(i<n && isspace(str[i])) i++;
// eliminate the i leading characters
   string res=str;
   res.erase(0,i);
   return res;
}

string CpmRoot::skipTrailingWhitespace(string const& str)
   // made 2004-10-12 in analogy to the previous function
{
   Z n=(Z)str.size();
   if (n==0) return str;
   char cTest=str[n-1];
   if ( !isspace(cTest)) return str;
      // fast part of the process, no trailing whitespace there
// now the last character is space
   Z iTest=n-1;
   Z found=0;
   while(iTest>=0 && isspace(str[iTest])){
      iTest--;
      found++;
   }
   // now iTest is -1 or the index of a character
   // so iTest++ is the insex of the first of the trailing
   // whitespaces
   iTest++;
// eliminate the found trailing characters
   string res=str;
   res.erase(iTest,found);
   return res;
}

bool CpmRoot::startsComment(char c)
{ return c=='/' || c==';'|| c=='*'|| c=='#';}
```

```
bool CpmRoot::eatComments(istream& in)
{
   Z mL=3;
   Word loc("CpmRoot::eatComments(istream&)");
   CPM_MA
   char c=0;
   string buff;
   if (!in){
      cpmmessage(mL,loc&": bad stream before reading");
      CPM_MZ
      return false;
   }
LOOP:
   if (!(in>>c)){
      ostringstream ost;
      ost<<loc<<":  bad stream after reading character c="<<(Z)c;
      cpmmessage(mL,Word(ost.str()));
      CPM_MZ
      return false;
   }
   if (startsComment(c)){
      std::getline(in,buff); // reads the line till end of line
         // including the '\n' into the self-adjusting buff
      goto LOOP;
   }  // now c is no longer the potential beginning of a comment
   in.putback(c);
   if (!in){
      cpmmessage(mL,loc&": bad stream after reading");
      CPM_MZ
      return false;
   }
   CPM_MZ
   return true;
}

bool CpmRoot::strVal(istream& str)
{ if (!str||str.bad()) return false; else return str.good(); }

bool CpmRoot::strVal(ostream& str)
{ if (!str||str.bad()) return false; else return str.good(); }

bool CpmRoot::readLine(string& line , istream& in)
{
   if  (!eatComments(in)) return false;
   std::getline(in,line);
   line=skipTrailingWhitespace(line);
   return in ? true : false;
}
```

```
template <>
bool CpmRoot::read(L& x, istream& in)
{
   Z z;
   bool suc=CpmRoot::read<Z>(z,in);
   x=(L)z;
   return suc;
}

template <>
bool CpmRoot::write(L const& x, ostream& out)
{
   out<<(Z)x<<endl;
   return strVal(out);
}

template <>
bool CpmRoot::write(R const& r, ostream& out)
// experimental avoidance of writing too small numbers
// which caused trouble under Windows
{
   const R wrTiny=1e-304;
      // 1e-305 works, 1e-310 works not
   out.precision(wrtPrc);
   R x=r;
   if (x>0){
      if (x<wrTiny) x=wrTiny;
   }
   else if (x<0){
      if (x>-wrTiny) x=-wrTiny;
   }
   else{
      x=0;
   }
   out<<x<<endl; return strVal(out);
}
```

25 cpmp.h

```
//? cpmp.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_P_H_
#define CPM_P_H_
/*

Purpose: defining classes of smart pointers, and polymorphic arrays.
classes Pp<>, Po<>, Vp<>.
class P<> is defined in cpmuc.h

Notice usage of the new macro definition (2017-05-23)
 #define const& const&
 from cpmbasictypes.h

*/

#include <cpmv.h>

namespace CpmArrays{

//////////////////////  class Pp< >  ////////////////////////////////
//
// class of non-constant smart pointers implementing 'copy on write' (see
// cpmv.h) and polymorphism (indicated by the 'p'). Let us first discuss
// the 'mono-morphic' aspects, i.e. we discuss the properties of Pp<T>
// for arbitrary but fixed T and not for a variety of T's:
//
// Let T be any non-abstact class that defines a member function
//
//       Tb* T::clone(void)const{ return new T(*this);}  (ii)
// (Notice that T has to define a copy constructor for this to work)
//
```

```
// where Tb is either T itself or a base class of T which declares
//
//       virtual *Tb Tb::clone(void)const{ return new Tb(*this);}
// or
//       virtual *Tb Tb::clone(void)const=0;
//
// Then, let us say that T is cloneable, with clone base Tb. Then the
// main property of the Pb<> template is:
//
//    cloneable T ==> value class Pb<T>
//
// If the source code of a non-abstract T is open to modifications, it
// can be made cloneable by inserting the definition
//    T* T::clone()const{ return new T(*this);}
// (lets ignore the complication that T may already use the name
//  clone in a conflicting manner).
//
// Properties of Pp<T> as a 'polymorphic container':
// -------------------------------------------------
//
// We consider a finite set L of cloneable classes which all have the
// same clone base T0. Let T1 be one of those and let L(T1) denote the
// classes in L which are derived from T1. Then Pp<T1> is a polymorphic
// container for L(T1) in the following sense:
// Let pp be an instance of Pp<T1>, created e.g. as
//    Pp<T1> pp;
// and let T2 be a class belonging to L(T1), and let t2 be an instance of
// T2.
//
// Then t2 can be put into the container pp by the
// statement
//
//       pp=t2; (or also pp.set(t2);) (**)
//
// Now pp carries just the information that was carried before (and is
// carried still) by t2. Warning: using to the right of = a ternary
// expression (... ? ... : ...) may destroy polymorphism. This is no real
// problem since the ternary can always replaced by if (...) ... else ...
// .
// If t2 gets changed or destroyed (e.g. as a local variable going out of
// scope) pp, will conserve the information and can return it in two
// ways.
// 1. Reconstruction as object: T2 t2_=pp(T2()) will create an object t2_
//    which is equivalent to t2.
// 2. Reconstruction by action: Let
//
//       virtual void T1::show(Media&)const;
//
// be a function which creates a detailed textual or graphical
// description of the calling instance by interacting with a class Media
```

```
// that provides the basis for that. Then 'Media m; pp().show(m);' will
// leave m in a state that carries the same record that we would get from
// 'Media m; t2.show(m)'. Technically, pp() is a quantity that is typed
// T1&; the object it refers to is of type T2 however.
// Let T3 be another class derived from T1, then pp, after pp.set(t3)
// behaves as T3 object. Thus pp behaves changing its type depending on
// initialization.
// Pp<> is closely related to the surrogate classes as discussed in
// Andrew Koenig: Ruminations on C++, AT&T 1997 Chapter 5. As stated
// there, the main benefit of such a 'polymorphic class' is that its
// instances can be put into a strictly typed container. This will be
// carried out in defining the extremely useful class Vp<>.
//
// Note that that in line (**) a constuction
// Vp<T1> pp=t2
// for a not yet existing pp does not work. The safe mode of operation is:
// 1. create the container; 2. fill the container.

template <class T>

class  Pp: public P<T>{ // polymorphic smart pointers
   typedef P<T> Base;
public:
    Pp(T* p=0):P<T>(p){}
      // carrying over the constructor from P. Notice that the
      // primary act of construction was to initialize p_. This can be by
      // T* p_=new Td(....) where Td is some derived class (or T itself)

    explicit Pp(T const& t):P<T>(static_cast<T*>(t.clone())){}
    // 2004-09-27: finally a pointer-less interface of Pp<> enabled!
    // The queer casting is needed in the general situation that
    // T0 and T1 are different (in the description above). This general
    // situation is probably not really important (I had it
    // unintentionally for T0=RigidObject and T1=ExtSys, but
    // simplified it to T0=T1=ExtSys without encountering problems)
    // Notice, however, that t.clone() involves call to the copy
    // constructor of T which might be expensive.

    Pp(const P<T>& x):P<T>(x){}
      // down-cast constructor

   T& operator()(void){ return *Base::p_;}
      // Pp<T> pp;
      // T t;
      // pp=t;
      // makes t.aVirtualFunction(...);
      // and
      // pp().aVirtualFunction(...);
      // calling the same function with the same data.
```

```
   T const& operator()(void)const{ return *Base::p_;}
      // not clear why the MS-compiler cannot infer this definition from
      // the base class.
      // Pp<T> pp;
      // T t=pp()
      // is the usual way to retreive the value of pp. If the value is
      // known to belong to a derived class Td, use the next function:
      //    Td t=pp(Td()).
      // If we don't want to retreive t but only want to call a
      // (potentially virtual) member function, one always proceeds as in
      // the following example:
      // Word w = pp().nameOf(); // see also comment to previous function

   template <class X>
      X operator()(X const& x)const
      // retreiving an element from the container as a copy of a
      // prescribed type.
      // The value of x has no influence, only the type matters.
      // T has to be a clone base of X.
      { X* px=static_cast<X*>(Base::p_->clone()); return X(*px);}

   template <class X>
   bool get(X& x)const
      // retreiving the element from the container as a reference
      // to the prescribed type. The value of x at input has no
      // influence, only the type matters. T has to be a clone base of X.
   { X* px=static_cast<X*>(Base::p_->clone()); x=X(*px); return true;}

   void set(T const& x){ cow(); Base::p_=static_cast<T*>(x.clone());}
      // loading an object of type T or of a derived type into Pp<T>
      // no longer needed since assignment works now

   Pp<T>& operator=(T const& x){ cow(); return *this=Pp<T>(x);}
      // 2004-09-27 together with a redefinition of
      // Vp<>::operator[]() this allows the intialization
      // of components of polymorphic arrays by assignment
      // so that the less idiomatic set(i,T $)-functions
      // should no longer be needed. The problems with the old
      // attempts to set values of components via assignment are
      // now understood (and eliminated)
   Word nameOf()const
   {
      return Word("Pp< "&CpmRoot::Name<T>()(T())&" >");
   }

protected:
   void cow(void);
      // 'copy on write'. The first '*this-changing' statement in the
      // body of a non-constant member function has to be cow();
};
```

```
template <class T>
void  Pp<T>::cow(void)
{
   if (Base::u_.makeOnly()){
      if (Base::p_!=0) Base::p_=static_cast<T*>(Base::p_->clone());
         // let Tb be the class closest to the root of the class
         // hierarchy where a virtual ... clone(void)const is defined,
         // then present clone is Tb* T::clone()const. Notice that
         // modern compilers sometimes accept T* T::clone()const.
         // Microsoft Visual C++ 5.0 is not of this kind,
         // so in defining clone() in a class, we have always to
         // remember the base. This is only a minor inconvenience.
      Base::u_.startNew_();
   }
}

//////////////////////////// class Po< > ////////////////////////////////

template <class T>
class  Po: public  Pp<T>{  // Adding order operations to Pp<T>
     // Allows to form, for instance, std::vector<Po<T> >
     // and thus allows to combine STL functionality with polymorphism
     // and banning of pointers
   typedef Pp<T> Base;
public:
   Po(T* p=0): Pp<T>(p){}
   Po(const Pp<T>& x): Pp<T>(x){}
   bool operator  <(const  Po& x)const{ return *Base::p_ <  *(x.p_);}
   bool operator ==(const  Po& x)const{ return *Base::p_ == *(x.p_);}
   bool operator  >(const  Po& x)const{ return *Base::p_ >  *(x.p_);}
   bool operator !=(const  Po& x)const{ return *Base::p_ != *(x.p_);}
   bool operator <=(const  Po& x)const{ return *Base::p_ <= *(x.p_);}
   bool operator >=(const  Po& x)const{ return *Base::p_ >= *(x.p_);}
};

////////////////////////////  class Vp< >  ////////////////////////////////
// This template behaves just as Pp<T> except that access is indexed
// which allows to store a number of instances and to retreive them:
// Let T,T1,T2,T3 as in the explanation of Pp<>. Then a typical usage is
//
// T1 t1(...);
// T2 t2(...);
//
// V<T3> t3(2);
// t3[1]=T3(...);
// t3[2]=T3(...);
//
// Vp<T1> vp(4);
// vp[1]=t1;
```

```
// vp[2]=t2;
// vp[3]=t3[1];
// vp[4]=t3[2];

// A detailed description (see Pp<>) of the the vp-components can be
// created like
// that:
//
// Z nItems=4;
// V<Media> m(nItems);
// (for Z i=1;i<=nItems;i++) vp(i).show(m[i]);
//
// Thus, despite their differing types, the t1, t2, t3[1], t3[2] get
// processed in a single loop; this capability is indispensible for
// polymorphism to be useful in non-trivial situations. The most
// expressive application is to use Vp<T1>-typed data members in classes:
//
// class ParticleMix{ // assume that Particle is clone base for both
//    // ParticleA and ParticleB
//    Vp<Particle> p;
// public:
//    ParticleMix(Z nA, Z nB):p(nA+nB)
//       // creates a mix of nA particles of type A and
//       // and nB particles of type B
//    {
//       Z i;
//       V<ParticleA> pA(nA); for (i=1;i<=nA;i++) p[i]=pA[i];
//       V<ParticleB> pB(nB); for (i=1;i<=nB;i++) p[i+nA]=pB[i];
//    }
//    void moveIndependently(R timeStep)
//    {
//       for (Z i=1;i<=p.dim();i++) p(i).moveIndependently(timeStep);
//             // applies the law of motion for A-particles if p(i) is
//             // a A-particle, and for B-particles correspondingly
//    }
// };
//
// Whereas in a 'stand alone'-control structure like the 'Media-loop'
// given above it would not be much more complicated to use a T1 pointer
// for implementing the loop, in the class example we gain more: Since
// Vp<Particle> is a value class, the compiler will properly and
// automatically define copy constructor, assignment operator, and
// destructor for us, wheras using pointers would leave this burden on
// us. Programmers who did this pointer implementation several times
// successfully might begin to love the procedure. I consider it wasted
// time and energy, and a lost opportunity to deal with complexity in a
// state-of-the-art manner.

template <class T>
class  Vp{ // polymorphic V template
```

```
   CpmArrays::V< Pp<T> > rep;

   explicit Vp(V< Pp<T> > const& vp):rep(vp){}

public:

    Vp(){}
      // notice that  both Pp<T> and V define a default constructor

   explicit Vp(Z n):rep(CpmArrays::V<Pp<T> >(n,Pp<T>())){}
      // No pre-setting of values occurs. If needed, this has to be
      // done by function void set(T const& x).

 //  Vp(Z n, Pp<T> const& p):rep(CpmArrays::V<Pp<T> >(n,p)){}
      // 'Making a series of copies of a single polymorphic object'.

 //  Pp<T> at(Z i)const{ return rep(i);}

   Pp<T>& operator[](Z i){ return rep[i];}
      // We use [] for access to the true components, which
      // are instances of the true value class Pp<T>.
      // Typical usage:
      // Vp<T> vp(2); T x=..., y=...; vp[1]=x; vp[2]=y;
   Pp<T> const& operator[](Z i)const{ return rep[i];}

   T& operator()(Z i){ return rep[i]();}
   T const& operator()(Z i)const{ return rep[i]();}
      // We use () for access to the T& that will be in
      // most cases the objects needed for further
      // processes, as for usage as function arguments.
      // Typical usage:
      // Vp<T> vp=...; Word w1=vp(1).nameOf(), w2=vp(2).nameOf();

   template <class X>
   X operator()(Z i, const X& x)const
      // retreiving the i-th component from the container as a copy of a
      // prescribed type. The value of x has no influence, only the type
      // matters. T has to be a clone base of X.
      { X* px=static_cast<X*>(rep[i]().clone()); return X(*px);}

   template <class X>
   bool get(Z i, X& x)const
      // retreiving the i-th component from the container as a referemnce
      // to the prescribed type. The value of x at input has no
      // influence, only the type matters. T has to be a clone base of X.
   {
      if (i<1 || i>rep.dim()) return false;
      X* px=static_cast<X*>(rep[i]().clone()); x=X(*px);
      return true;
```

```
   }

   void set(Z i, const T& x){ rep[i]=x;}
      // no longer essential since the defining statement
      // is natural enough not to need a wrapper

   void set(T const& x){ for (Z i=1;i<=rep.dim();i++) rep[i]=x;}

   Z dim()const{ return rep.dim();}
   Z b()const{ return rep.b();}
   Z e()const{ return rep.e();}
   bool valInd(Z i)const{ return rep.valInd(i);}

   Vp<T> rev()const
      //: reversed
   {
      return Vp<T>(rep.rev());
   }

   Word nameOf()const{
      Word nt=CpmRoot::Name<T>()(T());
      Word wi="Vp<";
      return wi&nt&">";
   }
};

} // namespace

#endif
```

26 cpms.h

```
//? cpms.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_S_H_
#define CPM_S_H_
/*

   Description: Defines a template class S<T> of sets
       the elements of which are of type T.
      Intented as a basis for a mathematics style definition of
           associative lists.

*/

#include <cpmvo.h>
#include <cpmtypes.h>

//////////////////////////// class S<> ////////////////////////////////////

namespace CpmArrays{

   using CpmFunctions::F;
   using CpmRoot::Word;

template <class T>
class S{ // sets of homotypic elements
   // Represents sets, all the elements of which are of type T.
   // It is assumed that T<T and T>T is defined.

protected:

   void create(Z n)
   // tool for constructors
```

```
   { n_=n; Vo<T> xx(n_); Vo<Z> vv(n_); x_=xx; v_=vv;}

   void create(const S& s)
   // tool for constructors
   { n_=s.n_; x_=s.x_; v_=s.v_;}

explicit   S(Z n){ create(n);}
      // Not yet a set since elements not initialized
      // for internal use only !!!!

   void setValidity(Z b);
      // sets a constant value b into the v-list
   void setValidity(T const&  t, Z b);
      // sets the validity of the element which equals t to b

// data all well typed so no copy constructor and assignment operator
// will be defined correctly by the compiler

   Z n_;
      // number of elements (i.e. cardinality)

   Vo<T> x_;   // ordered list of elements

   Vo<Z> v_;
      // auxiliary list indicating validity (valid:1 unvalid: 0 )

   typedef S<T> Type;

public:

   CPM_IO_V
      // CPM_ORDER comes later in this class declaration/definition

// constructors

   S(void){ create(0);}
      // constructor for the void set
   S(Word const& w, T const& t){w; create(1);x_[1]=t;v_[1]=1;}
      // constructor for the set {t}
      // e.g. S<Z> a("",5); seems to be acceptable
      // note that explicit S(T const& t){create(1);x_[1]=t;v[1]=1;}
      // would be ambiguous for T==Z

   S(Type const& s){ create(s);}
      // copy constructor

   void merge(V<Type> const& setList);
      // changes *this into the union of *this and the sets in the list

   explicit S(V<T> const& vec):n_(vec.dim()),x_(vec),v_(n_,1)
```

```
      { normalize();}
      // constructor from an array of T-values (V<T> sufficient, not
      // needing Vo<T>; T, however, has to support ordering (which is
      // clear if S<T> is under consideration). Clearly, multiple
      // appearing of components in vec does not prevent their
      // single appearence in the result (which, otherwise, would not be
      // a set)

    S(V<T> const& vec, V<bool> const& vecValid);
      // constructor from an array of T-values, together with a list of
      // validity values. vec[i] is considered valid iff
      // 1<=i<=vec.dim() && 1<=i<=vecValid.dim() && vecValid[i]==true
      // The valid values are put into a set.
      // Clearly, muliply appearing components will appear only
      // once in the result (which, otherwise, would not be a set)

// cardinality access

   Z car()const{ return n_;}
      //: cardinality
   Z dim()const{ return n_;} // for uniformity with other array types
      //: dimension
   Z size()const{ return n_;} // for uniformity with STL code
      //: size

   Z b()const{ return 1;}
      //. begin , as in V<>

   Z e()const{ return n_;}
      //. end , as in V<>

// set formation operations
   bool addAct(T const& t);
      // changes *this into the union of (*this) and {t}
      // and returns true if t was not already there
      // so that it had to be added 'actually'

   void add(T const& t){ addAct(t);}
      // changes *this into the union of (*this) and {t}
      // return value of addAct not used

   void add_(T const& t){ Type s(t); add(s);}
      // experimental

   void add(Type const& s);
      // changes *this into the union of (*this) and s

   void insert(T const&  t){ add(t);}
      // for uniformity with STL code
```

```
void push_back(T const&  t){ add(t);}
   // for uniformity with STL code where the container is vector<>

bool remove(T const&  t);
   // *this will be changed into *this\{t}
   // If this reduces the cardinality of *this, the return
   // value is 'true' and 'false' else

Type operator |(Type const&  s)const;
   // forming the union of *this and s (corresponds to the
   // 'or' operation for indicator functions)

Type operator &(Type const&  s)const;
   // forming the section of *this and s (corresponds to the
   // 'and' operation for indicator functions)

Type operator -(Type const&  s)const;
   // forming *this\s i.e. the set of all those elements of
   // *this which don't belong to s (relative complement)

Type operator ^(Type const&  s)const{ return (*this-s)|(s-*this);}
   // forming the symmetric difference of s1 and s2
   // (corresponds to 'exclusive or' for indicator functions)

T const&  operator[](Z i)const{ return x_[i];}
   // Means of iterating over all elements of the set
   // The specific indexing is gouverned by the < order.
   // If i is 'out of range' i.e. not 1<=i<=card(), this is
   // considered an error which is handled by Vo<T> (finally
   // by V<T>).

T& ref(Z i){ return x_[i];}
   // dangerous operation since changing x[i] may invalidate the
   // internal representation of *this. Needed for implementing
   // M<X,Y>.
   // Till 02-1-7 I had instead:  T operator[](Z i){ return x[i];}
   // This prevented Y& M<X,Y>::operator[](const X&) from working
   // properly. See also function normalize().

T const&  fir()const;
   //: first
   // first element of *this according to the ordering on
   // which the construction of Type is based.
   // causes error if *this is void

T const&  last()const;
   // last element of *this according to the ordering on
   // which the construction of Type is based.
   // causes error if *this is void
```

```
   void normalize(void);
      // shortens a list representation with multiple appearence of
      // a component such that all components are mutually different.
      // It also eliminates from the x-list all those elements x[i] for
      // which v[i]==0.
      // It thus achieves the status which for a well formed
      // list representation of a set is mandatory.
      // Not needed to be called normally (although this would not
      // cause problems). However if function ref() was called and
      // used to change elements it might be necessary to call
      // normalize to get an benign set again.

   T operator()(Z i)const{ return x_(i);}
      // Means of iterating over all elements of the set
      // The specific indexing is gouverned by the < order.
      // For i's 'out of range', T() is returned. This is not
      // considered an error.

// boolean operations

   bool isVoid(void)const{ return n_==0;}

   bool hasElm(T const& t)const;
      //: has element
      // returns true iff t is an element of *this

   bool ni(T const&  t)const{ return hasElm(t);}
      // 'in' reversed, from LATEX symbol \ni for the mirror image of
      // the epsilon-like 'is element'-symbol
      // Notice that the 'in' function would have as first argument a T
      // and as second argument a Type. This cannot be made a member
      // function. Introducing a non-member friend function would offend
      // the C+- coding style.

   Z locate(T const&  t)const;
      // returns 0 if t is not an element of *this
      // and its index if it is an element of *this

   bool isSubSetOf(Type const&  s)const;
      // returns true iff *this is a subset of s

   bool isSuperSetOf(Type const&  s)const;
      // returns true iff s is a subset of *this

// selection by functions

   void select(Z (*f)(T const& ));
      // turns *this into the subset consisting of those elements
      // x[i] of *this for which f(x[i])!=0
```

```
   void select(F<T,bool> const& f);
      // as previous function, only different type of the selection
      // function

   void select(V<bool> const& sel);
      // eliminates from *this all components x[i] for which sel[i]
      // is defined and statisfies sel[i]==false.
      // (see corresonding (non-mutating) function of V<T>)

   void eliminate(Z i);
      // eliminates from *this just component x[i] for 1<=i<=car()
      // Else, nothing is done. Does the same as remove((*this)[i])
      // but much faster , due to known position of the element to
      // remove.

   Type subSet(F<T,bool> const& f)const;
      // turns *this into the subset consisting of those elements
      // x[i] of *this for which f(x[i])==true

// relations to other classes:

   Vo<T> toV(void)const{ return x_;}
      // getting the representing vector as a new entity. It may be
      // useful to manipulate this vector by, for instance,
      // the V<T>::condense() function and then form a new set via the
      // constructor from vectors

// Infrastructure functions as declaration macros  known from including
// <cpmvo.h>
   CPM_ORDER
      // since T has to allow ordering, we implement the order
      // operations. This allows to form S< Type >.

   virtual S<T>* clone()const{ return new S(*this);}

// static bool effectivelyEqual(T const& t1, const T& t2){ return t1==t2;}
      // If no additional infrastructure in T is available, we can't do
      // better. Could be made non-static and virtual or template argument
      // to allow equivalence classes to be formed during set formation;
      // not persued here.

   Word toWord()const;

   virtual Word nameOf()const{
      Word wi="S< ";
      Word wt=CpmRoot::Name<T>()(T());
      return wi&wt&" >";
   }
};
```

```
// Implementation of S<T>

#define CPM_FOR_ALL for (i=1; i<=n_; i++)

template <class T>
void S<T>::setValidity(Z b)
{
   Z i;
   CPM_FOR_ALL v_[i]=b;
}

template <class T>
bool S<T>::addAct(T const&  t)
{
   if (n_==0){
      x_=Vo<T>(1,t);
      v_=Vo<Z>(1,1);
      n_=1;
      return true;
   }
   else if (n_==1){
      if (t==x_[1]){
         return false;
      }
      else{
         x_=x_.enqueue(t);
            // no call to locate() in this case
         v_=v_&true;
         n_++;
         return true;
      }
   }
   else{
      X2<Z,bool> ib=x_.findAsc(t);
      if (ib.c2()) return false; // t found in x_
      Z i=ib.c1()+1;
      x_=x_.ins(i,t);
      v_=v_&true;
      n_++;
      return true;
   }
}

template <class T>
void S<T>::add(const S<T>& s)
{
   Z m=s.car();
   if (m==0) return; // nothing to do
   if (n_==0){
      *this=s;
```

```
      return;
   }
   n_+=m;
   x_=x_&s.x_;
   v_=v_&s.v_;
   normalize(); // gives n the right value
}

template <class T>
void S<T>::normalize(void)
{
   Z mL=10;
   cpmmessage(mL,"S<T>::normalize(void) started");
   if (n_==0){
      cpmmessage(mL,"S<T>::normalize(void) done, first n_==0 exit");
      return; // nothing to do
   }

   Z i,j,nOld=n_;

// Eliminating the invalid entries

   n_=0;
   for (i=1;i<=nOld;i++){
      if (v_[i]==1) n_++;
   }
      // now n_ is the number of valid terms

   if (n_==0){
      Vo<T> xx;
      Vo<Z> vv;
      x_=xx;
      v_=vv;
      if (cpmverbose>=4)
         cpmmessage("S<T>::normalize(void) done, second n_==0 exit");
      return;
   }

   Vo<T> arr(n_);
   i=1;j=1;
   while(i<=n_ && j<=nOld ){
      while(j<=nOld && v_[j]==0){ j++; }
      if (j>nOld) break;
      arr[i++]=x_[j++];
   }

   Vo<Z> indx=arr.permutationForIncreasingOrder();

 // Eliminating  repetitions of terms
```

```
   Vo<Z> ind(n_);
   T t;
   i=0;j=0;
   while (i<n_ && j<n_){
      i++; j++;
      t=arr[indx[j]];
      // now we have to look whether j has to be increased.
      // while (j<n && effectivelyEqual(t,arr[indx[j+1]]) ){j++;}
      while (j<n_ && t==arr[indx[j+1]] ){j++;}
         // now j is the largest value for which arr[indx[j]]==t
         // since arr[indx[j+1]]!=t by the while condition
      ind[i]=j;
   }

   n_=i;
 //  cpmcerr<<"now n_ equals "<<n_;
   Vo<T> xx(n_);
   Vo<Z> vv(n_);

   x_=xx;
   v_=vv;

   CPM_FOR_ALL{
      x_[i]=arr[indx[ind[i]]];
      v_[i]=true;
   }
   n_=x_.dim();
   cpmmessage(mL,"S<T>::normalize(void) done");
}

template <class T>
S<T> S<T>::operator |(const S<T>& s)const
{
   Z nNew=n_+s.n_, i;
   S res(nNew);
   for (i=1;i<=n_;i++){
      (res.x_)[i]=x_[i];
      (res.v_)[i]=v_[i];
   }
   for (i=1;i<=s.n_;i++){
      (res.x_)[n_+i]=(s.x_)[i];
      (res.v_)[n_+i]=(s.v_)[i];
   }
   res.normalize();
   return res;
}

template <class T>
S<T> S<T>::operator &(const S<T>& s)const
{
```

```
   Z i;
   S<T> res;
   if (n_>s.n_){
      res=*this;
      res.setValidity(false);
      for (i=1;i<=s.n_;i++){
         res.setValidity((s.x_)[i],true);
      }
   }
   else{
      res=s;
      res.setValidity(false);
      for (i=1;i<=n_;i++){
         res.setValidity(x_[i],true);
      }
   }
   res.normalize();
   return res;
}

template <class T>
S<T> S<T>::operator -(const S<T>& s)const
{
   Z i;
   S<T> res=*this;
   for (i=1;i<=s.n_;i++){
      res.setValidity((s.x_)[i],false);
   }
   res.normalize();
   return res;
}

template <class T>
bool S<T>::hasElm(T const&  t)const
{
   Z r=x_.find(t);
   return (r!=0);
}

template <class T>
Z S<T>::locate(T const&  t)const
{
   return x_.find(t);
}

template <class T>
bool S<T>::remove(T const&  t)
{
   Z jf=x_.find(t);
   if (jf==0){
```

```
      return false;
   }
   else{
      v_[jf]=0;
      normalize();
      return true;
   }
}

template <class T>
void S<T>::setValidity(T const&  t, Z b)
{
   Z r=x_.find(t);
   if (r!=0) v_[r]=b;
}

template <class T>
bool S<T>::isSuperSetOf(const S<T>& s)const
{
   if (n_<s.n_) return false;
   S diff=s-(*this);
   return diff.isVoid();
}

template <class T>
bool S<T>::isSubSetOf(const S<T>& s)const
{
   if (n_>s.n_) return false;
   bool b=s.isSuperSetOf(*this);
   return !b;
}

template <class T>
void S<T>::select(Z (*f)(T const&))
{
   Z i;
   CPM_FOR_ALL v_[i]=f(x_[i]);
   normalize();
}

template <class T>
void S<T>::select(F<T,bool> const& f)
{
   Z i;
   CPM_FOR_ALL v_[i]=f(x_[i]);
   normalize();
}

template <class T >
void S<T>::select(const V<bool>& sel)
```

```
{
   Z i;
   Z ns=sel.dim();
   CPM_FOR_ALL v_[i]=(i<=ns ? sel[i] : true);
   normalize();
}

template <class T >
void S<T>::eliminate(Z i)
{
   if (i>1 && i<=n_){
      v_[i]=0;
      normalize();
   }
}

template <class T>
S<T> S<T>::subSet(F<T,bool> const& f)const
{
   S<T> res=*this;
   res.select(f);
   return res;
}

template <class T>
Z S<T>::com(S<T> const& s)const
{
   if (n_<s.n_){
      return 1;
   }
   else if (n_>s.n_){
      return -1;
   }
   else{ // now n==s.n
      for (Z i=1;i<=n_;i++){
         if ( x_[i]<(s.x_)[i]) return 1;
         if (x_[i]>(s.x_)[i] ) return -1;
      }
      return 0; // if all are equal
   }
}

template <class T>
S<T>::S(const V<T>& vec, const V<bool>& vecValid)
{
   n_=vec.dim();
   x_=vec;
   v_=Vo<Z>(n_);
   Z n2=CpmRootX::inf<Z>(n_,vecValid.dim());
   if (n2==0)
```

```
      cpmerror("S<T>::S(const V<T>&, const V<bool>&): dimension\
 mismatch");
   for (Z i=1;i<=n2;i++) v_[i]=(vecValid[i] ? 1 : 0);
   normalize();
}

template <class T>
Word S<T>::toWord()const
{
   Z nd=dim();
   Word res="{ ";
   for (Z i=1;i<nd;i++){
      Word wi=CpmRoot::toWord<T>((*this)[i]);
      wi&=", ";
      res&=wi;
   }
   res&=CpmRoot::toWord<T>((*this)[nd]);
   res&=" }";
   return res;
}

template <class T>
void S<T>::merge(const V< Type >& setList)
{
   Z d=setList.dim();
   for (Z i=1;i<=d;i++) add(setList[i]);
}

template <class T>
T const&  S<T>::fir()const
{
   if (n_==0) cpmerror("S<T>::fir(): set is void");
   return x_[1];
}

template <class T>
T const&  S<T>::last()const
{
   if (n_==0) cpmerror("S<T>::last(): set is void");
   return x_[n_];
}

template <class T>
bool S<T>::prnOn(ostream& str)const
{
   Z mL=1;
   static Word loc("S<>::prnOn(ostream&)");
   CPM_MA
   cpmwbt;
   bool bi;
```

```
   bi=CpmRoot::IO<Z>().o(n_,str);
   cpmassert(bi==true,loc);
   for (Z i=1;i<=n_;++i){
      if (CpmRoot::wrtTit){
         Word wi("// j=");
         wi&=cpm(i);
         bi=wi.prnOn(str);
            // the method here is to combine Words as Words and
            // write the result to stream (once!)
         cpmassert(bi==true,loc);
      }
      bi=CpmRoot::IO<T>().o(x_[i],str);
      cpmassert(bi==true,loc);
      if (bi==false){
         cpmdebug(x_[i]);
         CPM_MZ
         return false;
      }
   }
   cpmwet;
   CPM_MZ
   return true;
}

template <class T>
bool S<T>::scanFrom(istream& str)
{
   Z mL=2;
   static Word loc("S<>::scanFrom(istream&)");
   CPM_MA
   if (!CpmRoot::IO<Z>().i(n_,str)){
      cpmwarning(loc&": invalid stream after reading cardinality");
      CPM_MZ
      return false;
   }
   else{
      if(n_>dimMax || n_<=0)
         cpmwarning(loc&" n_= "&cpm(n_));
      S<T> res;
      T t;
      Z i;
      CPM_FOR_ALL{
         if (!CpmRoot::IO<T>().i(t,str)){
            cpmwarning(loc&": reading failed for i="&cpm(i)&
               " of "&cpm(n_));
            CPM_MZ
            return false;
         }
         res.add(t);
      }
```

```
      *this=res;
      CPM_MZ
      return true;
   }
}

#undef CPM_FOR_ALL

} // CpmArrays

#endif
```

27 cpmsr.h

```
//? cpmsr.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_SR_H_
#define CPM_SR_H_
/*
   Description: Defines a class template Sr<T> of sets
       the elements of which are of type T.

   History: Started 95-3-13
      95-3-15 transformed in a class derived from Obj
      97-9-5 transformed as to fit the CTL framework
      99-7-27 transformed as to fit the CPM framework
      02-5-24 implementation changed to STL, Test_set carried out
      2005-09-10 implementation purged from io_clone and r_clone
*/

#include <cpms.h>

namespace CpmArrays{

   using CpmRoot::Word;

template <class T>
class Sr: public S<T>{  // S with a rich interface
   // represents sets, all the elements of which are of type T.
   typedef Sr<T> Type;
   typedef S<T> Base;
public:
// constructors.

   Sr(void):S<T>(){}
         // constructor for the void set (n=0 added 95-11-24)
```

```
   Sr(const T& t):S<T>(t){}
         // constructor for the set {t}

   Sr(const S<T>& s):S<T>(s){}
      // down-cast constructor

   Sr<T> condense(void)const;
      // forms a new set obtained by shrinking to a single representative
      // element every equivalence class with respect to function equiv,
      // which itself is controlled by the static element acc

// Infrastructure functions

   virtual S<T>* clone()const{ return new Sr(*this);}

   CPM_IO
      // prnOn, scanFrom
   CPM_TEST_X
      // ran, test, hash, dis, abs, absSqr
   CPM_DESCRIPTORS
      // toWord, nameOf
      // The missing 3 from Root are: net, com, con

   static R acc;

protected:

   static bool equiv(const T& t1, const T& t2)
   { return CpmRoot::disT<T>(t1,t2)<acc;}

};

// test functions based on set theoretical identities

 template <class T>
   void idem1(Sr<T>& lhs, Sr<T>& rhs, const Sr<T>&s)
      { lhs=(s&s); rhs=s;}

   template <class T>
   void idem2(Sr<T>& lhs, Sr<T>& rhs, const Sr<T>& s)
      { lhs=(s|s); rhs=s;}

   template <class T>
   void comm1(Sr<T>& lhs, Sr<T>& rhs, const Sr<T>& s1, const Sr<T>& s2)
      { lhs= (s1&s2); rhs=(s2&s1);}

   template <class T>
   void comm2(Sr<T>& lhs, Sr<T>& rhs, const Sr<T>& s1, const Sr<T>& s2)
      {   lhs= (s1|s2); rhs=(s2|s1);}
```

```
   template <class T>
   void assoc1(Sr<T>& lhs, Sr<T>& rhs, const Sr<T>& s1, const Sr<T>& s2,
    const Sr<T>& s3)
      { lhs= ((s1&s2)&s3); rhs=(s1&(s2&s3));}

   template <class T>
   void assoc2(Sr<T>& lhs, Sr<T>& rhs, const Sr<T>& s1, const Sr<T>& s2,
    const Sr<T>& s3)
      { lhs= ((s1|s2)|s3); rhs=(s1|(s2|s3));}

   template <class T>
   void distr1(Sr<T>& lhs, Sr<T>& rhs, const Sr<T>& s1, const Sr<T>& s2,
    const Sr<T>& s3)
      { lhs= (s1|(s2&s3)); rhs=((s1|s2)&(s1|s3)) ;}

   template <class T>
   void distr2(Sr<T>& lhs, Sr<T>& rhs, const Sr<T>& s1, const Sr<T>& s2,
    const Sr<T>& s3)
      { lhs= (s1&(s2|s3)); rhs=((s1&s2)|(s1&s3)) ;}

   template <class T>
   void diffdistr1(Sr<T>& lhs, Sr<T>& rhs, const Sr<T>& s1, const Sr<T>&
    s2, const Sr<T>& s3)
      { lhs= (s1-(s2|s3)); rhs=((s1-s2)&(s1-s3)) ;}

   template <class T>
   void diffdistr2(Sr<T>& lhs, Sr<T>& rhs, const Sr<T>& s1, const Sr<T>&
    s2, const Sr<T>& s3)
      { lhs= (s1-(s2&s3)); rhs=((s1-s2)|(s1-s3)) ;}

// more technical tests

   template <class T>
   int addElements(const Sr<T>& s, const T& t);
      // adding and finding an element

   template <class T>
   R writeRead(const Sr<T>& s);
      // read write test (to and from file)

// Implementation of Sr<T>

#define CPM_FOR_ALL for (i=1;i<=Base::dim();i++)

template <class T>
R Sr<T>::acc=1e-3;

template <class T>
Sr<T> Sr<T>::condense(void)const
```

```
{
   Vo<T> xr=Base::toV();
   Vo<T> x1=(static_cast< V<T> >(xr).condense(equiv)).first();
   return S<T>(x1);
}

template <class T>
Word Sr<T>::nameOf(void)const
{
   Word a="Sr<";
   Word b=CpmRoot::Name<T>()(T());
   return a&b&">";
}

template <class T>
R Sr<T>::absSqr(void)const
//
{
   R s=0;
   Z i;
   CPM_FOR_ALL{
      s+=CpmRoot::AbsSqr<T>()((*this)[i]);
   }
   return s;
}

template <class T>
R Sr<T>::abs(void)const
//
{
   return cpmsqrt(absSqr());
}

/*
// looks natural but behaves not well in comparing two sets where the
// second is a noisy copy of the first (e.g. ofter writing to file with
// limited numerical accuracy and reading back. In this case the next
// version is superior. This illustrates the point that the notion of sets
// depends strongly on the notation of equality (and vice versa).

template <class T>
R Sr<T>::dis(const Sr<T>& s)const
{
   Sr<T> sa=(*this).condense();
   Sr<T> sb=s.condense();
   Sr<T> sc=sa^sb;

   R s1=sa.abs();
   R s2=sb.abs();
   R s3=sc.abs();
```

```
   return CpmRootX::distFunc(s1,s2,s3);
}
*/

template <class T>
R Sr<T>::dis(const Sr<T>& s)const
{
   if ( Base::n_ != s.car()) return 1;
   Vo<R> d(Base::n_);
   Z i;
   CPM_FOR_ALL d[i]=CpmRoot::Dis<T>()((*this)[i],s[i]);
   d.order();
   return d[Base::n_]; // max of d
}

template <class T>
Sr<T> Sr<T>::ran(Z j)const
{
   Z i;
   Sr<T> res=*this;
   Z j0=1000;
   Z j_=(j!=0 ? CpmRoot::ranT<Z>(j0,j) : 0);
   Z j1=j;
   bool b=true;
   V<T> xr(2*Base::n_);
   V<bool> vr(2*Base::n_);
   CPM_FOR_ALL{
      xr[i]=CpmRoot::ranT<T>(Base::x_[i],j1);
      vr[i]=CpmRoot::ranT<bool>(b,j1);
      j1+=j_;
      // random generator for T and B used
   }
   vr[1]=1; // that not all become false for small n
   return S<T>(xr,vr);
}

template <class T>
Sr<T> Sr<T>::test(Z tsz)const
{
   Z nt,i;
   nt=CpmRoot::testT<Z>(nt,tsz);
   T tTest;
   tTest=CpmRoot::testT<T>(tTest,tsz);
   V<T> xr(nt);
   for (i=1;i<=nt;i++){
      xr[i]=CpmRoot::ranT<T>(tTest,i);
   }
   return S<T>(xr);
}
```

```
template <class T>
Z Sr<T>::hash(void)const
{
   Z i,h=0;
   h+=Base::dim();
   CPM_FOR_ALL h+=CpmRoot::hashT<T>((*this)[i]);
   return h;
}

template <class T>
bool Sr<T>::prnOn(ostream& str)const
{
   return Base::prnOn(str);
}

template <class T>
bool Sr<T>::scanFrom(istream& str)
{
   return Base::scanFrom(str);
}

template <class T>
Word Sr<T>::toWord()const
{
   Word res="{ ";
   for (Z i=1;i<Base::n_;i++){
      Word wi=CpmRoot::toWord<T>((*this)[i]);
      wi&=", ";
      res&=wi;
   }
   res&=CpmRoot::toWord<T>((*this)[Base::n_]);
   res&=" }";
   return res;
}

template <class T>
int addElements(const Sr<T>& s, const T& t)
   // Tests whether an element added to a set is retrieved as an element
   {
      Sr<T> s1=s;
      s1.add(t);
      return s1.hasElem(t);
   }

template <class T>
R writeRead(const Sr<T>& s)
   // Tests whether a set can be read again when written to file
   {
      Word filename=s.nameOf()+".dat";
      Word name=s.nameOf();
```

```
      ofstream out(-filename);

      if (!out) cpmerror(name,"test: output file cannot be opened");
      out<<s;
      out.close();
      Sr<T> sr;
      R res;
      ifstream in(-filename);
      if (!in) cpmerror(name, "test: input file cannot be opened");
      in>>sr;
      in.close();
      res=s.dis(sr);
      return res;
    }
#undef CPM_FOR_ALL
} // CpmArrays

#endif
```

28 cpmsystem.h

```
//? cpmsystem.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_SYSTEM_H
#define CPM_SYSTEM_H
/*
   Purpose: Declaring basic facilities for a C+- based program to
   communicate with the user.
*/

#include <cpmword.h>

namespace CpmSystem{ // C+- system basic facilities

   using namespace CpmStd;

   using CpmRoot::Word;
   using CpmRoot::Z;
   using CpmRoot::R;

   class Message;
   void iniMPREAL();
   void updFromConfigFile();
   class Error;
   void error(Word const& w);
   void error(std::ostringstream const& ost);
   void error(Word const& w1, Word const& w2);

////////////////////// function glue ///////////////////////////////////

Word glue(Word const& path, Word const& file);
   //: glue
   // Connects a (partial)directory name and a file name
```

```
   // to give a full file name. 'glues' path and file together.
   // For extended==false, the result is simply
   // path&Word('/')&file. This is the correct behavior if file is a
   // normal filename which does not contain directory separators.
   // if extended==true we take into account that file may start
   // with ./ or ../. Then these dots will be eliminated.
   // Actually all all dots will be eliminated except the one
   // preceeding the file extension and

//////////////////////////// class Message ////////////////////////////////

class Message{ // provides basic output (unidirectional communication)
      // channels for a program. Two output files (cpmcerr and cpmdata
      // according to their define alias, see comments to data member
      // 'out' for these important entities) and a status bar featuring
      // 4 addressable segments are made available.
      // If we are running a console application, messages also go to the
      // console.
      // There is also limited communication in the other direction:
      // A file cpmconfig.ini, if it can be opened, will be used to
      // modify some static data members of the present class. This
      // excludes the data used to create the files cpmcerr and cpmdata.
      //
      // This class only has static data and methods, thus all instances
      // are identical. The information for initializing the static data
      // comes from the file cpmsystemdependencies.h.
      // A statement
      //    Cpmsystem::updFromConfigFile();
      // allows to change most static data by replacing them by values
      // read from the ini-file cpmconfig.ini. Since updFromConfigFile is a
      // friend of Message, it can do this replacement of private data of
      // class Message.

   friend void updFromConfigFile();
      // this allows to manipulate the private data of Message, by a
      // function which can be implemented only later when class
      // RecordHandler is available. Will be implemented in
      // cpmapplication.cpp. A project which does not call
      // function updFromConfigFile() needs not to have cpmapplication.cpp
      // in its file set.

   static bool streamInitialized;

   static Z vpw_, vph_, fsw_, fsh_;
      // viewport width, viewport height,
      // framestore width, framestore height,
   static Word sttmes1;
   static Word sttmes2;
   static Word sttmes3;
   static Word sttmes4;
```

```
static Z sttBarWid1;
   // status bar width 1
   // the status bar has 4 'panes' and the width of each is
   // characterized by a number; sttBarWidi is this number for the
   // i-th pane
static Z sttBarWid2;
static Z sttBarWid3;
static Z sttBarWid4;

static Word relPos1;
static Word relPos2;
static Word relPos3;
static Word relPos4;
   //: relative position
   // Relative directory positions, presently used only to locate
   // basic input file  cpmconfig.ini starting from the location
   // of the executable when starting from the command line or
   // from the working directory as seen by an IDE one is working
   // with. The search for cpmconfig.ini will be done for relPos1,
   // relPos2, relPos3, relPos4 in succession. Values for these
   // 4 static variables get set by reading cpmsystemdependencies
   // during compilation. An example is
   /*
      #define CPM_REL_POS1 "."
      #define CPM_REL_POS2 "./control"
      #define CPM_REL_POS3 ".."
      #define CPM_REL_POS4 "../control"
   */
   // The file cpmconfig.ini will be read at run-time and thus allows to
   // set quantities the change of which should not require
   // recompilation. Examples for this are size of the application
   // window, numerical precision in multiprecision computations,
   // input directory (e.g. ./webControl), and output directories
   // (e.g. ./r20200613a). This flexible handling of input directories
   // allows to feed several programs with identical input for
   // speed and accuracy benchmarking.
   // Notice that searching for cpmconfig.ini is automated only for
   // applications that have cpmapplication.cpp among its source
   // files.

static Word inpDir;
   //: input directory
   // directory root read from cpmconfig.ini that is available for
   // defining interaction of a program with the file system of the
   // machine

static Word outDir1;
   //: output directory 1
   // directory to which cpmcerr.txt and cpmdata.txt get written.
```

```
      // Can't be changed by reading from config.ini.

   static Word outDir2;
      //: output directory 2
      // directory to which output data of a program run will be written
      // can be set to a value different from outDir1 by reading from
      // config.ini

    static Word fontDir;
      //: font directory
      // font location  read from cpmconfig.ini that is available for
      // defining interaction of a program with the file system of the
      // machine. So a C+- user is completely free where to place
      // the C+- font file on his computer.

    static Word runId;
      //: run ID
      // Is created prior to the creation of the log files.

    static bool appRunId;
      //: append run id
      // If this is true, the run ID will be appended to the names of
      // auto-created image files made by class CpmGraphics::Frame
      // and documentation files made by
      // class CpmApplication::IniFileBasedApplication.
      // In the latter case there are provisions in place
      // to prescribe name appendices in ini-files, for instance
/*************************************************************************
         selection
         W runName=090616x
*************************************************************************/
      // This mechanism alone
      // does not make evident what version of the log files describes
      // their generation. So if appRunId is true, one appends runID
      // to this runName so that documentation files and log files which
      // belong togeher always have runID as a common name fragment.
      // One may change appRunId from cpmconfig.ini, where there is an
      // entry
/*************************************************************************
         append run id
         B val=false // or true
*************************************************************************/
    // calling void CpmSystem::updFromConfigFile() reads cpmconfig.ini and
    // may change appRunId.

   static Z runIdLength;

   static bool initialized(void){ return streamInitialized;}
      //
```

```
protected: // for usage by derived class Error

   static Z maxNumMes;
      // number of 'unenforced' messages to be written on out (helps to
      // avoid creating unintentionally huge text files and to slow
      // program run down

   static bool silent;
      // if this is true, the basic function communication runs idle
      // If it is set to true initially no log files will be created
      // and no possibility exists to swich it to false.
      // Thus setting CPM_NOLOG reliably says that there will no log
      // files be created during program run.

   static void communication(Word const&, Word const&, Word const&,
      bool, Z barSeg=1);

public:

// data
   static Z verbose;
      // controls the ammount of messages to be written to file in a way
      // that every message that writes under some value of
      // Message::verbose, will also be written under all higher values.

   static Z debug;
      // controls the behaviour of the cpmassert macro.
      // For debug==1 Message::warning will be executed
      // For debug==2 Message::error instead
      // In all other cases the asserted condition will not be tested

   static Z trigger;
      // Device for debugging, should be used only in code inserted for
      // debugging or temporary analysis. For instance one may
      // add
      //    cpmtrigger=1;
      // just before a dubious function call, and add to the block
      // of this function the statements:
      // if (cpmtrigger>0){
      //    cpmdebug(<argument 1>);
      //    cpmdebug(<argument 2>);
      //       ...
      //    cpmtrigger=0;}
      // This allows to make sure that the function one had in mind was
      // actually called, and it shows the arguments used for the call.
      // Who runs a debugger has probably never to do such tricks.

// functions
   static void ini(void);
      // one has not to call this function, but it can do no harm either,
```

```
   // helps experimentation in non-standard situations
static void fini(void){ out.close(); outData.close();}
   // only after being sure that the communication files have been
   // closed one can copy them for final documentation purposes
static Z pane1(){ return sttBarWid1;}
   //: pane 1
static Z pane2(){ return sttBarWid2;}
   //: pane 2
static Z pane3(){ return sttBarWid3;}
   //: pane 3
static Z pane4(){ return sttBarWid4;}
   //: pane 4

static Z w(){ return vpw_;}
   // tendative frame width in pixel
static Z h(){ return vph_;}
   // tendative frame height in pixel
   // Intended is a call
   // Z nx=Message::w(), ny=Message::h();
   // CpmGraphics::Viewport vp(nx,ny,title);
   // Making a statusbar within this field is the task of Viewport.

static Z ws(){ return fsw_;}
   // tendative frame store width in pixel
static Z hs(){ return fsh_;}
   // tendative frame store height in pixel

static ofstream out;
   // writing via '>>' to Message::out is the primarily provided
   // (unidirectional) communication facility of the system.
   // After program run, one will find the text file cpmcerr.txt in
   // the directory Message::outDir1 that contains all what has been
   // was written to stream Message::out.

   // If CPM_USE_MPI is defined and the number np of processes is > 1
   // the program is aware of its rank and this will be appended
   // to the filenames cpmcerr and cpmdata in a way that
   // alphabetic order is the natural order. E.g. for np=64
   // we have cpmcerr_01.txt,...cpmcerr_09.txt, cpmcerr_10.txt, ...,
   // cpmcerr_64.txt.
   // The next program run will overwrite cpmcerr(_rank).txt. In most
   // cases this is OK in order not to pollute the working directory.
   // Sometimes it is better to retain the cpmcerr, cpmdata files
   // e.g. for comparing them for two related runs. Of course,
   // we may rename the files after creation in an informative way
   // to enable this. There is, however, the function
   // CpmSystem::docRun(bool) which does this using Message::runId
   // as as a name appendix and copies the 'name-mangeled' files to
   // directory Message::outDir2. If cpmapplication.cpp is among the
   // source files of the project, entries in the file cpmconfig.ini
```

```
      // determine whether CpmSystem::docRun(bool) will be called
      // automatically and which preexisting directory will be used as
      // Message::outDir2.

   static ofstream outData;
      // writing via '>>' to Message::outData is a second
      // (unidirectional) communication facility of the system.
      // After program run, one will find the text file cpmdata.txt in
      // the execution directory that contains all what was written to
      // stream Message::outData and to the text file
      // text file cpmdata.txt in the directory Message::outDir1. Function
      // CpmSystem::docRun(bool) will also rename and copy this file if the
      // boolean argument is given the value true.

   static void onStatusBar(Word const& w, Z barSegment=1);
      // stores w in the Word memory for 'pane'=barSegment
      // If possible for the system, it displays the Words
      // sttmes1,...sttmes4 on the status bar of the application
      // window. The status bar has several segments, numbered
      // 1,2,... and the second argument determines the field
      // on which the message shall be placed. Presently
      // there are 4 such fields. In CONSOLE applications, only
      // w is displayed on the console.
      // For barSegment<=0 no action is done

   static void setMaxMes(Z mes=1000){maxNumMes=mes;}
      // setting the maximum number of (non-enforced) messages to be
      // written on out.

   // writing to the status bar

   static void progress(Word const& taskName, R frac, Z barSegment=1,
      Z field=3); // field =3 is usual
      // Displays task progress on the status bar.
      // for 0<=frac<=1 we get 000, 001, 002, ... 999, 1000
      // as a sufficient information on the degree of completion, only if
      // a digit changes there is change in the display. This is the
      // situation for field==3; for field=2 we go from
      // 00 to 100; etc. Only field=1,2,3,4,5 work this way; for any other
      // value one gets the same output as for field==4.

// writing to out

   static void report(Word const& text, Z val);
      // writes the number val on out if verbose>0

   static void report(Word const& text, R val);
      // writes the number val on out if verbose>0

// writing to both the status bar and out
```

```
   static void value(Word const& text, R val, Z barSegment=1);
      // writes the number val on the status bar and out if verbose>0

   static void values2(Word const& text, R val1, R val2, Z barSegment=1);
      // writes the numbers val1 and val2 on the status bar and out if
      // verbose>0

   static void values3(Word const& text, R val1, R val2, R val3,
      Z barSegment=1);

   static void values4(Word const& text, R val1, R val2, R val3,
      R val4, Z barSegment=1);

   static void value(Word const& text, Z val, Z barSegment=1);
         // writes the number val on the status bar and out if verbose>0

   Message(void);
      // initializes Message::out (if not done earlier)

   explicit Message(Word const& w);
      // initializes out (if not done earlier) and writes w to it

/* gets never called
   ~Message(){
      cout<<"Message destructor called"<<endl;
      out.flush();
      outData.flush();
   }
*/

// Message and warning only differ in the wording of the output.
// Two Word arguments allow to specify the class in the scope of which
// the message is created. For barSeg<=0, no action on the status bar.
// Reason: Sometimes it is important to write some data
// to cpmcerr which would overload and disturb the status display
// on the status bar. If barSeg<0, the message goes to the
// console if we have one, even if CPM_NOGRAPHICS is not defined.

   static void warning(Word const& text, bool fileAlways=false);
      // A simple warning message function writing on out.
      // If fileAlways is false, a limitation of messages to
      // be written to file will take place to avoid slowing down program
      // execution too much.

   static void warning(std::ostringstream const& ost,
      bool fileAlways=false)
   { Message::warning(Word(ost),fileAlways);}

   static void message(Word const& text, Z barSeg=1,
```

```
   bool fileAlways=false);

static void message(std::ostringstream const& ost, Z barSeg=1,
   bool fileAlways=false)
{ Message::message(Word(ost),barSeg,fileAlways);}

static void message(Word const& className, Word const& text,
   bool fileAlways=false);

static void warning(Word const& className, Word const& text,
   bool fileAlways=false);

static void message(Z mL, Word const& text, Z barSeg=1,
   bool fileAlways=false);
   // messaging takes place only if cpmverbose>=mL

static void urgentMessage(Word const& text)
   // a message that will always be given
{
   Z verbMem=verbose;
   verbose=1; message(text, 0, false); verbose=verbMem;
}

static void setSil(bool sil){ silent=sil;}
   // set silent

static bool getSil(){ return silent;}

static Word getRunId(){ return runId;}
   //: get run ID

static void setRunIdLength(Z const& l){ runIdLength=l;}
   //: set run ID length

static Z getRunIdLength(){ return runIdLength;}
   //: set run ID length

static void setRunId(Word const& id){ runId=id;}
   //: set run ID

static void augRunId(Word const& topic)
   //: augment run ID
{ if (topic.dim()>0) runId=topic&"_"&runId;}
   // allows making the ID more 'speaking'

static bool getAppRunId(){ return appRunId;}
   //: get append run ID

static void setAppRunId(bool app){ appRunId=app;}
   //: set append run ID
```

```
   static Word getInpDir(){ return inpDir;}
      //: get input directory

   static Word getOutDir1(){ return outDir1;}
      //: get output directory 1

   static Word getOutDir2(){ return outDir2;}
      //: get output directory 2

   static Word getFontDir(){ return fontDir;}
      //: get font directory

   static Word ext(Word const& w){ return glue(inpDir,w);}
      //: extend
      // Returns a file name which can be used to open a file.
      // For instance,
      //    IFileStream ifs(Message::ext("basics.txt"));
      //    Z sel; ifs>>sel;
      // reads an integer sel from file "basics.txt" which will be
      // searched in the directory determined by Message's conception
      // of a input directory. This allows different programs (with
      // different execution directories) access the same input data
      // for reliable performance comparisons. The different
      // programs need to mention in their respective files cpmconfig.ini
      // the same directory location as input directory.
      // It also allows to test a single program with various input
      // data sets (e.g. an experimental one and one already delivered
      // to a customer) by changing the content of the programs
      // cpmconfig.ini. This file may e.g. contain the lines
/*
      d:/cpm/pala
         // stable delivery data
      d:/cpm/palaX
         // experimental data
*/
   // Such entries between which one can switch by commentarization,
   // help to keep track of what one is doing.

   static Word extOut(Word const& w)
   {
      if (w.isVoid()) return Word(); else return glue(outDir2,w);
   }
      //: extend (for) output directory (2)
      // Notice that only outDir2 (not outDir1) can be set via config.ini
      // The void Word as filename as argument in some filing function
      // is always meant as the direction to ignore this function call.
      // So, it should not be transformed in a non-void Word by appending
      // a directory name to it. Changing the old and heavily used function
      // glue(...) to take this into account seems dangerous.
```

```
};

void error(Word const& w);
void error(std::ostringstream const& ost);
void error(Word const& w1, Word const& w2);
void lethalError(Word const& w);
   // does not write on cpmcerr.txt

//////////////////////////// class Error ////////////////////////////

class Error: public Message{ // used if C+- classes throw errors
    // No new data to Message, thus also a pure action class.
    // Since the action is determined by the constructor, the
    // functionality is new.
    // Inheritence allows to use the data and methods of Message to use
    // for implementation.

public:

explicit Error(Word const& messageText=Word("unspecified error"));
       // reporting a text in case of error, throws terminating exception

   Error(Word const& className, Word const& messageText);
       // reporting a class name and a text in case of error,
       // throws terminating exception
   friend void error(Word const& w);
      // terminates execution only if Message::debug>0
   friend void error(std::ostringstream const& ost);
   friend void error(Word const& w1, Word const& w2);

};

inline void error(std::ostringstream const& ost)
   { CpmSystem::error(Word(ost));}

struct Exception{ // simple debugging tool
// instead of saying cpmmessage(" reached point 3"); one simply says
// Exception(3);
   Z i_;
   Exception(Z i):i_(i){ Message::message("Exception "&Word::write(i_));}
};

///////////////// class Timer ////////////////////////////////////////

void wait(R t, Z pane=0);
      // delays program run for t seconds (especially to have time to
      // look to some screen information in absense of getchar() interupt
      // in Win32s programs. When the thread of control reaches a wait
      // statement it is not guaranteed that all graphical effects
```

```
      // created earlier on video memory can actually be seen on screen!
      // If the last argument is equal to one of the valid pane indexes
      // 1,2,3,4 and cpmverbose>0 the remaining time in seconds will be
      // shown there.
      // Of course, for t <= 0 nothing is being done.

R time(void);
      // returns the time at evaluation instant in seconds with respect
      // to the first call to time(). Relies on system function
      // gettimeofday.

class Timer{ // implements e.g. getSecondsLeft()
   R tBegin,tEnd;
   bool act;
public:
   Timer():tBegin(0),tEnd(0),act(false){}
   explicit Timer(R sp): tBegin(CpmSystem::time()),tEnd(tBegin+sp),
      act(true){}
      // sets tBegin to the time of function call
      // and tEnd to tBegin+sp. All times in seconds
   R getSecondsLeft()const;
      // returns the remaining span to tEnd expressed in seconds
      // For the default counter this function always returns 137*3600
   R getHoursLeft()const;
      // returns getSecondsLeft()/3600
   bool timeOut()const{ return getSecondsLeft()<=0.;}

};

////////////////////////// class IFileStream ////////////////////////////
//
// typical usage of file stream classes
// Word fIn("c:/d/cpm/myfile.txt");
// Word fOut("log.txt");
// IFileStream infi(fIn);
// OFileStream outfi(fOut);
// V<R> vr;
// vr.scanFrom(infi());
// vr.prnOn(outfi()); // notice '()' in using the streams
// In cpmfile.h there will be defined classes OFile and IFile which
// define assignment and copy.
// See also classes IFile and OFile in cpmfile.h for files which
// satisfy the strict value interface.

const bool binDef=true;
//const bool binDef=false;
   // true is proven, false for experimentation
   // default value for setting the argument 'binary'
   // for opening streams. Introduced 2004-04-29
```

```
class IFileStream{ // input file stream

   ifstream* fs_;
   const Word name_;
   typedef IFileStream Type;
   CPM_INVAR(IFileStream)

public:

   explicit IFileStream(Word const& fileName,
      bool safe=true, bool binary=binDef);
      // If the second argument is true one gets an error if the
      // file could not be opened, otherwise only a warning will
      // be issued.

   IFileStream():fs_(0){}

   virtual ~IFileStream(){ fs_->close(); delete fs_;}

   char readChar(){ char res('x'); *fs_>>res; return res;}
      //: read character
      // reading a char from stream

   string readWord(){ string res; *fs_>>res; return res;}
      //: read word
      // reads the next contiguous group of non-whitespace characters

   string readLine(){ string res; std::getline(*fs_,res); return res;}
      //: read line
      // reads the next line with all charcters also whitespace

   bool isVal()const
      //: is valid
      // returns stream status
   { if (!*fs_||fs_->bad()) return false; else return fs_->good(); }

   Word getName()const{ return name_;}
      //: get name

   ifstream& operator()(void){return *fs_;}
      // returns the object for direct access

   bool skipComments();
      // over-reads lines starting with '/', ';', '*', or '#'
};

////////////////////////// class OFileStream ////////////////////////////////

class OFileStream{ // output file stream
```

```
   ofstream* fs_;
   const Word name_;
   typedef OFileStream Type;
   CPM_INVAR(OFileStream)
public:

   explicit OFileStream(Word const& fileName,
      bool safe=true, bool binary=binDef);

   OFileStream():fs_(0){}

   virtual ~OFileStream(){ fs_->close();delete fs_;}

    bool isVal()const
       //: is valid
       // returns stream status
    { if (!*fs_||fs_->bad()) return false; else return fs_->good(); }

   Word getName()const{ return name_;}
      //: get name

   ofstream& operator()(void){return *fs_;}
      // returns the object for direct access,
      // this is all one needs to write to the object
};

void copyFile(Word const& fileNameIn, Word const& fileNameOut);
   //: copy file
   // lean file copy function, closely following BS3, p. 638 function main
   // The first argument needs to be the name of a file that can be opened.
   // The output file will be created (and written to) and will have the
   // name given by the second argument. Since as the result of a
   // successful function call the output file exists in the file system
   // of the machine, the file needs not to be returned.
   // Any malfunction exits from the program after having left a message
   // on cout. Since the raison d'etre of this function is to copy
   // the files cpmcerr.txt and cpmdata.txt they are not allowed to
   // write on these files.

void docRun(bool all=false);
   //: document run
   // creates a copy of cpmcerr.txt and, if all==true, also cpmdata.txt
   // with Message::getRunId() appended to the name in the directory
   // Message::outDir2. Notice that Message::outDir1 is the place where
   // the original cpmcerr.txt and cpmdata.txt are placed.

} // namespace

// An assert macro which is fully under the control of the C+- class
// system. The first argument should be a piece of C++ code which
```

```
// evaluates to bool. (Thus no C++-function can do the job since there
// are no variables that hold pieces of code)
// Example of usage:
//    cpmassert(x>0,"norm in function eval should be positive");
// notice that the final ';' is not needed

#define cpmassert(X,Y)\
if (CpmSystem::Message::debug==1){\
if (!(X)) CpmSystem::Message::warning(CpmRoot::Word(#X)&\
   " violated: "&Y);}\
else if (CpmSystem::Message::debug==2)\
{ if (!(X)) CpmSystem::error(CpmRoot::Word(#X)&" violated: "&Y);}

//////////////////// short names ////////////////////////////////////
#define cpmcerr         CpmSystem::Message::out
#define cpmdata         CpmSystem::Message::outData
#define cpmverbose      CpmSystem::Message::verbose
#define cpmtrigger      CpmSystem::Message::trigger
#define cpmsilent       CpmSystem::Message::getSil
#define cpmdbg          CpmSystem::Message::debug
   // the name cpmdebug, which would be natural here, was defined
   // earlier, see cpmtypes.h
#define cpmmessage      CpmSystem::Message::message
#define cpmurgent       CpmSystem::Message::urgentMessage
#define cpmwarning      CpmSystem::Message::warning
#define cpmprogress     CpmSystem::Message::progress
#define cpmreport       CpmSystem::Message::report
#define cpmstatus       CpmSystem::Message::onStatusBar
#define cpmrunid        CpmSystem::Message::getRunId()
#define cpmerror        CpmSystem::error
#define cpmwait         CpmSystem::wait
#define cpmvalue        CpmSystem::Message::value
#define cpmvalues2      CpmSystem::Message::values2
#define cpmvalues3      CpmSystem::Message::values3
#define cpmvalues4      CpmSystem::Message::values4
#define cpmtime         CpmSystem::time
#define cpmexc          CpmSystem::Exception

#endif // guard
```

29 cpmsystem.cpp

```
//? cpmsystem.cpp
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#include <sys/time.h> // <time.h> not sufficient
#include <iomanip>

#include <cpmgreg.h>
#include <cpmsystem.h>
#include <cpmsystemdependencies.h>

#if !defined(CPM_NOGRAPHICS)
   #include <cpmviewport.h>
#endif

#if defined(CPM_USE_BOOST)
   #include <boost/filesystem.hpp>
#endif // defined

//////////////////// class Message ////////////////////////////////////
using namespace CpmStd;
using CpmRoot::Z;
using CpmRoot::R;
using CpmRoot::Word;
using CpmRoot::toDouble;
using CpmArrays::V;

namespace{
   const char sep='/';
   const char dot='.';

   bool startDot(Word const& file)
   {
      if (file.dim()==0) return false;
```

```
      return file[1]==dot;
   }

   bool absPath(Word const& file)
   // if file start with /home/... it is an absolute path
   // and should not be augmented by a leading dot
   {
      if (file.dim()<6) return false;
      return file[1]==sep && file[2]=='h' &&
      file[3]=='o' && file[4]=='m' && file[5]=='e'
      && file[6]==sep;
   }

   bool startSep(Word const& file)
   {
      if (file.dim()==0) return false;
      return file[1]==sep && !absPath(file);
   }
}

void CpmSystem::iniMPREAL(void)
{
   Z mL=1;
   Word loc("CpmSystem::iniMPREAL()");
   CPM_MA
   #if defined(CPM_MP)&&defined(CPM_USE_MPREAL)
      CpmRoot::numPrc=CPM_MP;
      cpmprec(CpmRoot::numPrc);
      cpmmessage(loc&
            ": numerical precision set to "&cpm(CpmRoot::numPrc));
   #else
      cpmmessage(loc&": nothing to do");
   #endif
   CPM_MZ
}

Word CpmSystem::glue(Word const& path, Word const& file)
{
// Do not insert diagnostic messages, like cpmdebug(temp); into
// this function block, since the function is used in Message::ini()
// before the messaging file cpmcerr exists.
   if (file.dim()==0) return path;
   if (path.dim()==0) return file;
   Word res= startSep(file) ? path&file : path&Word(sep)&file;
   if (startSep(res)) res=Word(dot)&res;
   return res;
}

////////////////////// class Message ///////////////////////////////////
// initialization by code
```

```
Z CpmSystem::Message::verbose=2;
Z CpmSystem::Message::debug=2;
Z CpmSystem::Message::trigger=0;
Z CpmSystem::Message::maxNumMes=100000;
Word CpmSystem::Message::sttmes1="*";
Word CpmSystem::Message::sttmes2="**";
Word CpmSystem::Message::sttmes3="***";
Word CpmSystem::Message::sttmes4="****";
Word CpmSystem::Message::runId="";
ofstream CpmSystem::Message::out;
ofstream CpmSystem::Message::outData;
bool CpmSystem::Message::appRunId=true;

//initialization from cpmsystemdependencies.h
#if defined(CPM_RUN_ID_LENGTH)
   Z CpmSystem::Message::runIdLength=CPM_RUN_ID_LENGTH;
#else
   Z CpmSystem::Message::runIdLength=4;
#endif

#if defined(CPM_OUT_DIR) // for backward compatibility
   Word CpmSystem::Message::outDir1=CPM_OUT_DIR;
         // setting the directory location of cmcerr and cpmdata
   Word CpmSystem::Message::outDir2=CPM_OUT_DIR;
#else
   Word CpmSystem::Message::outDir1=CPM_REL_POS2; // old style
   Word CpmSystem::Message::outDir2=CPM_REL_POS2;
#endif
Word CpmSystem::Message::relPos1=CPM_REL_POS1;
Word CpmSystem::Message::relPos2=CPM_REL_POS2;
Word CpmSystem::Message::relPos3=CPM_REL_POS3;
Word CpmSystem::Message::relPos4=CPM_REL_POS4;
Word CpmSystem::Message::inpDir=CPM_INP_DIR;
Word CpmSystem::Message::fontDir=CPM_FONT_DIR;

#if defined(CPM_NOLOG)
   bool CpmSystem::Message::streamInitialized=true;
   bool CpmSystem::Message::silent=true;
#else
   bool CpmSystem::Message::streamInitialized=false;
   bool CpmSystem::Message::silent=false;
#endif
Z CpmSystem::Message::sttBarWid1=CPM_PANE1;
Z CpmSystem::Message::sttBarWid2=CPM_PANE2;
Z CpmSystem::Message::sttBarWid3=CPM_PANE3;
Z CpmSystem::Message::sttBarWid4=CPM_PANE4;

Z CpmSystem::Message::vph_=CPM_HEIGHT;
Z CpmSystem::Message::vpw_=CPM_WIDTH;
Z CpmSystem::Message::fsh_=CPM_HEIGHT;
```

```
Z CpmSystem::Message::fsw_=CPM_WIDTH;

// functions

void CpmSystem::Message::ini(void)
{
   if (initialized()) return;
   cout<<endl<<"Message::ini(void) started"<<endl;
   Z size=1;
   Z rank=1;
#if defined(CPM_USE_MPI)
   size=CpmMPI::Cpm_com.getSize();
   rank=CpmMPI::Cpm_com.getRank();
#endif
   Word loc("CpmSystem::Message::ini(void)");
   Word logFile("cpmcerr");
   Word datFile("cpmdata");
   CpmTime::Greg date;
   date.now();
   runId=date.getCode().tail(runIdLength);
      // runId will no longer be used to create an name appendix for
      // cpmcerr and cpmdata, since 'personalized' versions will generated
      // in the same directory as the other output by function
      // CpmSystem::docRun(bool). runId can be changed by getting a new
      // value for runIdLength from cpmconfig.ini. In the C+- framework
      // the run Id gets used only after the deployment of cpmconfig.ini
      // (if there is no such file, of course the presently generated
      // runId will be employed.
   if (size>1){
      // log files for various processes should be numbered
      Z w;
      if (size<=9) w=1;
      else if (size<=99) w=2;
      else if (size<=999) w=3;
      else w=0;
      Word wrd=cpmwrite(rank,w);
      logFile&=wrd;
      datFile&=wrd;
   }
   logFile&=".txt";
   datFile&=".txt";
// the folowing two lines were added 2014-01-19 after a new installation of
// Ubuntu as a conseqence of which the log files according to the unchanged
// logic were created always directly in the user directory
// (/home/mutze in my case).
   logFile=outDir1&"/"&logFile;
   datFile=outDir1&"/"&datFile;
   cout<<endl<<" trying to open "<<-logFile<<endl;
   out.open(-logFile, std::ios_base::out);
   bool streamInitialized1=true;
```

```
   if (!out) {
      streamInitialized1=false;
      cout<<endl<<"stream out is not valid"<<endl;
   }
   else cout<<endl<<"stream out is valid"<<endl;
   cout<<endl<<" trying to open "<<-datFile<<endl;
   outData.open(-datFile, std::ios_base::out);
   bool streamInitialized2=true;
   if (!outData){
      streamInitialized2=false;
      cout<<endl<<"stream outData is not valid"<<endl;
   }
   else cout<<endl<<"stream outData is valid"<<endl;
   if (streamInitialized1&&streamInitialized2){
      streamInitialized=true;
      date.writeFormatted(out);
      Word mes=loc&
         ": files "&logFile&" and "&datFile&" successfully created,\n";
      mes&="  rank="&cpm(rank)&", size="&cpm(size);
      cout<<endl<<-mes<<endl;
      cpmmessage(mes); // works
      // The files get created in the executable's working directory.
      // In MS Visual C++ the executable is automatically put (can be
      // changed in the 'project properties') into a Release or Debug
      // subdirectory of the project directory. cpmcerr.txt and
      // cpmdata.txt then get created just outside of these
      // subdirectories and thus directly in the project directory.
   }
   else{ // then we had trouble in creating the message streams
      if (streamInitialized1){ // since we really need only logFile, this
      // might be a situation not asking for stopping the program
         Word mes1=loc&": not able to create "&datFile;
         cout<<endl<<-mes1<<endl;
         cpmmessage(mes1);

      }
      else{
         Word mes2=loc&": not able to create "&logFile&
            ", stopping program";
         cout<<endl<<mes2<<endl;
         cpmmessage(mes2);
         throw;
            // something went fundamentally wrong, no message can be sent
      }
   }
   iniMPREAL(); // only active if R is implemented via mpreal.h
   cout<<endl<<"Message::ini(void) done"<<endl;
}

CpmSystem::Message::Message(void)
```

```
{
   ini();
}

CpmSystem::Message::Message(Word const& w)
{
   ini();
   if (verbose>0) out<<endl<<w<<endl;
   std::cout<<endl<<w<<endl;
  //  if (!out) cpmerror("CpmSystem::Message::Message(...): out==0");
}

#if !defined(CPM_NOGRAPHICS)
void CpmSystem::Message::onStatusBar(Word const& w, Z i)
{
   static Z firstrun_=1;
   if (i<=0) return; // was i<0 till 2006-06-30
   //in this case we can't write on the status bar before a graphical
   // window is available
   bool vii=CpmGraphics::Viewport::isInitialized();
   if (firstrun_ && vii){
      firstrun_=0;
      for (Z j=1; j<=4;j++) {
         CpmGraphics::Viewport::onStatusBar("pane "&cpm(j)&" active", j);
      }
      cpmwait(0.5);
   }
   if (vii) CpmGraphics::Viewport::onStatusBar(w, i);
}
#else
   void CpmSystem::Message::onStatusBar(Word const& w, Z i){}
#endif

void CpmSystem::Message::communication(Word const& classText,
   Word const& commType, Word const& commText, bool fileAlways,
   Z barSeg)
// common building block for messages and warnings, not for errors
// Dependence on preprocessor directives:
// CPM_USE_MPI : messaging done only in process 1 (0 in original style)
// _CONSOLE: everything written on cpmcerr.txt will also be written on
// cout
{
   static R tRetard=0.;
   if (silent) return; // this allows us to use C+- functions which
      // contain cpmmessage or cpmwarning statements in situations
      // in which creation of the log streams did not yet happen.
   static Z messageCounter=0;
   ini(); // esuring that 'out' can be used, no work load if already
   // initialized. For silent==true we never come here by message
   // calling and thus we are sure never to create the log files
```

```
   // as long as silent==true. If silent gets switched to false
   // (by code or whatever, next call to the present function will
   // call ini() and thus create the log files.
   Z eowMem=Word::getEndOfWord();
   Word::setEndOfWord(0);
      // then writing Words to a stream has the same effect as writing
      // string literals
   Z rank=1;
#if defined(CPM_USE_MPI)
   rank=CpmMPI::Cpm_com.getRank();
#endif
   if (verbose){ // only then, something has to be done
      messageCounter++;  // it is useful to count the messages
         // even if they are not restricted in number
      ostringstream ostStatus;
      ostStatus<<classText<<" "<<commType<<" "<<commText<<
      " # "<<messageCounter;
         // status bar should be written also if number of filed messages
         // is limited
      if (barSeg>0){
         onStatusBar(Word(ostStatus.str()),barSeg);
         if (messageCounter>110) cpmwait(tRetard,2);
      }
      ostringstream ostFile;
      if (fileAlways || messageCounter<maxNumMes){
         ostFile<<endl<<"counter = "<<messageCounter<<endl;
         ostFile<<classText<<" "<<commType<<" "<<commText<<endl;
         R t=time(); ostFile<<"  "<<cpmtod(t)<<" s after program start";
         out<<ostFile.str()<<endl<<std::flush;
#if defined(_CONSOLE)
         if (rank==1 && barSeg<0) std::cout<<ostFile.str()<<endl;
#endif
#if defined(CPM_NOGRAPHICS)&&defined(_CONSOLE)
         if (rank==1 && barSeg>0) std::cout<<ostFile.str()<<endl;
#endif
      }
      if (messageCounter==maxNumMes){
         ostFile<<endl<<
         "No more normal messages or warnings will be filed";
         out<<ostFile.str()<<endl;
#if defined(CPM_NOGRAPHICS)&&defined(_CONSOLE)
         if (rank==1) std::cout<<ostFile.str()<<endl;
#elif defined(_CONSOLE)
         if (rank==1 && barSeg<=0) std::cout<<ostFile.str()<<endl;
#endif
      }
   }
   Word::setEndOfWord(eowMem);
      // restoring the writing behavior of Word
}
```

```
void CpmSystem::Message::warning(Word const& errorText, bool fileAlways)
{
   static R waitTime=0.4;
   waitTime*=0.9;
      // if there are many warnings the run should not get slowed down
      // considerably
   communication("","Warning:",errorText, fileAlways);
   if (cpmverbose>=2) cpmwait(waitTime);
}

void CpmSystem::Message::message(Word const& text, Z barSeg,
   bool fileAlways)
{
   communication("","Message:",text,fileAlways,barSeg);
}

void CpmSystem::Message::message(Z mL, Word const& errorText,
   Z barSeg, bool fileAlways)
{
   if (cpmverbose>=mL)
   communication("","Message:",errorText, fileAlways,barSeg);
}

void CpmSystem::Message::warning(Word const& className,
   Word const& errorText, bool fileAlways)
{
   communication(className,"Warning:",errorText, fileAlways);
}

void CpmSystem::Message::message(Word const& className,
   Word const& errorText, bool fileAlways)
{
   communication(className,"Message:",errorText, fileAlways);
}

void CpmSystem::Message::progress(Word const& taskName, R frac, Z seg,
   Z field)
// new implementation (2005-04-17) asks for graphical action only if
// needed.
// This assumes that a specific segment on which progress display is
// active will not have to insert messages of different character.
// Actually the static memory elements should be associated with the
// segments directly and not only within a specific function such as
// 'progress'.
{
   if (silent) return;
   static Word mem1;
   static Word mem2;
   static Word mem3;
```

```
   static Word mem4;
   Word act1,act2,act3,act4;
   R fac2=1000;
  // R_ frac2=cpmtod(frac);
   if (field==3) ;
   else if (field==1) fac2=10;
   else if (field==2) fac2=100;
   else if (field==4) fac2=10000;
   else if (field==5) fac2=100000;
   else field=3;
   bool write=false;
   Word act=cpm(cpmtoz(frac*fac2),field);
   if (seg==1){
      if (act!=mem1){
         mem1=act;
         write=true;
      }
   }
   else if (seg==2){
      if (act!=mem2){
         mem2=act;
         write=true;
      }
   }
   else if (seg==3){
      if (act!=mem3){
         mem3=act;
         write=true;
      }
   }
   else if (seg==4){
      if (act!=mem4){
         mem4=act;
         write=true;
      }
   }
   if (write){ // graphical action only if needed
      write=false;
      Word mes=(verbose && seg==1) ? "Task: "&taskName&": Progress " :
         taskName&": ";
      mes&=act;
      if (verbose>2){
         message(mes,seg); // messages go always to the status bar
      }
      else{
         onStatusBar(mes,seg);
      }
   }
}
```

```
// shared code
// prec was 4 now experimental value 8


#define CPM_SC\
   static const Z prec=4;\
   if (verbose){\
      ostringstream ost;\
      ost.precision(prec);\
      ost<<text<<" = ";\
      ost.width(8);\
      ost<<valx;\
      if (verbose>2){\
         message(Word(ost.str()),seg);\
      }\
      else{\
         onStatusBar(Word(ost.str()),seg);\
      }\
   }

void CpmSystem::Message::value(Word const& text, R val, Z seg)
{
   double valx=cpmtod(val);
   CPM_SC
}

void CpmSystem::Message::value(Word const& text, Z val, Z seg)
{
   Z valx=val;
   CPM_SC
}

#undef CPM_SC

void CpmSystem::Message::values2(Word const& text, R v1, R v2, Z seg)
{
   static const Z prec=4;
   if (verbose){
      ostringstream ost;
      ost.precision(prec);
      ost<<text<<": ";
      ost.width(8);
      double v1x=cpmtod(v1);
      double v2x=cpmtod(v2);
      ost<<v1x<<", "<<v2x;
      if (verbose>2){
         message(Word(ost.str()),seg);
      }
      else{
         onStatusBar(Word(ost.str()),seg);
```

```
      }
   }
}

void CpmSystem::Message::values3(Word const& text, R v1, R v2,
   R v3, Z seg)
{
   static const Z prec=3;
   if (verbose){
      ostringstream ost;
      ost.precision(prec);
      ost<<text<<": ";
      ost.width(7);
      double v1x=cpmtod(v1);
      double v2x=cpmtod(v2);
      double v3x=cpmtod(v3);
      ost<<v1x<<", "<<v2x<<", "<<v3x;
      if (verbose>2){
         message(Word(ost.str()),seg);
      }
      else{
         onStatusBar(Word(ost.str()),seg);
      }
   }
}

void CpmSystem::Message::values4(Word const& text, R v1, R v2,
   R v3, R v4, Z seg)
{
   static const Z prec=3;
   if (verbose){
      ostringstream ost;
      ost.precision(prec);
      ost.width(7);
      double v1x=cpmtod(v1);
      double v2x=cpmtod(v2);
      double v3x=cpmtod(v3);
      double v4x=cpmtod(v4);
      ost<<text<<": "<<v1x<<", "<<v2x<<", "<<v3x<<", "<<v4x;
      if (verbose>2){
         message(Word(ost.str()),seg);
      }
      else{
         onStatusBar(Word(ost.str()),seg);
      }
   }
}

#define CPM_SC\
   if (verbose){\
```

```
      ostringstream ost;\
      ost<<text<<"="<<valx;\
      message(Word(ost.str()));\
   }

void CpmSystem::Message::report(Word const& text, Z val)
{
   Z valx=val;
   CPM_SC
}

void CpmSystem::Message::report(Word const& text, R val)
{
   double valx=cpmtod(val);
   CPM_SC
}

#undef CPM_SC
///////////////////////////// class Error ////////////////////////////////

// notice: base class constructor Message() called anyway

CpmSystem::Error::Error(Word const& messageText)
{
#if !defined(CPM_NOLOG)
   silent=false; // errors have to be reported
   verbose=1;
   communication("C+-",messageText,"CpmError",true,-1);
      // -1 lets this write also to the console
#elif defined(_CONSOLE)
   cout<<endl<<"C+- "<<messageText<<" CpmError"<<endl;
#endif
   throw;  // errors should terminate
}

CpmSystem::Error::Error(Word const& className, Word const& messageText)
{
#if !defined(CPM_NOLOG)
   silent=false;
   verbose=1;
   communication(className,messageText,"CpmError",true,-1);
      // -1 lets this write also to the console
#elif defined(_CONSOLE)
   cout<<endl<<"C+- "<<className<<" "<<messageText<<" CpmError"<<endl;
#endif
   throw;  // errors should terminate
}

////////////////// class Timer ///////////////////////////////////////
```

```
R CpmSystem::Timer::getSecondsLeft()const
   {
      static const R future=137*3600;
      if (!act) return future;
      R tAct=CpmSystem::time();
      return tEnd-tAct;
   }

R CpmSystem::Timer::getHoursLeft()const
{
   const R inv_h=1./3600.;
   return inv_h*getSecondsLeft();
}

////////////////////////// class-less functions //////////////////////////

void CpmSystem::lethalError(Word const& w)
{
   cout << "CpmSystem::letalError: " << -w << endl;
      // no service from Message needed. In particular,
      // cpmcerr.txt needs not to exist.
   exit(137);
}

void CpmSystem::error(Word const& w)
{
#if !defined(CPM_NOLOG)
   Message::ini();  // unclear whether needed
   Message::onStatusBar("Error",1);
   Message::onStatusBar(w,2);
#endif
   if (Message::debug>0) throw Error(w);
}

void CpmSystem::error(Word const& w1, Word const& w2)
{
#if !defined(CPM_NOLOG)
   Message::ini();
   Message::onStatusBar(w1,1);
   Message::onStatusBar(w2,2);
#endif
   throw Error(w1,w2);
}

R CpmSystem::time(void)
{
   static bool first=true;
   static timeval t1;
   if (first){
      gettimeofday(&t1, NULL);
```

```
      first=false;
   }
   timeval t2;
   gettimeofday(&t2, NULL);
   long sec  = t2.tv_sec  - t1.tv_sec;
   long usec = t2.tv_usec - t1.tv_usec;
   double t = sec + usec*1.e-6;
   return R(t);
}

void CpmSystem::wait(R t, Z pane)
{
   if (t<=0.) return;
   R tStart=time();
   R tEnd=tStart+t;
   R tLeft=tEnd-tStart;
   Z sLeft=cpmrnd(tLeft);
   while (tLeft>0)
   {
      tLeft=tEnd-time();
      if (pane>=1 && pane<=4){
         Z s0Left=cpmrnd(tLeft);
         if (s0Left<sLeft){
            sLeft=s0Left;
            cpmvalue(" tLeft",s0Left,pane);
               // visible only if cpmverbose>0
         }
      }
   }
   return;
}

////////////////// class IFileStream ////////////////////////
CpmSystem::IFileStream::IFileStream(Word const& fileName,
   bool safe, bool binary):name_(fileName)
{
   fs_=(binary? new ifstream(-fileName,std::ios_base::binary) :
      new ifstream(-fileName));
   if (!*fs_||fs_->bad()){  // till 2010-09-06 the test was fs==0
      Word mes="IFileStream(Word,bool,bool): unable to open "&fileName;
      if (safe) cpmerror(mes);
      else cpmwarning(mes);
   }
   else{
      cpmmessage("IFileStream(Word,bool,bool) did open "&fileName);
   }
}

bool CpmSystem::IFileStream::skipComments()
{
```

```
   return CpmRoot::eatComments(*fs_);
}

////////////////// class OFileStream ////////////////////////
CpmSystem::OFileStream::OFileStream(Word const& fileName,
   bool safe, bool binary):name_(fileName)
{
   fs_=(binary? new ofstream(-fileName,std::ios_base::binary) :
      new ofstream(-fileName));
   if (!*fs_||fs_->bad()){
      Word mes="OFileStream(Word,bool,bool): unable to open "&fileName;
      if (safe) cpmerror(mes);
      else cpmwarning(mes);
   }
   else{
      cpmmessage("OFileStream(Word,bool,bool) did open "&fileName);
   }
}

void CpmSystem::copyFile(Word const& fileNameIn, Word const& fileNameOut)
#if defined(CPM_USE_BOOST)
{
   cout<<"try to copy "<<-fileNameIn<<" to "<<-fileNameOut<<endl;
   boost::filesystem::copy(-fileNameIn,-fileNameOut);
   cout<<" copy done"<<endl;
}
#else
{
   std::ifstream from(-fileNameIn);
   if (!from)
      lethalError("CpmSystem::copyFile: cannot open input file "
      & fileNameIn);

   std::ofstream to(-fileNameOut);
   if (!to) lethalError("CpmSystem::copyFile: cannot open output file "
      & fileNameOut);

   char ch;
   while (from.get(ch)) to.put(ch);
   if (!from.eof()||!to) lethalError(
      "CpmSystem::copyFile: something strange happened");
   to.close();
   from.close();
}
#endif

void CpmSystem::docRun(bool all)
{
#if !defined(CPM_MPI)
   cout<<"CpmSystem::docRun(bool) started"<<endl;
```

```
   if (Message::getAppRunId()==false) return;
   Word appendix = cpmrunid ;
   Word pre1=Message::getOutDir1();
   Word pre2=Message::getOutDir2();
   Word s1=pre1&"/cpmcerr.txt";
   Word t1=pre2&"/cpmcerr_"&appendix&".txt";
   Word s2=pre1&"/cpmdata.txt";
   Word t2=pre2&"/cpmdata_"&appendix&".txt";
   CpmSystem::copyFile(s1,t1);
   if (all){
      CpmSystem::copyFile(s2,t2);
   }
   cout<<"CpmSystem::docRun(bool) done"<<endl;
#endif
}
```

30 cpmsystemdependencies.h

```
//? cpmsystemdependencies.h
//? C+- by Ulrich Mutze. Status of work 2016-06-15.
//? Copyright (c) 2009 Ulrich Mutze, www.ulrichmutze.de
//? All rights reserved

#ifndef CPM_SYSTEMDEPENDENCIES_H_
#define CPM_SYSTEMDEPENDENCIES_H_
/*************************************************************************
   cpmsystemdependencies.h
   Description: By this file one can place defines in implementation
    files in cases that the proper implementation is dependent on the
    computer system, e.g. on the access to a display or a file system.
    These defines may also modify the way how we use such facilities;
    e.g. using or not using the graphical display. Or, creating or
    not creating log files. Especially such changes of usage should
    not entail too much recompilation. So the present file should
    never be included in header files. Presently it is only included
    in six files: cpmnumbers.cpp, cpmsystem.cpp, cpmapplication.cpp
    cpmgraph.cpp, cpmimg24.cpp, cpminifilebasapp.cpp.

    Notice that changes in file cpmdefinitions.h trigger recompilation
    of virtually all C+- translation units.

    See also cpmconfig.ini, where most data can be overwritten.

*************************************************************************/

#define CPM_REL_POS1 "."
#define CPM_REL_POS2 "./control"
#define CPM_REL_POS3 ".."
#define CPM_REL_POS4 "../control"

#define CPM_INP_DIR "./control"
#define CPM_OUT_DIR "."

#define CPM_FONT_DIR "./fonts"

#define CPM_HEIGHT 832
#define CPM_WIDTH  1432

#define CPM_PANE1 350
#define CPM_PANE2 250
#define CPM_PANE3 250
#define CPM_PANE4 100

#define CPM_STEREOLEFT_R 255
#define CPM_STEREOLEFT_G 0
```

```
#define CPM_STEREOLEFT_B 0

#define CPM_STEREORIGHT_R 0
#define CPM_STEREORIGHT_G 255
#define CPM_STEREORIGHT_B 255

#define CPM_RUN_ID_LENGTH 4

#define CPM_WRITE_PRECISION 8

#define CPM_WRITE_TITLE

//#define CPM_CERR_EXTEND

#define CPM_USE_BOOST

#define CPM_USE_EIGEN
   // using the Eigen library for fast matrix and vector algorithms
   // for real and for complex quantities. The algorithms for
   // eigenvalues/eigenvectors and SVD do not cooperate with multiple
   // precision. Defining CPM_MP automatically undefines CPM_USE_EIGEN.
   // Using Eigen requires no linking with a lib. Only include files
   // are required.

#if defined(CPM_USE_EIGEN)
   #include <Eigen/Core>
   #include <Eigen/Dense>
#endif

//#define CPM_NOLOG

//#define CPM_NOGRAPHICS

#if !defined(CPM_NOGRAPHICS)
   #include <GL/glut.h>
#endif

#endif
```

31 cpmtypes.h

```
//? cpmtypes.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_TYPES_H_
#define CPM_TYPES_H_
/*

   Description: Declares basic infrastructure functions, macros,
      and ---most importantly--- the class versions B, Z1, and R1
      of bool, Z, and R.
*/
#include <cpmsystem.h> // includes cpmwords.h and
   // thus also cpmnumbers.h
#include <cpmmacros.h>
#include <cpmx.h>

#include <cmath>

namespace CpmRootX{ // CpmRoot extended
      // A statement
      //    using namespace CpmRoot;
      // makes life much easier when working with C+-. It is therefore
      // desirable that this does not inject to many names in the users
      // scope. Thus we separate the 'true root' from an 'extended root'
      // that hosts also names such as 'order', 'inf', and 'sup' which
      // are more likely to interfere with a user's names.

   using namespace CpmStd;

   using CpmRoot::Z;
   using CpmRoot::N;
   using CpmRoot::R;
   using CpmRoot::L;
```

```
   using CpmRoot::Word;

   const R hugeNumber=1e64;
   const R tinyNumber=R(1.)/hugeNumber;

#define cpmcheck(X)\
 if (!CpmRoot::isVal(X)){\
   ostringstream ost;\
   ost<<" bad value: "<<#X "="<< X;\
   cpmmessage(Word(ost.str()));}
   // checks whether a quantity of type Z,N,R,Rh is valid
   // (and stops the program) and gives a message

#define cpmdebug(X) cpmcerr<<endl<<"C+- debug: "<<#X "="<< X <<endl
   // useful for placing temporary debugging statements into one's code.
   // From Bruce Eckel: Thinking in C++, Prentice Hall 1995, p. 325
   // typical usage:
   //
   // R_Matrix y=....;
   // cpmdebug(y);          // ';' needed !
   // The part 'debug:' helps to find the debug info in the potentially
   // very large file cpmcerr.txt

#define cpmnote(X) cpmdata<<endl<<"C+- note: "<<#X "="<< X <<endl
   // Works as cpmdebug, but but writes on cpmdata. More suitable
   // for data which are of temporary interest, so that it would
   // not pay to provide a more elaborate documentation for them.
   // Notice that cpmdata is more easily surveyed than cpmcerr.

#define cpmis(X) std::string(#X)
   // 'is' stands for 'identifier to string'.
   // This can't be written as a C or C++ function. Such a function
   // always looks on the value of a variable, wheras the present
   // construct looks on the name of the variable and returns it as a
   // quantity of type string.
   // Consider:
   //    R x=3.14;
   //    cout<<cpmis(x);
   // This will write 'x' and not '3.14'

#define cpmiw(X) Word(#X)
   // much alike previous macro

typedef void (*fpWordToVoid)(const Word&);
   // function pointer for function Word to void as a type

typedef void (*fpVoidToVoid)(void);
   // function pointer for function void to void as a type

Z sig(Z i);
```

```
R sig(R i);
   // added 03-11-05, 0 for i==0, -1 for i<0, +1 for i>0

R krn(Z i, Z j);
   // Kronecker; defined as 1. for i==j, 0. else
   // added 2004-08-26

Z parity(Z k);
/*{
   if (k%2) return -1; else return 1;
}*/

Z fac(Z n);
//: factorial
// fac(n) = 1x2x3x...x(n-1)xn , n>0 (no other values allowed here!)
// Most basic implementation.

Z pow(Z const& x, Z p);

bool isPowOfTwo(Z const& x);

Z nextPowOfTwo(Z const& x);
// We return the smallest y \in P2 such that *this <= y
// Here P2 := {0,2,4,8,16,32,....}

Z nextLog2(Z const& x);
// We return the conventional dual logarithm of
// nextPowOfTwo() ( which then is always \in N)

// Auxiliar functions.

void copyTextFile(ifstream& from, ofstream& to);
   // here, the streams from and to have to exist already

void copyTextFile(const Word& nameOfSource, const Word& nameOfCopy );
   // a file of the name nameOfSource has to exist and the copy with the
   // name indicated by the second argument will exist after the function
   // body has been executed

string toString(ifstream& from);
   // copies the content of a text file into a standard library string
   // and thus opens up rich manipulation capabilities mainly by writing
   // these to stringstreams which finally can easily be filed back. In
   // this manner e.g. concatenation of text files becomes nearly trivial

// addition 97-6-5, notice that names min and max clash with
// macro names of Windows. Thus we choose presumably save naming.
// Version with const T& arguments is not always working. Sometimes one
// gets unability to resolve function overload
```

```
//////////////////////// sup ////////////////////////////////////////
   template <class T>
   T sup(T t1, T t2)
      // sup=supremum=maximum
   {
      if (t1<t2) return t2; else return t1;
   }

   template <class T>
   T sup(T t1, T t2, T t3)
   {
      T temp;
      if (t1<t2) temp=t2; else temp=t1;
      if (temp<t3) return t3; else return temp;
   }

   template <class T>
   T sup(T t1, T t2, T t3, T t4)
   {
      T temp1=sup<T>(t1,t2);
      T temp2=sup<T>(t3,t4);
      return sup<T>(temp1,temp2);
   }

//////////////////////// inf ////////////////////////////////////////

   template <class T>
   T inf(T t1, T t2)
      // inf=infimum=minimum
   {
      if (t1<t2) return t1; else return t2;
   }

   template <class T>
   T inf(T t1, T t2, T t3)
      // inf=infimum=minimum
   {
      T temp;
      if (t1<t2) temp=t1; else temp=t2;
      if (temp<t3) return temp; else return t3;
   }

   template <class T>
   T inf(T t1, T t2, T t3, T t4)
   {
      T temp1=inf<T>(t1,t2);
      T temp2=inf<T>(t3,t4);
      return inf<T>(temp1,temp2);
   }
```

```
   ////////////////////////// swap /////////////////////////////////////////

   template <class T>
   void swap(T& t1, T& t2)
   // After function call t1 has the value that t2 had before
   // and t2 has the value that t1 had before.
   // If T is a class (not built-in type) we assume that T defines
   // operator = and a copy constructor.
   {
      T temp=t2;
      t2=t1;
      t1=temp;
   }

////////////////////////// order /////////////////////////////////////////

   template <class T>
   void order(T& t1, T& t2)
      // after function call we have t1<=t2
      // If T is a class (not built-in type) we assume that T defines
      // a '<=' operation such that always at least one of t1<=t2 ,
      // t2<=t1 is valid. Also operator = and a copy constructor have to
      // be defined.
   {

      if (t1<=t2){
         return;
      }
      else{
         T temp=t2;
         t2=t1;
         t1=temp;
      }
   }

// Class equivalents of R , Z, bool.
// These can conveniently be used for class data members
// since they get initialized properly.
using CpmArrays::X2;

#if !defined(CPM_MP) && !defined(CPM_MPREAL)
// Once I introduced an implementalion of floatingpoint reals and of
// integers as classes. This was the only way to provide for real and for
// integer data members a automatic initialization by means of the
// defined default constructor. With C++11 one can data members
// define as e.g. R x{0.}; for what was R1 x; before.
// We can now avoid the ugly mixture of R and R1 (and Z and Z1).
// Particularly inconvenient was the maintenance of R1, Z1 for the
// the multiprecision options. So the intent is to avoid any use of
// R1 and Z1 in regular C+- code. Nevertheless I conserve the definition
```

```
// of R1 and Z1 to enable experimentation, especially with the discrete
// lattice underlying the floating point numbers.

////////////////////////////// class R1 ////////////////////////////

class R1{  // real numbers as a class
   // since there is no need for 1-tupels, we
   // define this a class endowed with infratructure similar than R2,
   // R3, ... but with arithmetics gained from the non-class type R via
   // automatic conversion (in one direction only to avoid ambiguities)
   // Most functions use R-arguments and not R1-ones since R1's have to
   // be converted to R anyway in most situations.
public:
   static const R1 min;
   static const R1 max;
   static const R1 eps0;
   static const R1 eps1;
   static const R1 logmax;
   static const R1 log10max;
   static const R1 pi;
   static const R1 piInv;

   R x1;
   typedef R1 Type;
// constructors
   R1():x1(0){}
   explicit R1(R a):x1(a){} // no automatic conversion wanted
   operator R()const{ return x1;}
      // auto-conversion to R is wanted here
   R operator()(void)const{ return x1;}
      // natural output of the intrinsic value
   R1& operator=(R a){ x1=a; return *this;}
   R1& operator+=(R a){ x1+=a; return *this;}
   R1& operator-=(R a){ x1-=a; return *this;}
   R1& operator*=(R a){ x1*=a; return *this;}
   R1& operator/=(R a){ x1*=R(R1(a).inv()); return *this;}
   R1 operator - ()const{ return R1(-x1);}
 //  R operator*(R const& a)const{ return x1*a;}
 //  friend R operator*(R const& a, R1 const& b){ return a*b.x1;}

   R abs()const{ if (x1<0) return -x1; return x1;}
   R dis(const R1&)const;
   X2<R,R1> polDec()const;
      // polar decomposition: First component (res.first if res is the
      // return value) is the absolute value r and the second component
      // is sign(x1)
   CPM_IO
 //  CPM_ORDER
   friend bool operator == (R1 r1, R1 r2){ return r1.x1 == r2.x1;}
   friend bool operator != (R1 r1, R1 r2){ return r1.x1 != r2.x1;}
```

```
   friend bool operator >= (R1 r1, R1 r2){ return r1.x1 >= r2.x1;}
   friend bool operator <= (R1 r1, R1 r2){ return r1.x1 <= r2.x1;}
   friend bool operator > (R1 r1, R1 r2){ return r1.x1 > r2.x1;}
   friend bool operator < (R1 r1, R1 r2){ return r1.x1 < r2.x1;}
   Z com(R1 r)const; // Compare function as tool, since R itself
      // does not work with com.
   Z dim()const{ return 1;}
   R1 inf(R1 a)const{ return x1<=a.x1 ? R1(x1) : R1(a.x1);}
   R1 sup(R1 a)const{ return x1>=a.x1 ? R1(x1) : R1(a.x1);}
   R& operator[](Z i){ i; return x1;}  // defining components in analogy
      // to what is done for R2, R3, R4
   const R& operator[](Z i)const{ i; return x1;}
   void set(R t){ x1=t;}
   void makePos_(){ if (x1<0) x1=-x1;}
      //: make positive
   Word nameOf()const{ return "R1";}
   Word toWord()const{ return Word::write(x1);}
   bool isVal()const{ return CpmRoot::isVal(x1);}
// functions referring to R and R1 as a discrete sets ('lattice')
   R1& next_();
      // next (in lattice)
   R1& prv_();
      // previous (in lattice)
   Z dz(R a)const;
      // distance on the lattice of represented numbers
      // counts the points exactly and thus is very slow for
      // typical pairs of numbers. If two numbers arrose from
      // getting the same 'mathematical number' by different
      // numerical expressions, the values should be close enough
      // so that dz should be small (approximately the number of
      // operations in the algorithm ???). Function dr is a much
      // faster approximate representation of this discrete distance.
   R eps(Z dir=1)const;
      // distance to the next lattice point in positive direction
      // for dir>=0 and in negative direction for dir<0.
      // In this latter case also the result is returned as negative
      // Thus x+x.eps(1) is the next lattice neighbor of x in
      // positive direction and x+x.eps(-1) is that in negative
      // direction
   R dr(R a)const; // const?
      // approximate number of lattice intervals between a and *this
      // (always >=0)
   Z run(R xFinal, ostream& str, Z writePeriod=1000)const; //const?
      // runs from *this to xFinal in steps over all represented
      // numbers and prints out ( to stream o) each n'th of
      // them with n given by writePeriod
      // Returns the number of printed numbers.
// safe functions
   R1 inv()const;
      // inverse, returns R1::max.x1 for x1==0
```

```
   R exp()const;
      // Returns R1::max.x1 for all x1 that would normally give infinity.
      //
      // Example of usage: instead of exp(x) write R1(x).exp() if it is
      // important to get a valid, non-exceptionl, result
   R log()const;
   R log10()const;
      // these four functions are defined on whole R1.
      // logs are log of absolute value, a combination which
      // is common in mathematics.

   R sign()const{ return (x1> 0. ? 1. : (x1==0. ? 0. : -1.));}
         //: sign (English),
         // akin to Latin signum

   R sign(R b)const{ return x1>0 ? x1*R1(b).sign() : -x1*R1(b).sign();}
        // 'takes the modulus from *this and the sign from b'
};

////////////////////////////// class Z1 ////////////////////////////////

class Z1{ // integer numbers as a class
   // tuples of 1 elements, for formal
   // symmetry with others
public:
   Z x1;
   static const Z1 min;
   static const Z1 max;
   typedef Z1 Type;
// constructors
   Z1():x1(0){}

   explicit Z1(long int a):x1(a){} // no automatic conversion wanted
   explicit Z1(int a):x1(a){} // no automatic conversion wanted
   explicit Z1(R r):x1(cpmrnd(r)){} // no automatic conversion wanted
   operator Z()const{ return x1;}
   R toR()const{ return R(x1);}
   Z1& operator=(Z a){ x1=a; return *this;}
  // Z1& operator=(bool b){ if (b) x1=1; else x1=0; return *this;}
   Z1& operator+=(Z a){ x1+=a; return *this;}
   Z1& operator-=(Z a){ x1-=a; return *this;}
   Z1& operator*=(Z a){ x1*=a; return *this;}

   Z1 pow(Z p)const;
      // Z1(x1^p), cpmassert p>=0

   Z abs()const{ if (x1<0) return -x1; return x1;}
   R dis(const Z1&)const;
   X2<Z,Z1> polDec()const;
      // polar decomposition: First component (res.first if res is the
```

```
      // return value) is the absolute value r and the second component
      // is sign(x1)
   CPM_IO
   CPM_ORDER
   Z dim()const{ return 1;}
   Z1 inf(Z1 a)const{ return x1<=a.x1 ? Z1(x1) : Z1(a.x1);}
   Z1 sup(Z1 a)const{ return x1>=a.x1 ? Z1(x1) : Z1(a.x1);}
   Z& operator[](Z i){ i; return x1;}  // defining components in analogy
   // to what is done for Z2
   const Z& operator[](Z i)const{i; return x1;}
   Word nameOf()const{ return "Z1";}
   Word toWord()const{ return Word::write(x1);}
   bool isVal()const{ return CpmRoot::isVal(x1);}
   Z parity()const{ return CpmRootX::parity(x1);}

   bool isPowOfTwo()const;
      // Let N:={1,2,3,4,...} (no zero!) and
      // P2:={ 2,4,8,16,...}
      // Then we return true iff *this belongs to P2

   Z1 nextPowOfTwo()const;
      // We return the smallest y \in P2 such that *this <= y

   Z1 nextLog2()const;
      // We return the conventional dual logarithm of
      // nextPowOfTwo() ( which then is always \in N)
};
#endif

class B{ // boolean values as a class
public:
   bool x1;
public:
   typedef B Type;
// constructors
   B():x1(false){}
      // default is false !!!
      // corresponds to default 0 in Z1,R1,...

   explicit B(Z a):x1(a!=0){} // no automatic conversion wanted
   operator bool()const{ return x1;}

   B& operator=(bool a){ x1=a; return *this;}
   B& operator=(Z a){ x1=(a!=0); return *this;}

   CPM_IO
   CPM_ORDER
   Word nameOf()const{ return "B";}
   Word toWord()const{ return (x1 ? "true" : "false");}
   B operator!()const{ return B(!x1);}
```

```
   B operator&(const B& b)const{ return B(x1&&b.x1);}
   B operator|(const B& b)const{ return B(x1||b.x1);}
// escaping to Latin since 'not', 'and', and 'or' now
// are restricted in usage very much like C++ key words
   B non()const{ return B(!x1);}
   B et(const B& b)const{ return B(x1&&b.x1);}
   B vel(const B& b)const{ return B(x1||b.x1);}
   B ran(Z j=0){ return B(CpmRoot::ranT(x1,j));}
};

} //namespace

namespace CpmRoot{
// All similar declarations are in cpmword.h There, the present
// type is not yet defined. So we have to defer the declaration till
// here. That this works, is a triumph of C++'s templates.

   template<>
   class Name<CpmRootX::fpVoidToVoid>{ // also this type needs a name
   public:
      Name(){}
      Word operator()(CpmRootX::fpVoidToVoid const& t)const
      { t; return Word("fpVoidToVoid");}
   };

   template<>
   class Name<CpmRootX::fpWordToVoid>{ // also this type needs a name
   public:
      Name(){}
      Word operator()(CpmRootX::fpWordToVoid const& t)const
      { t; return Word("fpWordToVoid");}
   };

}// namespace

// new names
#define cpminf       CpmRootX::inf
#define cpmsup       CpmRootX::sup

#define cpmord      CpmRootX::order
#define cpmswp      CpmRootX::swap
#define cpmtin      CpmRootX::tinyNumber
#define cpmhug      CpmRootX::hugeNumber

#endif
```

32 cpmtypes.cpp

```
//? cpmtypes.cpp
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#include <stdlib.h>
#include <sstream>

#include <cpmtypes.h>

using namespace CpmSystem;
using namespace CpmRoot;
using CpmArrays::X2;

////////////////////////////////////////////////////////////////////

void CpmRootX::copyTextFile(ifstream& from, ofstream& to)
{
   char ch;
   while (from.get(ch)) to.put(ch);
   if (!from.eof() || to.bad())
   cpmerror("copyTextFile(const ifstream&, ofstream& to) failed");
     // Stroustrup Section 10.5 p. 351 top of page
}

void CpmRootX::copyTextFile(const Word& nameOfSource,
   const Word& nameOfCopy)
{
   Z mL=1;
   Word loc("copying "&nameOfSource&" to "&nameOfCopy);
   cpmassert(nameOfSource!="cpmcerr.txt",loc);
      // copying "cpmcerr.txt" caused problems, which are hard to fully
      // understand
   cpmmessage(mL,loc&" started");
   IFileStream from(nameOfSource);
```

```
   OFileStream to(nameOfCopy);
   CpmRootX::copyTextFile(from(),to());
   cpmmessage(mL,loc&" done");
}

string CpmRootX::toString(ifstream& from)
{
   if (!from) cpmerror("toString(): cannot open input file");
   ostringstream to;
   if (!to) cpmerror("toString(): cannot create output file stream");
   char ch;
   while(from.get(ch)) to.put(ch);
   if (!from.eof() || !to) cpmerror("failure in toString");
   return to.str();
}

////////////////////////// class Z1 ////////////////////////////////

const Z maxZ(std::numeric_limits<Z>::max());
const Z minZ(std::numeric_limits<Z>::min());

Z CpmRootX::sig(Z i)
{
   if (i==0) return Z(0);
   else if (i>0) return Z(1);
   else return Z(-1);
}

R CpmRootX::sig(R x)
{
   if (x==R(0.)) return R(0.);
   else if (x>R(0.)) return R(1.);
   else return R(-1.);
}

R CpmRootX::krn(Z i, Z j)
{
   return i==j ? R(1.) : R(0.);
}

Z CpmRootX::parity(Z k)
{
   return k%2==0 ? Z(1) : Z(-1);
}

Z CpmRootX::fac(Z n)
{
   cpmassert(n>0,"Z fac(Z)");
   Z res=1;
   for (Z i=1; i<=n; i++) res*=i;
```

```
   return res;
}

Z CpmRootX::pow(Z const& x, Z p)
{
   cpmassert(p>=0,"Z powerZ(Z,Z)");
   if (p==0) return Z(0);
   else if (p==1) return x;
   else if (p==2) return x*x;
   else{
      Z res=x;
      Z nMult=p-1;
      for (Z i=1;i<=nMult;i++) res*=x;
      return res;
   }
}

bool CpmRootX::isPowOfTwo(Z const& x)
{
   Z fac{2};
   if (x<fac) return false;
   Z f=fac;
   while (f<x) f*=fac;
   return f==x;
}

Z CpmRootX::nextPowOfTwo(Z const& x)
{
   if (x<2) return Z(2);
   Z f=2;
   while (f<x) f*=2;
   return Z(f);
}

Z CpmRootX::nextLog2(Z const& x)
{
   Z res{1};
   Z f{2};
   while (f<x){ // after the loop f>=x1
      res++;
      f*=2; // guarantees that f grows over all bounderies
   }
   return res;
}

#if !defined(CPM_MP) && !defined(CPM_MPREAL)
////////////////////////// class R1 ////////////////////////////////

using CpmRootX::R1;
using CpmRootX::Z1;
```

```
const R1 R1::max(std::numeric_limits<R>::max());
const R1 R1::eps1(std::numeric_limits<R>::epsilon());
const R1 R1::eps0(std::numeric_limits<R>::min());

const R1 R1::min(-R1::max);
const R1 R1::logmax(::log(R1::max));
const R1 R1::log10max(::log10(R1::max));
const R1 R1::pi(cpmpi);
const R1 R1::piInv(R(1)/cpmpi);

bool R1::prnOn(ostream& out)const
{
   return CpmRoot::write(x1,out);
}

bool R1::scanFrom(istream& in)
{
   return CpmRoot::read(x1,in);
}

Z R1::com(R1 a)const
{
   if (x1<a.x1) return 1;
   if (x1>a.x1) return -1;
   return 0;
}

X2<R,R1 > R1::polDec()const
{
   if (x1>=0) return X2<R,R1 >(x1,R1(1.));
   else return X2<R,R1 >(-x1,R1(-1.));
}

R R1::dis(const R1& x)const
{
   return CpmRoot::dis(x1,x.x1);
}

// functions referring to R as a discrete 'lattice'

R1& R1::next_(void)
{
   R eps_=eps0.x1;
   R xMem=x1;
   while (x1==xMem){
      x1+=eps_;
      eps_*=2;
   }
   return *this;
```

```
}

R1& R1::prv_(void)
{
   R eps_=eps0.x1;
   R xMem=x1;
   while (x1==xMem){
      x1-=eps_;
      eps_*=2;
   }
   return *this;
}

namespace{

   void step_p(R& x, R& e)
   {
      R xMem=x;
      Label:
      x+=e;
      if (x==xMem){
         e*=2;
         goto Label;
      } // thus we end up with x!=xMem
   }

   void step_m(R& x, R& e)
   {
      R xMem=x;
      Label:
      x-=e;
      if (x==xMem){
         e*=2;
         goto Label;
      } // thus we end up with x!=xMem
   }
}

Z R1::dz(R a)const
{
   Z res=0;
   R e=R1::eps0.x1;
   if (x1>=a){
      while (x1>a){
         step_p(a,e);
         ++res;
      }
   }
   else{
      while (x1<a){
```

```
         step_m(a,e);
         --res;
      }
   }
   return res;
}

R R1::eps(Z dir)const
{
   R res= (x1>=1 || x1<=-1) ? eps1.x1 : eps0.x1;
   if (dir<0) res*=-1;
   R xc=x1;
   Label:
   xc+=res;
   if (xc==x1){
      res*=2;
      goto Label;
   }
   return res;
}

R R1::dr(R a)const
{
   if (a==x1) return 0;
   R dis=x1-a;
   if (dis<0) dis*=-1;
   if (x1>a){
      R ex=eps(-1);
      R ea=R1(a).eps(1);
      return R(2)*dis/(ea-ex);
   }
   else{
      R ex=eps(1);
      R ea=R1(a).eps(-1);
      return R(2)*dis/(ex-ea);
   }
}

Z R1::run(R final, ostream& str, Z writePeriod)const
{
   R xMem=x1;
   R xRun=x1;
   R e=R1::eps0.x1;
   Z wrt=0;
   Z count=0;
   if (final>xRun){
      while(xRun<final){
         step_p(xRun,e);
         count++;
         if (count==writePeriod){
```

```
            str<<"xRun-xMem="<<xRun-xMem<<" e="<<e<<endl;
            ++wrt;
            count=0;
         }
      }
   }
   else if (final<xRun){
      while(xRun>final){
         step_m(xRun,e);
         count++;
         if (count==writePeriod){
            str<<"xRun-xMem="<<xRun-xMem<<" e="<<e<<endl;
            ++wrt;
            count=0;
         }
      }
   }
   else;
   return wrt;
}

// save functions

R R1::log()const
{
   if (x1>0) return ::log(x1);
   else if (x1<0) return ::log(-x1);
   else return -logmax;
}

R R1::log10()const
{
   if (x1>0) return ::log10(x1);
   else if (x1<0) return ::log10(-x1);
   else return -log10max;
}

R R1::exp()const
{
   if (x1>logmax) return max;
   else if (x1<-logmax) return 0;
   else return ::exp(x1);
}

R1 R1::inv()const
{
   R e0=eps0;
   if (x1>e0 || x1<-e0) return R1(1./x1);
   else if (x1>=0) return R1(max);
   else return R1(min);
```

```
}

using CpmRootX::Z1;
/////////////////////////// class Z1 ///////////////////////////////

const Z1 Z1::max(std::numeric_limits<Z>::max());
const Z1 Z1::min(std::numeric_limits<Z>::min());

Z1 Z1::pow(Z p)const
{
   cpmassert(p>=0,"Z1 Z1::power(Z p)");
   if (p==0) return Z1();
   else if (p==1) return Z1(x1);
   else if (p==2) return Z1(x1*x1);
   else{
      Z res=x1;
      Z nMult=p-1;
      for (Z i=1;i<=nMult;i++) res*=x1;
      return Z1(res);
   }
}

bool Z1::prnOn(ostream& out)const
{
   return CpmRoot::write(x1,out);
}

bool Z1::scanFrom(istream& in)
{
   return CpmRoot::read(x1,in);
}

Z Z1::com(const Z1& a)const
{
   if (x1<a.x1) return 1;
   if (x1>a.x1) return -1;
   return 0;
}

X2<Z,Z1 > Z1::polDec()const
{
   if (x1>=0) return X2<Z,Z1 >(x1,Z1(1));
   else return X2<Z,Z1 >(-x1,Z1(-1));
}

R Z1::dis(const Z1& x)const
{
   return CpmRoot::disT<Z>(x1,x.x1);
}
```

```
bool Z1::isPowOfTwo()const
      // Let us write x for *this, and let N={1,2,3,...} (no zero!).
      // We return true iff x=2^n with some n \in N
{
   if (x1<2) return false;
   Z f=2;
   while (f<x1) f*=2;
   return f==x1;
}

Z1 Z1::nextPowOfTwo()const
{
   if (x1<2) return Z1(2);
   Z f=2;
   while (f<x1) f*=2;
   return Z1(f);
}

Z1 Z1::nextLog2()const
{
   Z res=1;
   Z f=2;
   while (f<x1){ // after the loop f>=x1
      res++;
      f*=2; // guarantees that f grows over all bounderies
   }
   return Z1(res);
}

#endif

////////////////////////// class B ////////////////////////////////
using CpmRootX::B;

bool B::prnOn(ostream& out)const
{
   return CpmRoot::write(x1,out);
}

bool B::scanFrom(istream& in)
{
   return CpmRoot::read(x1,in);
}

Z B::com(const B& a)const
{
   if (x1<a.x1) return 1;
   if (x1>a.x1) return -1;
   return 0;
}
```

33 cpmuc.h

```
//? cpmuc.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_UC_H_
#define CPM_UC_H_
/*
Purpose: Class UseCount is a tool for implementing reference
counting.

Credit: Modified from Andrew Koenig: Ruminations on C++,
AT&T Ch. 7. Authoritative and enlightening treatment. However,
compared to the present version, some essentials are missing
in Koenig's book. Desastrous misprints in the code there!

misprints in Koenig:
(p. 72 reattach():
replace --p by --*p,
p. 73 makeonly(): eliminate p=new ...; )

Recent history:
    2012-01-11 non-constant functions renamed to ending in '_'
    2017-02-18 p renamed to p_, u renamed to u_
    2017-05-23 macro const& used instead of $, '$' needs blanks to\
     separate it
      from variable names or type names. However, e. g.
      bool reattach_(UseCount $);
      is OK
*/

#include <cpmbasicinterfaces.h>

//////////////////////////// class UseCount ///////////////////////////////
```

```
namespace CpmArrays{

   using CpmRoot::Z;

class UseCount { // support for reference counting and 'copy on write'
   // class UseCount of Koenig. See explanations there.
   // Instances of this class are used as data members in client classes
   // which may describe the state of several instances by a common piece
   // of 'free store'. Typical example is class P<> within the present
   // file. Classes derived from P<> are defined in cpmp.h, cpmfl.h,
   // and in cpmfile.h
public:
   UseCount():p_(new Z(1)){}
   UseCount(UseCount const& u):p_(u.p_){++*p_;}
   ~UseCount(){ if (--*p_==0) delete p_;}
      // not virtual since no derivations will be defined
   bool only()const{return *p_==1;}
      //: only
      // to be called in the destructor of the client class
   bool makeOnly()const{if (*p_==1) return false; --*p_; return true;}
      //: make only
      // The name 'makeOnly' needs explanation: If *this is 'only' already
      // no 'making' is involved and 'makeOnly' returns 'false'.
      // If it is not 'only' we do something and thus return 'true'.
      // What is achieved in this doing is, however, not necessarily the
      // status 'only' one step in the direction of becoming 'only'.
      // Only if we had *p_==2 prior to calling makeOnly() we have
      // *p_==1, i.e. the status 'only' after the call.
      // To be called in the assignment operator of the client class.
      // Named unalias() by Bruce Eckel (p. 437). To me both names
      // don't provide a clear suggestion.
      // Consider a typical usage of this function as it appears in
      // the copy on write function (to be called in the assignment
      // operator) of the array class V<T> which has data
      // T* p_, UseCount u_, ... :
      // template <class T>
      // void V<T>::cow_(void){
      //    if (u_.makeonly()){ p_= (sz_==0 ? 0 : copy()); u_.startNew();}
      // }
      // This function thus calls u_.makeonly() once. Only if its return
      // value is true any action happens. This is the case only if
      // *(u_.p_) > 1 so that the piece of free store which u_'s client is
      // using (and to which p_ is pointing) is also being used by a
      // further client. Then action is needed: (*this) connects to a copy
      // of the original piece of free store so that this original piece
      // will have one user less. This is what u_.makeonly does in its
      // instruction --*p. The situation thus is that the old users of the
      // 'land' still use it and the individual *this who tries to change
      // things has to move to a new piece of 'land' which is made into
      // a copy of his old 'land' and on which the intended changes can be
```

```
      // made without interfering with the needs of the users of the
      // 'old land'. This is now a clear procedure. Earlier my
      // understanding was that the old users have move and it was not
      // clear how *this would enforce such a cooperation of the old
      // users.

   bool reattach_(UseCount const&);
      //:: reattach
      //'::' means that the name is longer than the one formed according
      // to the CPM standard abbreviation scheme. In most cases no
      // abbreviation at all.
      // To be called in the assignment operator of the client class
      // See P<> and V<>.
      // Recall that the names of non-constant functions end in '_'.

   void startNew_(){ p_=new Z(1);}
      //:: start new
      // my addition
   Z getCount(void)const{return *p_;}
      //:: get count

private:
   UseCount& operator=(UseCount const&);
      // not implemented,to make sure that this will never be used
   Z* p_;
};

////////////////////////  class P< >  ////////////////////////////////////
// Class of constant smart pointers, no arrays (thus no indexing),
// reference counting, but no copy on write. Tool for implementation
// of function class F<X,Y>.
// See cpmp.h for a derived class which implements copy on write.

template <class T>
   // T is a any non-abstract class or any built-in type.
   // If T is immutable, i.e. an instance of T cannot be modified after
   // it has been created, (see Andrew Koenig: Ruminations on C++, AT&T
   // 1997, p. 178, 190 for the relevance of immutable types) P<T>
   // implements the 'strict value interface' (see file cpmv.h).
   // In the notation of predicate logic:
   // non-abstract T & immutable T ==> value class P<T>
   //
   // Let T(A const& a) be a constructor of type T, then
   // the normal syntax for creating an instance of P<T> is
   // A a=...;
   // P<T> pt=new T(a)
   // or
   // P<T> pt;
   // pt=P<T>(new T(a));
   // or
```

```
   // P<T> pt(new T(a));

class P { // 'P' as in 'pointer', constant smart pointer.
   // Handle class the instances of which have to be initialized by
   // 'new'-generated pointers. The class takes the full responsibility
   // for deleting these pointers properly.
   typedef P<T> Type; // needed for declaration macro CPM_ORDER
public:
   CPM_ORDER
      // Will be implemented by comparison of adresses.
      // No order-related operators of T used.

   P(T *p=0):p_(p){}
      // Constructor, including default constructor.
      // important !!!: use this constructor only
      // for pointers initialized with a 'new' statement (on free store,
      // not static!) Important also: This allows to create P<T>'s from
      // pointers X* x where X is derived from T.
      // P<T>(new X(..arguments..)) will hold all information, that
      // the object X x(..arguments..) contains, although T does not know
      // about the additional members of X.
      // Also u_ gets initialized by its default constructor which sets
      // the use count to 1.
      // The piece of free store that thus comes under the control of P
      // is of course not associated with a use-counter already. The case
      // that he is, will be handled by the next function:

   P(P<T> const& h):u_(h.u_),p_(h.p_){}
      // copy constructor
      // Here the pointer h.p_ points to a piece of storage which is
      // already in use by h.u_.getCount() clients.
      // The copy constructor of UseCount is made such that it increases
      // the usecount by 1.

   virtual ~P(){ if (u_.only()) delete p_; }
      // destructor

   P<T>& operator=(P<T> const& h);
      // assignment

// access operator, which does not allow to change p
   T const& operator()(void)const{ return *p_;}
      // Let pt an instance of P<T>; then it is conventional to denote
      // the value as *pt. According to my aim to avoid pointers in
      // public interfaces, I also tend to avoid pointer notation.
      // So I denote the value of pt by pt().
      // This looks very natural since 'getting to the data content' of a
      // container-like quantity is much like evaluating a function.
      // Since no argument is needed (as for standard rand(), for example)
      // a void pair of brackets gives the right association.
```

```
      // Assume that T = V<X>, then we may form pt()[1].
      // In dereferencing notation we had to write (*pt)[1].

   bool isVal()const{ return !(p_==0);}
      //: is valid
protected:
   T *p_;
   UseCount u_;
};

////////////////////////////// Implementation ////////////////////////////

template <class T>
Z P<T>::com(P<T> const& obj)const
{
   if (p_<obj.p_) return 1;
   if (p_>obj.p_) return -1;
   return 0;
}

template <class T>
P<T>& P<T>::operator=(P<T> const& h)
{
   if (u_.reattach_(h.u_)) delete p_;
   p_=h.p_;
   return *this;
}

} // namespace

#endif
```

34 cpmuc.cpp

```
//? cpmuc.cpp
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#include <cpmuc.h>   // see for explanations

bool CpmArrays::UseCount::reattach_(UseCount const& u)
{
   ++*u.p_;
      // no conflict with constantness of u !!
      // count of u becomes incremented whereas in the next statement
      // the count of *this becomes decremented
   if (--*p_==0){     // --p==0 in Koenigs book
      delete p_;
      p_=u.p_;
      return true;
   }
   else{
      p_=u.p_;
      return false;
   }
}
```

35 cpmv.h

```
//? cpmv.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_V_H_
#define CPM_V_H_
/*

Purpose: Basic array class with valid indexing ranging in a contiguous
   subset of Z. Two cases are of particular interest: That the lowest
   valid index is 0 (as for std::vector<>) or 1 (as e.g. in the functions
   in Press et al.). When dealing with FFT index ranges
   {-n,...,-1,0,1,...,n} are adequate (not yet implemented).
   Most constructors of V create instances of V<T> with valid indexes
   starting at 1. The two functions b_(Z) and e_(Z) provide means
   to arbitrarily shift the range of valid indexes. Access to components
   by means of indexing is range checked.

History: Till October 2010 there was a class template Vl<T> with
indexing starting at 0 and - based on that (not derived from that)
a class template V<T> with indexing starting at 1. This caused an
uncomfortable situation: Whenever dealing with a topic where I felt
that efficiency would be of utmost importance (e.g. font representation
and quantum dynamics) I was tempted to use Vl<> not only as an
internal device but also in the interface of public member functions.
This made the implementation code and even the classes dependent
on a decision that with a slight twist of emphasis could also have
been made differently. The new state of affairs is that Vl has been
eliminated (actually replaced by V) in all C+- code. If it should still
shine up in some comment, ignore it.

Recent history:
    2012-01-11 function cow() renamed to cow_()
    2012-01-18 function X2<Z,bool> findAsc(T const& t)const added
```

```
    2012-01-19 function prnOn modified so that indexes are printed
      (in commentarized form) together with the components.
    2017-02-18 Short report on a project which looked promising
      but was finally abandoned:
      The rather mature state that C++ has reached with C++11 triggered the
      desire to make more systematic use of the standard library
      facilities. Especially that now all standard containers are said to
      implement moving as a replacement of copying where appropriate
      promised to make it feasible to replace T* p_ by std::vector<T> p_
      and thus free the implementation of V<T> from defining copy, move,
      assignment operators. The first disappointing problem was that then
      V<bool>, since based on std::vector<bool>, did no longer work in my
      code which used operator[](Z) for V<bool> as in all other V<T>'s.
      Of course replacing V<bool> by V<B> would help. But re-writing all
      the many functions of V<T> in the new style looks disappointingly
      tedious to me. Before making a second attempt I need to understand
      whether the new move functionality is in fact a full replacement for
      the reference counting and copy on write functionality which is
      implemented in V<T> as of today. There is a project
      'codingexperiments' in ~/e/cpm/codingexperiments which illustrates
      the malfunction of vector<bool>, which also is well known to the
      WWW. Further there is ~/e/cpm0ExperimentBasedOnvector which
      contains the experimental version of cpmv.h. In this version by far
      not all uses of T* are replaced by vector<T>.


Credit: Modified and extended from Andrew Koenig: Ruminations on C++,
AT&T Ch. 7. Authoritative and enlightening treatment. However,
compared to the present version, some essentials are missing in
Koenig's book. Also two misprints caused trouble.

This defines a template V<T> of T-valued lists or 'vectors with
T-valued components'. It is similar to std::vector<T> of the STL
but it differs from this in some respect. In describing
V<T> and these differences, I'll introduce some notions and notations
that will be used over and over in defining and stating properties of
other CPM classes
( CPM = 'C+-' or 'Classes for Physics and Mathematics').

1. V<> instances (called simply 'Vectors' here) never request more
   memory than necessary. Thus appending elements always needs new
   allocation of memory.
   V<> is more adapted to the role of vectors as compound
   (structured) quantities than to the role as a sequential container for
   its components. Therefore, appending components at the end is
   not basic to the concept and is not worth special optimizing effords.
   Notice that V<> is not a 'polymorphic container'
   i.e. the components can hold only T typed objects and not the
   additional information content which instances of classes derived
   from T may carry. We may, however, form V<T*> for any type or class
```

```
   to support polymorphism in the old-fashioned pointer-based way.
   A safer way is to use the polymorphic vector template Vp mentioned
   in item 5.

2. V<> implements 'reference counting' and 'copy on write'. Thus
   there is no need to work with references to vectors to avoid
   superfluous copying. Assume that we nevertheless observe the
   rule to declare function arguments always as references
   or constant references. Then, the main effect of reference
   counting is that large objects built within the body of a
   function (such as matrix multiplication) will not be copied
   to the outside but simply referenced. If they are no longer
   needed, they nevertheless get deleted automatically.
   If such a referenced large object gets changed by program
   action and some old client still needs the unchanged version
   for his reference, a copy of the large object is made automatically
   and made available to its user.

3. Requirements on T in order to allow the formation of V<T>:
   To illustrate the point up front: std::vector<T> will only be built
   (at least with the MS Visual C++ 5.0 compiler) if T<T and T==T are
   defined. For V<T> there is no such restriction. To be sure: T<T is an
   obvious abbreviation of the statement:
   'for all objects t1 and t2 of type T, the expression t1<t2 is defined'
   (compare David R. Musser, Atul Saini: STL Tutorial and Reference
   Guide, Addison-Wesley 1996, p. 246). Now the pertinent statement:
   V<T>  is well defined if (not iff!) T is a built-in type or a class
   which 'implements the value interface'. Then, V<T> also implements the
   value interface. If, moreover, T 'implements the strict value
   interface', then V<T> also implements the strict value interface.
   Here, a class T is said to implement the value interface if it
   publicly defines T(), T(const T&), and T& operator=(const T&).
   T is said to implement the strict value interface if, in addition, it
   has value semantics (as opposed to pointer semantics, see
   Andrew Koenig: Ruminations on C++, AT&T 1997, p. 62. and
   Bjarne Stroustrup: The C++ Programming Language, 3. edition,
   Addison-Wesley 1997, p. 294). Classes that implement the strict value
   are most convenient to use. Code only using such quantities is
   normally easy to understand. To have an even shorter denotation for
   this favorable species we call such a class a value class or a bit
   more general:

      T is a value class : <==> T is a built-in type or a class
      that implements the strict value interface.

   As was stated already, we have:

      value class T ==> value class V<T>

   The predicates 'T satisfies the value interface' and
```

```
   'T satisfies the strict value interface' can be evaluated for
   classes providing random generators by means of the test classes
   Test_v<T> and Test_sv<T> defined in file cpmtests.h. Although V<>
   provides no random generator (Vr<>, to be mentioned soon, does)
   the code of Test_sv<T> can be read as an operational definition
   of the concept 'strict value interface'. An interesting (or common-
   place ?) observation: syntactic aspects of a program are characterized
   by the program's interaction with the compiler; semantic aspects
   ('pointer sematics',...) are characterized by the data created during
   execution.

   Requiring order operators or even arithmetic operators in T, allows to
   define T-'valued' Vectors which themselves carry natural
   order/arithmetic operators.
   Therefore V is only the starting point in a series of vector
   templates with increasing functionality, accompanied by increasing
   assumptions on the structure of T. Presently this hierarchy looks as
   follows:

      V<T> , Vo<T> , Va<T> , Vr<T>

   Every such class is derived from its predecessor in this series
   by derivation without adding new data members. Thus there are
   unambiguous casts between all these classes.

      Vo : More order related methods added to V, basic order functions
         which are declared in CPM_ORDER now already here.
      Va : arithmetic operations added to Vo
      Vr : rich infrastructure supporting automated testing of
         internal consistency.

   This approach of integrating the functions (algorithms) into an
   inheritance tree of template classes is radically different from the
   STL approach. STL keeps algorithms as separate entities outside the
   classes and uses iterators and adaptors for making them work together.
   Both methods have their advantages and disadvantages. It is probably
   fair to say that the C+- approach is less universal but more
   convenient to use within the framework it fits.

4. On polymorphism: V<T*> may be formed, but doesn't implement the value
   interface for T and derived classes. However, with the smart pointer
   templates P<T>, Pp<T>, and Po<T> defined in file cpmp.h we can form
   e.g. V<Pp<T> > and get the functionality (together with some 'extras')
   which one would expect from  V<T*>. See file cpmp.h for details and
   the polymorphic version Vp of the vector templates mentioned so far.
   See test class PolymorphicMulti<*,*,*> in cpmtests.h for a operational
   definition of polymorphism of container classes.
*/

#include <cpmfl.h> // Includes <cpmuc.h> for std::size_t
```

```
   // (and std::ptrdiff_t, which is not being used so far).
   // Small and efficient function template with minimum
   // infra-structure requirements.
#include <cpmzinterval.h>
   // includes cpmsystem.h and cpmx.h
   // With using arrays something may go wrong and so messaging
   // capability is indispensible.

#include <cpmmacros.h>
   // Provides help to write debugging-friendly function blocks.
#include <vector>
#include <set>
////////////////////////////// class V<> ///////////////////////////////
// Array with index check, reference counting, and copy on write, and
// 'value semantics' (as opposed to 'pointer semantics')
// Generalized from Koenig's class Handle p. 72-73.
// Index range is now defined as an instance of class IvZ. So each
// 'vector' has its individual index range which may start with 1
// (following the convention of the Numerical Receipes) or with 0 as
// for into the 'vector' of the STL.
//  V< V<ColRef> > is the type of bitmap data in my graphical workhorse
// class Img24. This can be considered a proof for good performance of
// the class. Efficient conversion functions from and to std::vector<>
// are now provided.
// There is a nice way to iterate over the whole index range without
// mentioning the bounds as 0 and dim-1, or as 1 and dim:
//    V<T> v=....;
//    for (Z i=v.b();i<=v.e();++i) v[i]=...;

namespace CpmArrays{

   using namespace CpmStd;

   using CpmRoot::Word;
   using CpmRoot::Z;
   using CpmRoot::R;
   using CpmRoot::Root;
   using CpmSystem::Error;
   using CpmFunctions::F;

#ifdef CPM_Fn
   using CpmFunctions::F1;
#endif

// some infrastructure

   enum Begin { LEAN };
      //: begin
      // Never form V<Begin>
```

```
   enum Outside { DEFAULT, CYCLIC, CONSTANT };
      //: outside
      // Controls the meaning of 'out of range indexes'.
      // DEFAULT: definition as the default value associated with
      // the type under consideration
      // CYCLIC: setting the meaning of out of range indexes by
      // cyclic repetition of value
      // CONSTANT: continuation as constant from the nearest value

   extern Z dimMax;
      // If the dimension of an array was either the result of a
      // calculation or of reading from a file, then the result may be
      // off the programmer's intent by orders of magnitude if something
      // went wrong. So it is helpful to exclude unnaturally large arrays
      // from becoming allocated.

   extern bool ranChc;
      //: range check
      // Initialized as true.

   extern bool ranChcAlw;
      //: range check always
      // If this is true, all access operators even those the name
      // of which suggests the opposite get checked --- with poor
      // diagnostics, though.
      // Initialized as true, since access to non-allocated memory,
      // as a rule, causes disaster. I had to discover in 2008-03-03
      // that such a case happened in the workhorse function
      // CpmGraphics::Graph::mark(V< V<C> >,...)
      // on a regular basis, due to an seemingly safe but actually
      // un-safe usage of V<>::cui().

   extern bool signal;


   void setDimMax(Z n);
      // sets dimMax=n unless n<0. In this case error with message

   Z safeDim(Z n);
      // returns n for 0<=n<=dimMax, otherwise error with message

   Z makeIndValNotMember(Z i, IvZ iv, Outside mode);
       //: make index valid

template <class T>
   // We assume that T provides (explicitly or implicitly)
   // copy constructor, and assignment or that T is a built-in type.
   // If the index operator [] is to be used and the variable ranChc
   // (which is initialized as 'true') was not set to 'false' an out
   // of range error will result in a runtime error which will be
```

```
   // documented on cpmcerr.txt. See ranChcAlw for an additional control.
   // The error message is particularly explicit (indicating the type of
   // T) when also the macro CPM_NAMEOF is defined. If this is the case,
   // the type T needs to define the member function
   // CpmRoot::Word nameOf()const. For all Cpm-classes this is the case
   // and for user classes, there should be no difficulty in adding such
   // a function. If one wants to make use of the function declared
   // by the declaration macros CPM_ORDER type T needs to define the
   // member function  CpmRoot::Z com(T const&)const;
   // If one wants to make use of the functions declared by the
   // declaration macro CPM_IO, type T needs to define the member
   // functions bool prnOn(ostream&)const and bool scanFrom(istream&).
   // These requirements do not apply to basic types:
   // For T = N, Z, R, Rh, L, bool, string one  may use all functions of
   // V<T> without further requirements.

class V{ // vector template, indexing starts with 1 by default.
   // The index range can however be shifted or even initially set
   // arbitrarily.
   // Although implementation details are inspired from handle classes,
   // the V class is a value array and not a handle class.
   // All allocations made with new T[] all de-allocations are  delete[]
   // All data are private, so the only access to data in
   // derived classes is over the public functions of the class. All
   // these incorporate reference counting internally where needed (only
   // if the preprocessing directive CPM_USECOUNT is active). The user of
   // the class can't see this and has not to be aware of this.

   typedef V<T> Type;

public:
   CPM_IO
   CPM_ORDER
   R dis(V<T> const& h)const;

explicit V(Z n=0) ;
      // Has n components. Gives an error for n<0 and n>dimMax.
      // The components are initialized by the default constructor
      // of T if T is a class for which such a constructor is defined
      // (explicitly or implicitly).
      // If T is a built-in type, initialization is done as 0.
      // See BS3, p. 131 for initialization of built-in types via
      // formal constructor calls.
      // If n>0, the first valid index is 1, which reflects the
      // normal behavior of class V.

   V(Z n, Begin bg);
      // Defined as the previous function. But the first valid index
      // is 0 (if n>0 so that there is at least one valid index).
      // This is a somewhat contrived construction: One has to make sure
```

```
      // that this does not interfere with the definition
      // V(Z n, T const& t, Z first=1) for some choice of T. Since we
      // agree on using V<T> only for C+- types T, we will never be
      // tempted to consider V<Begin>.
      // Typical usage:
      //    V<R> v(4,LEAN); // recall: enum Begin { LEAN }
      // lets v have valid indexes 0,1,2,3. v[0]=...=v[3]=0.
      //    V<R> w(4);
      // lets w have valid indexes 1,2,3,4. w[1]=...=w[4]=0.

explicit V(IvZ const& iv);
      // Has a component for each element of iv.
      // The components are initialized by the default constructor
      // of T if T is a class for which such a constructor is defined
      // (explicitely or implicitely).
      // If T is a built-in type, initialization is done as 0.
      // See BS3, p. 131 for initialization of built-in types via
      // formal constructor calls.

   V(std::vector<T> const& v, Z first);
      // Construction from a standard library vector. This is useful
      // for interaction with the standard containers (which don't
      // implement reference counting). The second argument gives the
      // begin of the valid index range. A value 0 lets the result of
      // the construction behave like v with respect to indexing.

   V(Z n, T const& t, Z first=1) ;
      // Has n components all initialized as t.
      // The third argument gives the first valid index.

   V(Z n, T const& t, Begin bg);
      // Has n components all initialized as t.
      // The third argument (that can have only one value, namely LEAN)
      // says that the first valid index is 0.

   V(IvZ const& iv, T const& t);
      // Has iv.car() components all initialized as t

   V(Z n, F<Z,T> const& f);
      // construction from a function. Of course,
      // V<T> v(n,f);
      // implies v[i]==f(i) for all valid indexes i of v.

   V(IvZ const& iv, F<Z,T> const& f);
      // construction from a function. Of course,
      // V<T> v(iv,f);
      // implies v[i]==f(i) for all valid indexes i of v.

   V(V<T> const& h);
      // copy constructor
```

```
// constructors from explicit lists
   explicit V(std::initializer_list<T> il );
      // requires C++11
      // constructors from explicit lists such as
      // V<Z> v{1,2,4,8};

   virtual ~V();
      // destructor

   V<T>& operator=(V<T> const& h);
      // assignment

   virtual V<T>* clone(void)const{ return new V(*this);}

   V<T> toClnBase()const{ return *this;}
      //: to clone base

   Z dim()const { return sz_;}
      //: dimension
      // Returns the number of components of the vector *this

   Z size()const { return sz_;}
      //: size
      // for uniformity with STL

   IvZ dom()const { return iv_;}
      //: domain
      // Notice that with the array *this there is the
      // function f: {iv_b(),...,iv_.e()}-->T, i|-->(*this)[i]
      // associated in a natural manner.
      // For this function, dom() is just the domain.
      // The understanding of arrays as functions with domains of type
      // IvZ seems to be a good guide for defining some of the member
      // functions in a more natural manner by using arguments of type
      // IvZ. Present examples are the functions fa_ and valOn.

   bool isVoid()const { return iv_.isVoid();}
      //: is void
      //  short answer on whether the dimension is zero

   bool valInd(Z i)const{ return iv_.hasElm(i);}
      //: valid index
      // returns the validity of i as an index of *this

   Z makeIndVal(Z i, Outside mode=CYCLIC )const
   {
      if (iv_.hasElm(i)) return i;
      if (mode==CYCLIC) return iv_.cyc(i);
      else return iv_.con(i);
```

```
   }
      //: make index valid
      // If i is a valid index we return i. If not, the result is
      // iv_.cyc(i) for mode=CYCLIC and iv_.con(i) else. Notice that the
      // normal treatment of mode==DEFAULT works not by replacing one
      // value of the index by another. It works on the value of vector
      // components and makes use of the default constructor T()

   bool sameDom(V<T> const& v)const{ return iv_==v.iv_;}
      //: same domain

   std::vector<T> std()const;
      //: standard
      // Returns a STL-vector which holds all components of *this.

   virtual Word nameOf()const;
      //: name of
      // returns a name of the type

// constant access functions

   const T& operator[](Z i)const;
      // Getting to the value of a component of a const instant of
      // V<T>. For instance
      // const V<T> v = ... ;
      // T t = v[3];
      // If CPM_USECOUNT is enabled (the normal case) the actual
      // process of going from v to v[3] depends on whether v is of type
      // V<T> or const V<T>. In the non-constant case the evaluation of []
      // may involve a copy action on v (and returning the component of
      // the copy). If we intend only to read the component, such an
      // copy action is not neededd and should be avoided by casting v
      // to type const V<T>.
      // Casting v from V<T> to const V<T> works this way:
      //    v = static_cast<const V<T>>(v);
      // Casting back to mutable seems to work only by using a new name:
      //    auto vMutable = const_cast<V<T>&>(v);
      //    vMutable[1] = T(); //( for instance)
      // or by applying mutating operations to an anonymous object as in:
      //    const_cast<V<T>&>(v)[1]=T();
      // Notice the '&' here which the compiler requires. By the way, the
      // effect of the 'anonymus action' actually is to change the state
      // of v: v[1]==T().
      // My analysis of the situation is in ~/codingexperiments/main.cpp
      // In the following there are many pairs of functions
      // T const& f(...)const;
      // T& f(...);
      // for which the above considerations apply mutandis mutatis.

   T const& cui(Z i)const;
```

```
      //. component (with) unchecked index
      // getting to the value of a component for 'read', e.g.
      // T t=cui(i);
      // It helps to write efficient code in classes which use V<>-typed
      // data members.
      // Notice that writing loops by using b() and e() to define
      // the range is much safer than using limits like 1 and dim().
      // Index range check is missing only if variable ranChcAlw was
      // changed to 'false'.

   T const& cyc(Z i)const;
      //: cyclic
      // getting to the value of a component for 'read', e.g.
      // T t=cyc(i);
      // Here i is understood as cyclic (i.e. i modulo sz_). So no value
      // of the index has to be considered 'out of range' and for i
      // 'in range' cyc(i)==(*this)[i]

   T const& li(Z i)const{ return *(p_+i);}
      //. lean index
      // The valid range is for (Z i=0;i<sz_;++i) li(i)

   T& li(Z i)
      //. lean index
   {
#if defined(CPM_USECOUNT)
      cow_();
#endif
      return *(p_+i);
   }

   T const& con(Z i)const;
      //: constant
      // Same logic as cyc() but with constant continuation
      // instead of cyclic.

   T operator()(Z i, Outside mode=DEFAULT)const;
      //: ()
      // Read access defined for all i.
      // By this definition, a vector becomes a mapping from Z to T.
      // For i's outside the proper range,
      // the default T is returned for mode==DEFAULT or if
      // sz_==0. If mode==CYCLIC cyc(i) gets returned.
      // For mode==CONSTANT we return p_[0] for i<=0 and p_[sz_-1]
      // for i>=sz_.
      // Notice that the return value is not a reference in
      // accordance with function behavior.

   T const& read(Z i, Outside mode=DEFAULT)const;
      //: read
```

```
    // Same as previous function but returning a reference instead of a
    // T.
    // Reading components in an efficient (as &'s), safe and flexible
    // manner.
    // See T operator()(Z i, Outside mode=DEFAULT)const; for the
    // meaning of the second argument.

  T const& r(Z i)const{ return p_[iv_.ri(i)];}
    //. read
    // Unchecked and fast version of T const& oparator[](Z i)const

  T& w(Z i)const{ return p_[iv_.ri(i)];}
    //. write
    // Unchecked and fast version of T& oparator[](Z i)

  F<Z,T> fnc(Outside meth=DEFAULT)const;
    // function
    // returns the function Z --> T, i |--> (*this)(i,meth)

// non-constant access functions
  T& operator[](Z i);
    //: []
    // See T const& operator[](Z i)const for discussion of details
    // that are relevant in the case that CPM_USECOUNT is defined.

  T& cui(Z i);
    //. component (with) unchecked index
    // setting to the value of a component
    // T t=...;
    // V<T> x=...;
    // x.cui(i)=t;

  T& cyc(Z i);
    //: cyclic
    // setting to the value of a component
    // T t=...;
    // V<T> x=...;
    // x.cyc(i)=t; meaning of i is modulo sz_. So i is never out of
    // range

  T& con(Z i);
    //: constant
    // Same logic as cyc() but with constant continuation
    // instead of cyclic.

  V<T>& b_(Z i){ iv_.b_(i); return *this;}
    //. set b(), i.e. the first valid index.
    // For instance
    // V<R> v("",sqrt(2),sqrt(3),sqrt(4),sqrt(5));
    // for (Z i=1;i<=v.dim();++i) cout<<v[i]<<endl;
```

```
      // v.b_(0);
      // for (Z i=0;i<v.dim();++i) cout<<v[i]<<endl;
      // shows all components of the vector in both cases.

   V<T>& e_(Z i){ iv_.e_(i); return *this;}
      //. set e(),  i.e. the last valid index.

// accessing the first and the last element for reading
   T const& fir()const;
      //: first
   T const& last()const;
      //: last

// accessing the first and the last element for writing
   T& fir();
      //: first
   T& last();
      //: last

// getting the first and the last valid index
   Z b()const { return iv_.b();}
      //. begin
      // Returns the first valid index.

   Z e()const { return iv_.e();}
      //. end
      // Returns the last valid index.
      // Allows to write loops over all components as
      // for (Z i=v.b();i<=v.e();i++) ... v[i] ...;
      // Note that this is safe also for v.dim()==0, since then
      // v[e()] will never be called. Here one could replace
      // v[i] by v.cui(i) without danger.

   Z n()const { return iv_.n();}
      //. next
      // Returns the index next to the last valid one.
      // This allows to write loops over all components as
      // for (Z i=v.b();i<v.n();i++) ... v[i] ...;
      // Note that this is safe also for v.dim()==0, since then
      // v[e()] will never be called. Here one could replace
      // v[i] by v.cui(i) without danger.

   V<T> meet(IvZ const& iv)const;
      //: meet
      // Returns a vector which, when considered as a function is the
      // restriction of function *this to the domain iv_ & iv.

   V<T> join(IvZ const& iv)const;
      //: join
      // Returns a vector which, when considered as a function is the
```

```
   // extension of function *this to the domain iv_ | iv, where
   // all function values on iv\iv_ are T().

V<T> operator +(Z i)const{ return V<T>(iv_+i,p_,"");}
   //: operator +
   // Returns a vector res such that res.dom() is the shifted
   // domain dom()+i of *this and has the same components as
   // *this.

V<T> operator -(Z i)const{ return V<T>(iv_-i,p_,"");}
   //: operator -
   // Returns a vector res such that res.dom() is the shifted
   // domain dom()-i of *this and has the same components as
   // *this.

void set_(T const& t);
   //: set
   // non-constant function which sets all components of (*this)
   // equal to t

V<T> set(Z i, T const& t)const;
   //: set
   // Returns a vector that originates from *this by setting the
   // value of component i.
   // Regular behaviour:
   // if we say
   // Z i=...;
   // T t= ...;
   // V<T> v=...;
   // v=v.set(i,t);
   // then the i-th component (see eli() ) of v will get the value t
   // unless i<1.
   // If i is a value for which (*this)[i-1] is not yet defined,
   // the vector becomes enlarged to the necessary size and all
   // not specified components initialized with the default
   // constructor of T

T in_(T const& t, bool reversed=false);
   //. insert
   // This operations treats *this as a shift register:
   // All components get shifted by one position 'to the right' and
   // the total length (dim) of the vector remains the same. So what
   // was the last component prior to the operation has to be removed
   // from the vector, in order to not wasting information, this
   // removed component will shine up as the return value of the
   // function. After the operation, the first component of the vector
   // will be t.
   // If reversed==true, t gets inputted at the end, and all shift
   // operations go 'to the left'.
```

```
   Z locAsc(T const& t)const;
      //: locate ascending
      // Here it is assumed that *this is ascendingly strictly ordered:
      // If sz_>=2 we have p_[i]<p_[i+1] for all i \in {0,sz_-2}.
      // For sz_<2 there is no condition.
      // For sz_==0 we stop with error.
      // For t<fir() we return b()-1
      // For t>=last() we return e()
      // If none of the previous conditions was met we necessarily have
      // sz_>=2 and we return the uniquely determined j such that
      //    p_[j]<=t<p_[j+1].
      // The possible results from this regular part of the functionality
      // thus are b(),...,e()-1.
      // Notice that the very similar function locate of Press et al.
      // only guarantees p_[j]<=t<=p_[j+1] for its return value j.

   X2<Z,bool> findAsc(T const& t)const
      //: find ascending
      // The second component of the return value says if t is
      // among the components of *this. Then the first component
      // of the return value gives the i for which (*this)[i]==t.
   {
      Z i=locAsc(t);
      return X2<Z,bool>(i,valInd(i) && t==cui(i));
   }

   X2<Z,bool> find(T const& t)const;
      //: find
      // The second component of the return value says if t is
      // among the components of *this. Then the first component
      // of the return value gives the lowest i for which (*this)[i]==t.
      // No ordering of *this is assumed.

// building new objects from given ones (generative methods)

// concatenating vectors and appending components. These operations have
// always O(sz_) complexity. The corresponding combined assignements are
// less direct to implement are not considered useful in the present
// context (they would not be more efficient than the friend versions,
// since new memory has to be allocated anyway).

   V<T> app(T const& t)const;
      // returns a vector which is obtained from *this by appending t
      // as the last component
      // notice also the prepend functions to be introduced
      // after the insert-functions (in order to have the inline
      // definition available).

   V<T> app(V<T> const& h)const;
```

```
      // returns a vector which is obtained from *this by appending h
      // at the back end

   void push_back(T const& t) { *this=app(t);}
      // for uniformity with STL

   V<T>& operator<<(T const& t) { return *this=app(t);}
      // appending an element in Ruby-style
      // Allows successive application as in
      // V<Word> w;
      // w<<"many"<<"words"<<"get"<<"easily"<<"stored";

   V<T>& operator<<(V<T> const& h) { return *this=app(h);}
      // appending an array in Ruby-style

   V<T>& operator&=(T const& t) { return *this=app(t);}
   V<T>& operator&=(V<T> const& h) { return *this=app(h);}
   V<T> operator&(T const& t)const { return app(t);}
   V<T> operator&(V<T> const& h)const { return app(h);}
      // appending in my favorite style

   V<IvZ> valOn(F<T,bool> const& f)const;
      //: valid on
      // retuns the array of those sub-intervals of the
      // whole indexing interval dom() on which the function
      // dom()-->bool, i|-->f((*this)[i]) yields true.
      // Notice that the result res \in V<IvZ> is a convenient
      // representation of a subset of dom(). Considering
      // *this as a function g: dom()-->T, then the function
      // h:=g&f is of type dom()-->bool and res, as a subset
      // of dom(), is h^-1({true}).

// resizing

   V<T> resize(Z newDim)const;
      // Returns a vector res such that res.dim()=newDim. If newDim is
      // smaller than dim(), the end of *this will be cut away. If newDim
      // is larger, T()'s will be added.
      // For newDim<0 the action is as if newDim==0.

   V<T> cut(Z i)const{ return resize(sz_-i);}
      // returned is a V<T> which results from *this by removing i
      // components from the end. For exotic values of i, see code
      // and explanation of resize.

// elimination and insertion

   V<T> eli(IvZ const& iv)const;
      // eli stands for eliminate
      // Eliminating all components which belong to iv. No exceptions
```

```
    // can happen! Universal and convenient of elimination. All other
    // forms are superfluous. Moreover, they are to be considered as
    // obsolete.

 V<T> eli(Z i, Z nEli=1)const{ return eli(IvZ(nEli,i,0));}
    // eli stands for eliminate
    // returned is a vector which is obtained from *this by
    // eliminating the i-th component and the nEli-1 following ones
    // (thus nEli components are removed) and shifting all later
    // components (if there are such components left) 'to the left' to
    // close the gap.

// V<T> eliFirst(Z nEli=1)const { return eli(1,nEli);}
 V<T> eliFirst(Z nEli=1)const { return eli(IvZ(nEli,b(),0));}
    // returned is a vector which is obtained from *this by
    // eliminating the nEli first components. If no
    // argument is provided, actually the first component
    // gets eliminated. Notice that in the three-argument constructor
    // IvZ(i,j,k) the first argument is the cardinality, the second
    // argument is the first element, and the third argument is dummy.

// V<T> eliLast()const { return eli(sz_,1);}
 V<T> eliLast()const { return eli(IvZ(e(),e()));}
    // returned is a vector which is obtained from *this by
    // eliminating the last component.

 V<T> eli1(V<T>& h, Z i)const;
    // We return a vector which results from *this by eliminating
    // h.dim() components starting at the i'th component (for i<1,
    // i==1 is understood). After the call h will hold the
    // eliminated components in due order (and no more - even if h was
    // longer before).
    // The function name ends in '1' to indicate that the first
    // argument is a non-constant reference, used for communication
    // of a part of the result.

 // condensation (contracting equivalent components into one)

 X2< V<T>, V<Z> > condense( bool (*equi)(const T&, const T&))const;
    // given an equivalence relation equi on T (t1~t2 <==>
    // equi(t1,t2)==true )
    // we divide the components of *this into equivalence classes.
    // Let the resturned pair be written as (res1,res2) . Then
    // (i) (*this)[i]~res1[res2[i]]
    // (ii) there is a j such that  (*this)[j]==res1[res2[i]]
    // Actually, the construction is made such that this j is the
    // smallest j for which (*this)[j]~res1[res2[i]].

 V<T> select(V<bool> const& s)const;
    // returned is a list which is obtained from *this by eliminating
```

```
   // all components (*this)[i] for which s[i] is defined and
   // statisfies s[i]==false. Beside of this removal, the order of the
   // components in *this is retained. So if s is defined
   // by a condition s[i]=Condition((*this)[i]), for the vector
   // component, this condition has to express a property we like to
   // have fulfilled for the result-vector of the select-operation.
   // 's expresses the favorable condition' and  n o t  the one to be
   // eliminated. Notice, that the select operation makes sense for
   // all values of s.dim().

V<T> ins(Z i, T const& t)const;
   //: insert
   // returned is a vector which is obtained from *this by
   // inserting t as the i-th component and shifting all later
   // components 'to the right' to give room for t.
   // The phrase 'i-th component' refers to natural counting (thus
   // p_[i-1] is the i-th component of *this).

V<T> ins(Z i, V<T> const& h)const ;
   // returned is a vector which is obtained from *this by
   // inserting h as the i-th and following component,
   // and shifting all later components of *this
   // 'to the right' to give room for h.
   // The phrase 'i-th component' refers to natural counting (thus
   // p_[i-1] is the i-th component of *this).

V<T> prepend(T const& t)const
   // returns a vector which is obtained from *this by appending t
   // as the first component
   { return ins(1,t);}

V<T> prepend(V<T> const& h)const
   // returns a vector which is obtained from *this by appending h
   // at the front end
   { return ins(1,h);}

V<T> rev()const;
   //: reversed
   // returned is the reversed vector (indexing in the opposite
   // direction)

V<T>& rev_(){ return *this = rev();}
   //: reversed
   // changes *this into a reversed version of itndexing in the
   // opposite
   // direction)

// re-indexing

V<T> rot(Z s, Outside mode=CYCLIC)const;
```

```
      //: rotate
      // Name as the list transformation function Rotate of Mathematica.
      // v.rot(s)[i] == v(i-s,mode)
      // This means that we shift the component i of the original vector
      // into the new position i+s

   void rot_(Z s, Outside mode=CYCLIC){ *this=rot(s,mode);}
      //: rotate
      // Same as rot, but as a mutating operation.

   V<T> compose(V<Z> const& w)const;
      //:: compose
      // v.compose(w)[i] = v[w[i]]

   V<T> permute(V<Z> const& w)const{ return compose(w);}
      //:: permute

   // transforming components

   template <class Y>
   V<Y> operator()(F<T,Y> const& f)const { return V<Y>(iv_,fnc()&f);}
      // generating arrays of different type by a type changing function

   V<T> operator()(T (*f)(T const&))const{ return fa(f);}
      // generating arrays of the same type with components f(c) instead of
      // c.

   V<T> operator()(T (*f)(T))const{ return faa(f);}
      // generating arrays of the same type with components f(c) instead of
      // c.

// defining generative laws and mutating laws for various expressions by
// function pointers. The idea behind is, that for T which allows
// particular operations, we can define those without using iteration via
// a operator[] since these all are defined directly in terms of
// pointers.
// See implementation of +=, -=, ... in Va<T> how this works

   T fAcc1(T (*f)(T const&), void (*acc)(T&, T const&))const;
      // returns an accumulated value of all values f(c), where c
      // are the components of *this. Accumulation is defined by
      // the argument acc:
      // T res=T(); 'for all components c' acc(res,c)

   T fAcc2(T (*f)(T const&, T const&), void (*acc)(T&, T const&),
      V<T> const& h )const ;
      // returns an accumulated value of all values f(c,ch), where c and
      // ch are the components of *this and h respectively. Accumulation
      // is defined by the argument acc:
      // T res; 'for all components c' acc(res,c)
```

```
   // useful in defining scalar products

template <class Y>
Y fAcc3(Y (*f)(T const&, T const&), void (*acc)(Y&, Y const&),
   V<T> const& h )const ;
   // returns an accumulated value of all values f(c,ch), where c and
   // ch are the components of *this and h respectively. Accumulation
   // is defined by the argument acc

V<T> fAcc4(F<T2<T>,T> const& f)const;
   // The kind of accumulation needed for computing the forces in
   // particle systems with pair interaction. Here we assume that
   // forces and positions are of the same type, e.g. R2 or R3.
   // Let x[1],...x[n] be the components of *this. We first compute
   // the incomplete matrix f(x[i],x[j]) i=1,...n, j<i and complete it
   // assuming f(x[i],x[j]) = - f(x[j],x[i]). In application mentioned
   // earlier this is the collection of mutually forces. The total force
   // on particle i is sum over j of f(x[i],x[j]). It is the i-th
   // component of the V<T>-typed return value of the function.
   // The force type T thus needs to provide operations unary - and +=.

V<T> fAcc5(F<R,R> const& dPot)const;
   // The kind of accumulation needed for computing the forces in
   // particle systems with pair interaction derived from a distance-
   // dependent (radial symmetric) potential. The argument dPot is
   // the derivative with respect to r of the r-dependent radially
   // symmetric pair potential.

V<T> fAcc6(F<R,R> const& dPot)const;
   // forces from a space-dependent potential

V<T> fa(T (*f)(T const&))const ;
   // returns a vector defined by replacing the components c of *this
   // by f(c)

V<T> faa(T (*f)(T))const ;
   // returns a vector defined by replacing the components c of *this
   // by f(c)

V<T> fb(T (*f)(T const&, T const&), T const& t)const;
   // returns a vector defined by replacing the components c of *this
   // by f(c,t)

template <class Y>
V<T> fb2(T (*f)(T const&, Y const&), Y const& t)const;
   // returns a vector defined by replacing the components c of *this
   // by f(c,t)

template <class Y>
void fb2_(T (*f)(T const&, Y const&), Y const& t);
```

```
   // replaces *this by a vector defined by replacing the components
   // c of *this by f(c,t)

void fc_(T (*f)(T const&, T const&), T const& t) ;
   // replaces *this by a vector defined by replacing the components c
   // of *this by f(c,t)

void fd(T (*f)(T const&, T const&), T& t)const;
   // replaces t by f(c,t) for each component c
   // If, for instance, f(c,t)=t+c*c we replace t by t+sum of c*c

V<T> fe(T (*f)(T const&, T const&), V<T> const& h)const;
   // returns a vector defined by replacing the components c of *this
   // by f(c,c') where c' are the components of h.
   // Error if dim()!=h.dim()

template <class Y>
V<T> fet(T (*f)(T const&, Y const&), V<Y> const& h)const;
   // returns a vector defined by replacing the components c of *this
   // by f(c,c') where c' are the Y-typed components of h.
   // Error if dim()!=h.dim(). Template version of fe. Unfortunately
   // h.p_ is not accessible in the implementation of this function.
   // This enforced introducing the non-canonical access function rep()
   // in 2014-01-23.

V<T> fe2(T (*f)(T const&, T const&), V<T> const& h)const;
   // Very similar to fe. However, the dimension of the result
   // is the maximum of dim() and h.dim(). Non-existing components
   // of any of the operands are replaced by T().

void ff_(T (*f)(T const&, T const&), V<T> const& h, Z i=0);
   // Replaces the components of *this starting from (*this)[i]
   // by f(c,c') where c' are the components of h. Thus for i=0
   // and h.dim()>=sz_ the whole array *this is affected.

void fg_(T (*f)(T const&, T const&, T const&),
   V<T> const& h, T const& t);
   // replaces the components c of *this
   // by f(c,c',t) where c' are the components of h

void fa_(F<T,T> const& f, IvZ iv);
   // replaces the components c of *this with index in iv
   // by f(c); modern form of fa.

void fh_(T const& w1, T const& w2, T const& w3, Outside mode);
   // replaces the component c[i] of *this by
   // w1*read(i-1,mode)+w2*read(i,mode)+w3*read(i+1,mode)
   // Thus makes use of multiplication in T.

template <class Y>
```

```
   void fht_(Y const& w1, Y const& w2, Y const& w3, Outside mode);
      // replaces the component c[i] of *this by
      // read(i-1,mode)*w1+read(i,mode)*w2+read(i+1,mode)*w3
      // Thus makes use of multiplication in T.

   template <class Y>
   V<T> fh(T (*f)(T const&, T const&, T const&, Y const&, Z),
      Y const&, Outside mode)const;
      // Returns a vector in which the component c[i] is replaced
      // by f(c[i-1],c[i],c[i+1],y,i),where the i-/+1 are understood
      // according to mode. The quantity y helps to provide parameters
      // required by a concrete situation. Worked for implementing
      // the Hamiltonian corresponding to Dirac's relativistic wave
      // equation.

   template <class Y>
   V<Y> fi(Y (*f)(T const&, T const&), void (*acc)(Y&, Y const&))const;
      // Returns a vector with Y-valued components y[i] which are obtained
      // by accumulating the terms f((*this)[i],(*this)[j]).
      // Accumulation is defined by the argument acc.

   template <class Y>
   V<Y> fj(Y (*f)(T const&, T const&), void (*acc)(Y&, Y const&))const;
      // Returns a vector with Y-valued components y[i] which are obtained
      // by accumulating the terms f((*this)[i],(*this)[j]).
      // Accumulation is defined by the argument acc.
      // We assume f((*this)[i],(*this)[j]) == - f((*this)[j],(*this)[i])
      // and use this for doing only one evaluation of f
      // for the two pairs (i,j) and (j,i) and no evaluation
      // of f for any pair (i,i).
      // We assume that for any Y-object y, -y is defined.

// diagnostics
   void show(ostream& out)const ;
      // diagnostic messages;

     T *const rep()const{ return p_;} // 20014-01-24
      // This is needed in the implementation of V<T>::fet<Y>. Here we need
      // direct access to the data member p_ of a function argument of type
      // V<Y>. Unfortunately (and unexpectedly) what is OK for V<T> does
      // not work for V<Y>. This function conflicts with my ambition to
      // ban pointers from the public interface of C+- classes.

      Z getCount()const
      {
         Z res=0;
#if defined(CPM_USECOUNT)
         res=u_.getCount();
#endif
         return res;
```

```
      }

protected:
   void cow_(void);
      //. copy on write
      // The first '*this-changing' statement in the
      // body of a non-constant member function has to be
      //   cow_();

private:
// member functions
   V(IvZ iv, T* p1, Word dummy):iv_(iv){ini_(p1);dummy;}
      // p1 has to be created with new; since this is prone to misuse
      // this function has to be private.

   V(Z n, Z first, T* p1, Word dummy):iv_(n,first,137){ini_(p1);dummy;}
      // similar to previous function

   T* copy()const;
      //: copy
      // returns a pointer to a storage area that contains
      // a copy of the components of *this

// data members

#if defined(CPM_USECOUNT)
   UseCount u_;
#endif
   // u_ becomes initialized by the default constructor and thus gets
   // a count 1, saying that the piece of free store pointed to by p_
   // is in use by *this

// static data
   static const T def_;
      // allows read function to return references
// static functions
   static T fCyc(Z const& i, V<T> const& v) { return v(i,CYCLIC);}
   static T fCon(Z const& i, V<T> const& v) { return v(i,CONSTANT);}
   static T fDef(Z const& i, V<T> const& v) { return v(i,DEFAULT);}
   static size_t sit(Z i) { return static_cast<size_t>(i);}
      // size type
      // Instead of new T[i], I now write new T[sit(i)] so that
      // allocation is always fed with a parameter which fits the system
      // (in 64 bit systems, size_t may be 64 bit wide). size_t is known
      // due to inclusion of <cpmfl.h> and it is assumed to take the
      // bit-width of the machine properly into account.
   void ini_(){sz_=iv_.car();p_=new T[sit(sz_)];}
   void ini_(T *p){sz_=iv_.car();p_=p;}

protected:
```

```
// data members
   // These should be accessible to derived classes for enabling fast
   // component operations.
   IvZ iv_;
      // Set of valid indexes. Since (*this) can be considered a
      // function iv_ --> T, this object is also called the domain
      // of *this.

   Z sz_;
      // number of valid indexes (depends on iv_, equals iv_.car())

   T* p_;
      // pointer to beginning of the array.
};

//////// using V for some special template arguments//////////////////

V<Z> IvZtoVofZ(IvZ const& iv);
   //: Iv to V of Z
   // returns the Z's that make up the interval iv
   // as the components of an ordered array (increasing
   // order, of course)

V<Z> VofIvZtoVofZ(V<IvZ> const& viv);
   //: V of IvZ to V of Z
   // appends the results from applying the previous functions
   // to the viv[i] according to the obvious code
   //  { V<Z> res(0); Z n=viv.dim();
   //  for (Z i=0;i<n;++i) res&=IvZtoVofZ(viv[i]); return res;}

V<Word> comLine(int argc, char* argv[]);
   //: command line
   // converts the C-traditional command-line argument into a C+- array.

////////////////////// Implementation ////////////////////////////////
template <class T>
const T V<T>::def_=T();

template <class T>
inline   V<T>::~V(){
#if defined(CPM_USECOUNT)
   if (u_.only()){ /*cout<<"~V called"<<endl;*/delete[] p_;}
#else
   /*cout<<"~V called"<<endl;*/ delete[] p_;
#endif
}

template <class T>
Word V<T>::nameOf()const{
   Word nt=Root<T>(T()).nameOf();
```

```
   Word wi="V<";
   return wi&nt&">";
}

template <class T>
R V<T>::dis(V<T> const& h)const
{
   R res=0.;
   if (iv_!=h.iv_) return R(1);
   for (Z i=b();i<=e();++i){
      res+=Root<T>(cui(i)).dis(h.cui(i));
   }
   return res;
}

template <class T>
inline T const& V<T>::operator[](Z i)const
{
   if (ranChc){
      if (!iv_.hasElm(i)){
         cpmerror(nameOf()&
         "::operator[]const: read-index out of  range: i= "&
         cpm(i)&" iMin= "&cpm(iv_.b())&" iMax= "&cpm(iv_.e()));
      }
   }
   if (signal) cout<<" const [] called"<<endl;
   return p_[iv_.ri(i)];
}


template <class T>
inline T& V<T>::operator[](Z i)
{
   if (ranChc){
      if (!iv_.hasElm(i)){
         cpmerror(nameOf()&"::operator[]: write-index out of  range: i= "&
         cpm(i)&" iMin= "&cpm(iv_.b())&" iMax= "&cpm(iv_.e()));
      }
   }
   if (signal) cout<<" mutating [] called"<<endl;
#if defined(CPM_USECOUNT)
   cow_();  // copy on write since operator is not const
      // may change u_ and p_!
#endif
   return p_[iv_.ri(i)];
  //return p_ + iv_.ri(i); // Trying to return T& directly, i. e. without
  // a type conversion from T. Does not work!
}

// using intentionally previously defined []-indexing
```

```
// for common messaging and safeness.
template <class T>
T const& V<T>::fir()const{ return (*this)[iv_.b()];}

template <class T>
T& V<T>::fir(){ return (*this)[iv_.b()];}

template <class T>
T const& V<T>::last()const{ return (*this)[iv_.e()];}

template <class T>
T& V<T>::last(){ return (*this)[iv_.e()];}

template <class T>
inline T const& V<T>::cui(Z i)const
{
   if (ranChcAlw){
      if (!iv_.hasElm(i)) cpmerror("index out of range");
   }
   return p_[iv_.ri(i)];
}

template <class T>
inline T& V<T>::cui(Z i)
{
   if (ranChcAlw){
      if (!iv_.hasElm(i)) cpmerror("index out of range");
   }
#if defined(CPM_USECOUNT) // don't call a function if not needed
   // even if its implementation is trivial
   cow_();
#endif
   return p_[iv_.ri(i)];
}

template <class T>
T const& V<T>::cyc(Z i)const
{
   return (*this)[iv_.cyc(i)];
}

template <class T>
T& V<T>::cyc(Z i)
{
#if defined(CPM_USECOUNT)
   cow_();
#endif
   return (*this)[iv_.cyc(i)];
}
```

```
template <class T>
T const& V<T>::con(Z i)const
{
   if (sz_==0) return def_;
   return (*this)[iv_.con(i)];
}

template <class T>
T& V<T>::con(Z i)
{
   if (sz_==0){
      cpmerror("V<T>::con(Z i): i="&cpm(i)&
         " is no valid index in void array");
      return p_[0]; // never happens
   }
   cow_();
   return (*this)[iv_.con(i)];
}

template <class T>
T const& V<T>::read(Z i, Outside mode)const
{
   if (iv_.hasElm(i)) return p_[iv_.ri(i)];
   else{
      if (mode==DEFAULT) return def_; // reference to it can be returned
      else if (mode==CYCLIC) return cyc(i);
      else return con(i);
   }
}

template <class T>
T* V<T>::copy()const
{
 //  cout<<" copy() called"<<endl;
   T* itp=p_;
   T* q=new T[sit(sz_)];
   T* it=q;
   Z i=sz_;
   while (i--) *it++=*itp++;
   return q;
}

template <class T>
T V<T>::operator()(Z i, Outside mode)const
{
   if (sz_==0) return T();
   if (iv_.hasElm(i)) return p_[iv_.ri(i)];
   if (mode==DEFAULT) return T();
   else if (mode==CYCLIC) return cyc(i);
   else return con(i);
```

```
}

template <class T>
F<Z,T> V<T>::fnc(Outside mode)const
{
#ifdef CPM_Fn
   if (mode==DEFAULT) return F1<Z,V<T>,T>(*this)(fDef);
   else if (mode==CYCLIC) return F1<Z,V<T>,T>(*this)(fCyc);
   else return F1<Z,V<T>,T>(*this)(fCon);
#else
   if (mode==DEFAULT) return F<Z,T>(bind(fDef,_1,*this));
   else if (mode==CYCLIC) return F<Z,T>(bind(fCyc,_1,*this));
   else return F<Z,T>(bind(fCon,_1,*this));
#endif
}

template <class T>
void V<T>::cow_(void)
   // body is void if CPM_USECOUNT is not defined
{
#if defined(CPM_USECOUNT)
   if (u_.makeOnly()){
      p_= (sz_==0 ? 0 : copy());
      u_.startNew_();
   }
#endif
}

// constructors
// 137 is a dummy argument

template <class T>
V<T>::V(Z n):iv_(safeDim(n),1,137)
{
   ini_();
   T* q=p_;
   for (Z i=0;i<sz_;++i) *q++ = T();
}

template <class T>
V<T>::V(Z n, Begin bg):iv_(safeDim(n),0,137)
{
   ini_();
   T* q=p_;
   for (Z i=0;i<sz_;++i) *q++ = T();
}

template <class T>
V<T>::V(Z n, T const& t, Begin bg):iv_(safeDim(n),0,137)
{
```

```
   ini_();
   T* q=p_;
   for (Z i=0;i<sz_;++i) *q++ = t;
}

template <class T>
V<T>::V(Z n, T const& t, Z first):iv_(safeDim(n),first,137)
{
   ini_();
   T* q=p_;
   for (Z i=0;i<sz_;++i) *q++ = t;
}

template <class T>
V<T>::V(Z n, F<Z,T> const& f):iv_(safeDim(n),1,137)
{
   ini_();
   T* q=p_;
   Z j=iv_.b();
   for (Z i=0;i<sz_;++i) *q++ = f(j++);
}

template <class T>
V<T>::V(IvZ const& iv):iv_(iv)
{
   ini_();
   T* q=p_;
   for (Z i=0;i<sz_;++i) *q++ = T();
}

template <class T>
V<T>::V(IvZ const& iv, T const& t):iv_(iv)
{
   ini_();
   T* q=p_;
   for (Z i=0;i<sz_;++i) *q++ = t;
}

template <class T>
V<T>::V(IvZ const& iv, F<Z,T> const& f):iv_(iv)
{
   ini_();
   T* q=p_;
   for (Z i=0;i<sz_;++i) *q++ = f(i+iv_.b());
}

template <class T>
V<T>::V(std::initializer_list<T> il ):
iv_((Z)il.size(),1,137) // 137 is dummy argument
{
```

```
   ini_();
   std::uninitialized_copy(il.begin(),il.end(),p_);
      // see BS4, p. 498
}

template <class T>
V<T>::V(std::vector<T> const& v, Z first):
iv_((Z)v.size(),first,137)
{
   ini_();
   typename std::vector<T>::const_iterator i;
   T* q=p_;
   for (i=v.begin();i!=v.end();++i) *q++=*i;
}

template <class T>
std::vector<T> V<T>::std()const
{
   std::vector<T> res(sz_);
   typename std::vector<T>::iterator i;
   T* q=p_;
   for (i=res.begin();i!=res.end();++i) *i=*q++;
   return res;
}

template <class T>
#if defined(CPM_USECOUNT)
   V<T>::V(V<T> const& h) :iv_(h.iv_),sz_(h.sz_),u_(h.u_),p_(h.p_){}
#else
   V<T>::V(V<T> const& h) :iv_(h.iv_),sz_(h.sz_),p_(h.copy()){}
#endif

template <class T>
V<T>& V<T>::operator=(V<T> const& h)
{
   if (sz_==0 && h.sz_==0) return *this;
      // added 2002-03-07, should be OK
   if (this==&h) return *this;
#if !defined(CPM_USECOUNT)
   Z szMem=sz_;  // added 2000-11-17
   // definition restricted to the case !defined(CPM_USECOUNT)
   // 2005-06-22 to avoid waring on non use initialized
   // variable
#endif
   iv_=h.iv_;
   sz_=h.sz_;
#if defined(CPM_USECOUNT)
   if (u_.reattach_(h.u_)) delete[] p_;
   p_=h.p_;
#else
```

```
   if (sz_!=szMem){ // added 2000-11-17 in order to avoid
      // superfluous delete, new action
      delete[] p_;
      p_=new T[sit(sz_)];
   }
   T* q=h.p_;
   T* it=p_;
   Z i=sz_;
   while (i--) *it++ = *q++;
#endif
   return *this;
}

template <class T>
V<T> V<T>::meet(IvZ const& iv)const
{
   IvZ ivRes=iv_.meet(iv);
   V<T> res(ivRes); // all components are T()
   for (Z i=res.b();i<=res.e();++i){
       if (iv.ni(i)) res[i]=(*this)[i];
   }
   return res;
}

template <class T>
V<T> V<T>::join(IvZ const& iv)const
{
   IvZ ivRes=iv_.join(iv);
   V<T> res(ivRes); // all components are T()
   for (Z i=res.b();i<=res.e();++i){
       if (iv_.ni(i)) res[i]=(*this)[i];
   }
   return res;
}

template <class T>
X2<Z,bool> V<T>::find(T const& t)const
{
   Z i0=b();
   Z j=i0-1;
   for (Z i=i0; i<=e(); ++i){
      if (t==cui(i)){ // if this never happens we still have j == i0-1
         // hence j>=i0 signals success in finding t
         j=i;
         break;
      }
   }
   return X2<Z,bool>(j,j>=i0);
}
```

```
// modified from function locate of Press et al.

template <class T>
Z V<T>::locAsc(T const& t)const
{
   if (sz_==0){
      cpmerror("V<T>::locAsc(T): array is void");
      return -137; // never happens
   }
   if (t<fir()) return b()-1;
   if (t>=last()) return e();
      // if sz_==1 we have fir()==last and then the two previous
      // conditions are an alternative: one of them holds and we
      // are ready. Thus now sz_>=2
   Z jl=0,ju=sz_;
   while (ju-jl>1){
      Z jm=(jl+ju)/2;
      if (t >= p_[jm]) jl=jm; else ju=jm; // Press et al. have > here
   }
   return jl+b();
}

template <class T>
V<T> V<T>::app(V<T> const& h)const
{
   IvZ iv2=iv_.app(h.iv_);
   Z i,sz2=iv2.car();
   T* p2=new T[sit(sz2)];
   T* it=p2;
   T* q1=p_;
   i=sz_;
   while(i--) *it++ = *q1++;
   T* q2=h.p_;
   i=h.sz_;
   while (i--) *it++ = *q2++;
   return V<T>(iv2,p2,Word());
}

template <class T>
V<T> V<T>::app(T const& t)const
{
   IvZ iv2=iv_.app(IvZ(Z(1),Z(1)));
   Z i,sz2=iv2.car();
   T* p2=new T[sit(sz2)];
   T* it=p2;
   T* q=p_;
   i=sz_;
   while(i--) *it++ = *q++;
   *it++ = t;
   return V<T>(iv2,p2,Word());
```

```
}

template <class T>
V<T> V<T>::rev()const
{
   if (sz_<2){
      return *this;
      // nothing to do if we have no or one component
   }
   else{
      T* p2=new T[sit(sz_)];
      T* it=p2;
      T* itp=p_;
      itp+=(sz_-1);
      Z i=sz_;
      while (i--)  *it++ = *itp--;
      return V<T>(iv_,p2,Word());
   }
}

template <class T>
V<T> V<T>::resize(Z newDim)const
{
   Z n2= newDim<0 ? 0 : newDim;
   Z nCopy=(n2<=sz_ ? n2 : sz_);
   V<T> res(IvZ(n2,b(),137)); // correctly initialized even for n2>sz_
   T* it=res.p_;
   T* itp=p_;
   Z i=nCopy;
   while (i--) *it++ = *itp++;
   return res;
}

template <class T>
V<T> V<T>::eli(IvZ const& iv)const
{
   IvZ ie=iv&dom();
   if (ie.isVoid()) return *this ;
   else{
      V<bool> vb(dom());
      for (Z i=vb.b();i<=vb.e();++i){
         vb.cui(i)=!ie.hasElm(i);
      }
      return select(vb);
   }
    //return eli(i[1],i.car());
}

template <class T>
V<T> V<T>::eli1(V<T>& h, Z i)const
```

```
{
// Here we use natural counting of components so that
// the j-th component of h is h[j-1].
   Z nEli=h.dim();
   if (i<1) i=1;
   if (sz_==0){ // nothing eliminated, nothing left
      h=V<T>(0);
      return V<T>(0);
   }
   else if (i>sz_){ // nothing eliminated
      h=V<T>(0);
      return *this;
   }
   else{ // now i>=1 and i<=sz_ and sz_>=1
      // if true, we have sz_>=1
   // the i-th component is the first to be eliminated
   // so we have i-1 components of *this which are on the left-hand
   // side of the elimination area. These have to shine up in
   // the result vector to be returned
      Z nRes1=i-1;
   // So the maximum number of
   // components of *this that run the risk to get eliminated is
   // sz_-nRes1
      Z nEliRisk=sz_-nRes1;
      if (nEli>nEliRisk) nEli=nEliRisk;
      Z nRes2=sz_-nRes1-nEli;
      Z nRes=nRes1+nRes2;
      T* pRes=new T[sit(nRes)];
      T* pEli=new T[sit(nEli)];
      T* itRes=pRes;
      T* itEli=pEli;
      T* itp=p_;
      Z j=nRes1;
      while (j--) *itRes++=*itp++;
      j=nEli;
      while (j--) *itEli++=*itp++;
      j=nRes2;
      while (j--) *itRes++=*itp++; // for j==0 nothing done
      h=V<T>(nEli,pEli);
      return V<T>(nRes,pRes);
   }
}

/************************
template <class T>
V<T> V<T>::eli(Z i, Z nEli)const
{
   Word loc("V<T>::eli(Z,Z))");
   if (sz_<1 || i>sz_ || i<1 || nEli<1){
      cpmwarning(loc&": component to be eliminated does not exist");
```

```
      return *this;
   } // now sz_>=1
   Z remaining=1+sz_-i;   // number of component i and  followers
      // is >=1 due to assert()
   Z nEliActual=( nEli<=remaining ? nEli : remaining);
      // this is the number of components which we actually will be
      // removing
   Z j, sz1=sz_-nEliActual;
   if (sz1==0) return V<T>();
   else {
      T* p1=new T[sit(sz1)];
      T* it=p1;
      T* q1=p_;
      for (j=1;j<=i-1;j++) *it++ = *q1++; // the j is a counting device
         // only, no components are taken, so starting from 1 is no
         // mistake; it is a convenience.
      for (j=1;j<=nEliActual;j++) q1++;   // advancing over the elements
      // to be removed
      for (j=i;j<=sz1;j++) *it++ = *q1++;
      return V<T>(sz1,b(),p1,Word());
   }
}
*************************/

template <class T>
V<T> V<T>::ins(Z ia, V<T> const& h)const
{
   Z mL=3;
   Word loc("V<T> V<T>::ins(Z i, V<T> const& h)const");
   CPM_MA
   Z first=b();
   Z i=ia-first+1;
   if (i<1 || i>(sz_+1)) cpmerror(
      "V<T>::ins(Z i,V<T>): unvalid argument i="&cpmwrite(i));
   Z j, sz1=sz_+h.sz_;
   T* p1=new T[sit(sz1)];
   T* it=p1;
   T* q1=p_;
   T* q2=h.p_;
   for (j=1;j<=i-1;j++)   *it++ = *q1++;
   for (j=1;j<=h.sz_;j++)  *it++ = *q2++; // h.sz_ terms
   for (j=i;j<=sz_;j++)    *it++ = *q1++;
   CPM_MZ
   return V<T>(sz1,first,p1,Word());
}

template <class T>
V<T> V<T>::ins(Z ia, T const& t)const
{
   Z mL=3;
```

```
   Word loc("V<T> V<T>::ins(Z i, T const& t)const");
   CPM_MA
   Z first=b();
   Z i=ia-first+1;
   if (i<1 || i>(sz_+1))
      cpmerror("V<T>::ins(Z i,T): unvalid argument i="&cpmwrite(i));
   Z j, sz1=sz_+1;
   T* p1=new T[sit(sz1)];
   T* itp=p_;
   T* it=p1;
   for (j=1;j<=i-1;j++) *it++ = *itp++;
   *it++ =t;
   for (j=i;j<=sz_;j++)  *it++ = *itp++;
   CPM_MZ
   return V<T>(sz1,first,p1,Word());
}

template <class T>
T V<T>::in_(T const& t, bool reversed)
// safe logic by indexing, propably not utmost performance
{
   if (sz_==0) return T();
   cow_();
   Z i;
   T res;
   if (!reversed){
      res=p_[sz_-1]; // last component of array
      for (i=sz_-1;i>0;i--){
         p_[i]=p_[i-1]; // causal processing: on the right-hand side we
         // evaluate only expressions which were not yet changed during
         // the loop
      }
      p_[0]=t;
   }
   else{
      res=p_[0]; // first component of array
      for (i=0;i<sz_-1;i++){
         p_[i]=p_[i+1]; // causal processing: on the right-hand side we
         // evaluate only expressions which were not yet changed during
         // the loop
      }
      p_[sz_-1]=t;
   }
   return res;
}

// re-indexing

template <class T>
V<T> V<T>::compose(V<Z> const& w)const
```

```
{
   V<T> res(iv_);
   for (Z i=w.b();i<=w.e();i++){
     res[i]=operator()(w[i]);
   }
   return res;
}

template <class T>
V<T> V<T>::rot(Z s, Outside mode)const
{
   V<T> res(iv_);
   for (Z i=b();i<=e();++i) res[i]=operator()(i-s,mode);
   return res;
}

template <class T>
X2< V<T>, V<Z> > V<T>::condense(
   bool (*equiv)(const T&, const T&))const
// implementation based on function eclazz of the Numerical Receipes of
// Press et al.
{
   const Z mL=3;
   static Word loc("V<T>::condense()");
   CPM_MA
   Z n=dim();
   if (n<1){ // addition 2002-02-23
      CPM_MZ
      return X2< V<T>, V<Z> >(V<T>(0),V<Z>(0));
   }
   V<Z> res2(n);
   Z k,j;
   res2[1]=1;
   for (j=2;j<=n;j++) {
      res2[j]=j;
      for (k=1;k<=(j-1);k++) {
         res2[k]=res2[res2[k]];
         if ((*equiv)((*this)[j],(*this)[k])) res2[res2[res2[k]]]=j;
      }
   }
   for (j=1;j<=n;j++) res2[j]=res2[res2[j]];
   Z m=-1;
   Z rj;
   for (j=1;j<=n;j++){
      rj=res2[j];
      if (rj>m) m=rj;
   }
   V<Z> aux(m,0); // unfortunately the NR algorithm does not guarantee
      // that there are no gaps between the valid values of rj. Therfore
      // we find out the valid ones by an additional loop
```

```
   V<Z> found(m,0);
   for (j=1;j<=n;j++){
      rj=res2[j];
      if(found[rj]==0){
         found[rj]=1;
         aux[rj]=j;
      }
   }
   Z mAct=0;
   for (j=1;j<=m;j++) mAct+=found[j];
   V<T> res1(mAct);
   Z jAct=1;
   for (j=1;j<=m;j++){ // notice that aux[j] was initialized as 0
      if (aux[j]>0) res1[jAct++]=(*this)[aux[j]];
   }
   CPM_MZ
   return X2< V<T>,V<Z> >(res1,res2);
}


template <class T>
void V<T>::set_(T const& t)
{
   cow_();
   Z i=sz_;
   T* it=p_;
   while(i--) *it++ = t;
}

template <class T>
void V<T>::show(ostream& out)const
{
   out<<endl<<"V<T>::show()";
#if defined(CPM_USECOUNT)
   out<<endl<<" usecount = "<<u_.getCount();
#endif
   out<<endl<<" start address = "<<p_;
   out<<endl<<" end address = "<<(p_+(sz_-1));
   long b=1+(long)((p_+(sz_-1))-p_);
   out<<endl<<" bytes="<<b;
}

template <class T>
V<T> V<T>::set(Z i, T const& t)const
{
   Z iC=i-1;
   // now iC is a 'C-pointer index'

   if (iC<0) return *this;   // nothing changed
   else if (iC<sz_){
```

```
      // vector can already hold the new value
      V<T> res(*this);
      res[iC]=t;
      return res;
   }
   else{
      // now we have to return an enlarged vector
      Z j, sz2=iC+1;
      V<T> res(sz2);
      res[iC]=t;
      T* it=res.p_;
      T* itp=p_;
      Z i=sz_;
      while (i--) *it++ = *itp++;
      return res;
   }
}

template <class T>
T V<T>::fAcc1(T (*f)(T const&), void (*acc)(T&, T const&))const
{
   T* p1=p_;
   T res=T();
   for (Z i=0;i<sz_;i++) acc(res,f(*p1++));
   return res;
}

template <class T>
T V<T>::fAcc2(T (*f)(T const&, T const&), void (*acc)(T&, T const&),
   V<T> const& h)const
{
   T* p1=p_;
   T* p2=h.p_;
   T res=T();
   for (Z i=0;i<sz_;i++) acc(res,f(*p1++,*p2++));
   return res;
}

template <class T>
template <class Y>
Y V<T>::fAcc3(Y (*f)(T const&, T const&), void (*acc)(Y&, Y const&),
      V<T> const& h )const
{
   T* p1=p_;
   T* p2=h.p_;
   Y res=Y();
   for (Z i=0;i<sz_;i++) acc(res,f(*p1++,*p2++));
   return res;
}
```

```
template <class T>
V<T> V<T>::fAcc4(F<T2<T>,T> const& f)const
{
   Z d=dim();
   V< V<T> > aux(d,V<T>(d));
   for (Z i=0;i<d;++i){
      for (Z j=0;j<i;++j){
         T fij=f(T2<T>(li(i),li(j)));
         aux.li(i).li(j)=fij;
         aux.li(j).li(i)=-fij;
      }
   }
   T* p1=new T[sit(sz_)];
   for (Z i=0;i<sz_;i++){
      T res;
      for (Z j=0;j<sz_;j++) res+=aux.li(i).li(j);
      p1[i]=res;
   }
   return V<T>(dom(),p1,"");
}

namespace{

template <class T>
T fFunc5(T2<T> const& xij, F<R,R> const& dPot)
// This is intended to represent the force which particle i feels due to
// particle j being present.
{
   T eij=xij[1]-xij[2]; // eij points from j to i. Thus a positive
   // multiple of eij corresponds to a force on i which drives it away
   // from particle j. This such is the case of a repulsive potential.
   // This has dPot(r)/dr < 0 so that with the sign built into the formula
   // for the return value we in fact have the case of the positive
   // multiple we started with.
   R r=eij.nor_();
   return eij*(-dPot(r));
}

template <class T>
T fFunc6(T const& xi, F<R,R> const& dPot)
{
   T ei=xi; // ei points from the origin to i. Same situation as in fFunc5.
   R r=ei.nor_();
   return ei*(-dPot(r));
}

}

template <class T>
V<T> V<T>::fAcc5(F<R,R> const& dPot)const
```

```
{
#ifdef CPM_Fn
   F< T2<T>, T > f = F1< T2<T>, F<R,R>, T >(dPot)(fFunc5<T>);
#else
   F<T2<T>,T> f((std::bind(fFunc5<T>,_1,dPot)));
#endif

   Z d=dim(); // one could call fAcc4 here at the cost of an additional
      // function call. Here we ask for 'utmost efficiency'.
   V< V<T> > aux(d,V<T>(d));
   for (Z i=0;i<d;++i){
      for (Z j=0;j<i;++j){
         T fij=f(T2<T>(li(i),li(j)));
         aux.li(i).li(j)=fij;
         aux.li(j).li(i)=-fij;
      }
   }
   T* p1=new T[sit(sz_)];
   for (Z i=0;i<sz_;i++){
      T res;
      for (Z j=0;j<sz_;j++) res+=aux.li(i).li(j);
      p1[i]=res;
   }
   return V<T>(dom(),p1,"");
}

template <class T>
V<T> V<T>::fAcc6(F<R,R> const& dPot)const
{
#ifdef CPM_Fn
   F<T,T> f = F1<T,F<R,R>,T>(dPot)(fFunc6<T>);
#else
   F<T,T> f(std::bind(fFunc6<T>,_1,dPot));
#endif
   T* p1=new T[sit(sz_)];
   for (Z i=0;i<sz_;++i){
      p1[i]=f(li(i));
   }
   return V<T>(dom(),p1,"");
}


template <class T>
V<T> V<T>::fa(T (*f)(T const&))const
{
   T* p1=new T[sit(sz_)];
   T* it=p1;
   T* itp=p_;
   Z i=sz_;
   while(i--) *it++ = f(*itp++);
```

```
   return V<T>(dom(),p1,"");
}

template <class T>
V<T> V<T>::faa(T (*f)(T))const
{
   T* p1=new T[sit(sz_)];
   T* it=p1;
   T* itp=p_;
   Z i=sz_;
   while(i--) *it++ = f(*itp++);
   return V<T>(dom(),p1,"");
}

template <class T>
V<T> V<T>::fb(T (*f)(T const&, T const&), T const& t)const
{
   T* p1=new T[sit(sz_)];
   T* it=p1;
   T* itp=p_;
   Z i=sz_;
   while(i--) *it++ = f(*itp++,t);
   return V<T>(dom(),p1,"");
}

template <class T>
template <class Y>
V<T> V<T>::fb2(T (*f)(T const&, Y const&), Y const& t)const
{
   T* p1=new T[sit(sz_)];
   T* it=p1;
   T* itp=p_;
   Z i=sz_;
   while(i--) *it++ = f(*itp++,t);
   return V<T>(dom(),p1,"");
}

template <class T>
template <class Y>
void V<T>::fb2_(T (*f)(T const&, Y const&), Y const& t)
{
   cow_();
   T val;
   T* itp=p_;
   Z i=sz_;
   while(i--){
      val=*itp;
      *itp++=f(val,t);
   }
}
```

```
template <class T>
void V<T>::fc_(T (*f)(T const&, T const&), T const& t)
{
   cow_();
   T val;
   T* itp=p_;
   Z i=sz_;
   while(i--){
      val=*itp;
      *itp++=f(val,t);
   }
}

template <class T>
void V<T>::fd(T (*f)(T const&, T const&), T& t)const
{
   T* itp=p_;
   Z i=sz_;
   while (i--) t=f(*itp++,t);
}

template <class T>
V<T> V<T>::fe(T (*f)(T const&, T const&), V<T> const& h)const
{
   if (!sameDom(h)) throw Error("V<T>::fe(): domain mismatch");
   T* p1=new T[sit(sz_)];
   T* it=p1;
   T* p2=p_;
   T* p3=h.p_;
   Z i=sz_;
   while (i--) *it++ = f(*p2++,*p3++);
   return V<T>(dom(),p1,"");
}

template <class T>
template <class Y>
V<T> V<T>::fet(T (*f)(T const&, Y const&), V<Y> const& h)const
{
   if (dom()!=h.dom()) throw Error("V<T>::fet(): domain mismatch");
   T* p1=new T[sit(sz_)];
   T* it=p1;
   T* p2=p_;
  // Y* p3=h.p_;
   Y* p3=h.rep();
   Z i=sz_;
   while (i--) *it++ = f(*p2++,*p3++);
   return V<T>(dom(),p1,"");
}
```

```
template <class T>
V<T> V<T>::fe2(T (*f)(T const&, T const&), V<T> const& h)const
{
   if (sz_==h.sz_) return fe(f,h);
   else if (sz_>h.sz_){
      T t0=T();
      T* p1=new T[sit(sz_)];
      T* it=p1;
      T* p2=p_;
      T* p3=h.p_;
      Z i=sz_;
      Z ih=h.sz_;
      while (ih--){
         i--;
         *it++ = f(*p2++,*p3++);
      }
      while(i--){
         *it++ = f(*p2++,t0);
      }
      return V<T>(dom(),p1,"");
   }
   else{
      T t0=T();
      T* p1=new T[sit(h.sz_)];
      T* it=p1;
      T* p2=p_;
      T* p3=h.p_;
      Z i=sz_;
      Z ih=h.sz_;
      while (i--){
         ih--;
         *it++ = f(*p2++,*p3++);
      }
      while(ih--){
         *it++ = f(t0,*p3++);
      }
      return V<T>(h.dom(),p1,"");
   }
}

template <class T>
void V<T>::ff_(T (*f)(T const&, T const&), V<T> const& h, Z is)
{
   if (is<0) is=0;
   Z sz1=sz_-is;
   Z sz2=h.sz_;
   Z szMin=( sz1<sz2 ? sz1 : sz2);
   if (szMin<1) return; // nothing to do
   cow_();
   T val;
```

```
   T* p2=h.p_;
   T* itp=p_+is;
   Z i=szMin;
   while (i--){
      val=*itp;
      *itp++=f(val,*p2++);
   }
}

template <class T>
void V<T>::fa_(F<T,T> const& f, IvZ iv)
{
   IvZ iva=dom()&iv;
   if (iva.isVoid()) return; // nothing to be done
   cow_();
   T val;
   T* itp=p_;
   itp+=iva.inf();
      // setting the iterator pointer to the right place
   Z i=iva.car(); // this is the number of requested loop
      // actions, logic OK as the simple case i=1 tells
   while(i--){
      val=*itp;
      *itp++=f(val);
   }
}

template <class T>
void V<T>::fg_(T (*f)(T const&, T const&, T const&),
      V<T> const& h, T const& t)
{
   Z szMin=( sz_<=h.sz_ ? sz_ : h.sz_);
   cow_();
   T val;
   T* p2=h.p_;
   T* itp=p_;
   Z i=szMin;
   while (i--){
      val=*itp;
      *itp++=f(val,*p2++,t);
   }
}

template <class T >
void V<T>::fh_(T const& w_1, T const& w0, T const& w1, Outside mode)
{
   if (sz_<2) return;
   cow_();
   T v_1=read(b()-1,mode);
   T v0=p_[0];
```

```
   T v1=p_[1];
   Z k=0;
   while (k<sz_){
      p_[k]=v_1*w_1+v0*w0+v1*w1;
      k++;
      v_1=v0;
      v0=v1;
      v1=(k < sz_-1 ? p_[k+1] : read(e()+1,mode));
   }
}

template <class T >
template <class Y>
void V<T>::fht_(Y const& w_1, Y const& w0, Y const& w1, Outside mode)
{
   if (sz_<2) return;
   cow_();
   T v_1=read(b()-1,mode);
   T v0=p_[0];
   T v1=p_[1];
   Z k=0;
   while (k<sz_){
      p_[k]=v_1*w_1+v0*w0+v1*w1;
      k++;
      v_1=v0;
      v0=v1;
      v1=(k < sz_-1 ? p_[k+1] : read(e()+1,mode));
   }
}

template <class T >
template <class Y>
V<T> V<T>::fh(T (*f)(T const&, T const&, T const&, Y const&, Z),
      Y const& y, Outside mode)const
{
   if (sz_<2) return *this;
   T* p1=new T[sit(sz_)];
   Z iL=sz_-1;
   p1[0]=f(read(b()-1,mode),p_[0],p_[1],y,b());
   p1[iL]=f(p_[iL-1],p_[iL],read(e()+1,mode),y,e());
      // don't use read-functions in the loop
   T* v_1=p_;
   T* v0=v_1+1;
   T* v1=v0+1;
   T* p2=p1+1; // used to fill in values of p1 without changing p1
   Z i=b()+1;
   while (i<e()) *p2++=f(*v_1++,*v0++,*v1++,y,i++);
   return V<T>(dom(),p1,"");
}
```

```
template <class T >
template <class Y>
V<Y> V<T>::fi(Y (*f)(T const&, T const&), void (*acc)(Y&, Y const&))const
{
   Z i,j;
   Y* q=new Y[sit(sz_)];
   for (i=0;i<sz_;i++){
      Y yi=Y();
      for (j=0;j<sz_;j++){
         acc(yi,f(p_[i],p_[j]));
      }
      q[i]=yi;
   }
   return V<Y>(dom(),q);
}

template <class T >
template <class Y>
V<Y> V<T>::fj(Y (*f)(T const&, T const&), void (*acc)(Y&, Y const&))const
{
   Z i,j;
   Y* q=new Y[sit(sz_)];
   for (i=0;i<sz_;i++) q[i]=Y();
   for (i=0;i<sz_;i++){
      for (j=0;j<i;j++){
         Y fij=f(p_[i],p_[j]);
         acc(q[i],fij);
         acc(q[j],-fij);
      }
   }
   return V<Y>(dom(),q);
}

template <class T >
V<T> V<T>::select(V<bool> const& s)const
{
   Z n=dim(),ns=s.dim();
   V<T> resPrel=*this; // the actual result may have less components
   T* itPrel=resPrel.p_;
   T* itp=p_;
   Z iAct=0;
   Z i=n;
   Z is=s.b();
   while (i--){
      bool keepIt= (is<=s.e() ? s[is++] : true);
      T tAct=*itp++;
      if (keepIt){
         *itPrel++=tAct;
         iAct++;
      }
```

```
   }
   if (iAct==n) return resPrel; // no element was purged
   else return resPrel.resize(iAct);
}

template <class T >
V<IvZ> V<T>::valOn(F<T,bool> const& f)const
{
   Z mL=3;
   static Word loc("V<T>::valOn(F<T,bool>)");
   CPM_MA
   V<IvZ> res;
   bool yetFoundTrue=false;
   Z firstTrue=0, firstFalseAfterTrue=0;
   for (Z i=b();i<=e();++i){
      bool val=f((*this)[i]);
      if (val){ // we found true
         if (!yetFoundTrue){ // then start a interval of validity
            yetFoundTrue=true;
            firstTrue=i;
         }
         else{ // normally nothing to do
            // but if we are at the end of the array the
            // last pending truth interval has to be
            // considered as finished and has to be added
            if (i==e()){
               firstFalseAfterTrue=n();
               IvZ ivAct(firstTrue,firstFalseAfterTrue-1);
               res&=ivAct; // appending
               // nothing else to do since we are finished
               CPM_MZ
               return res;
            }
         }
      }
      else{ // we found false
         if (yetFoundTrue){ // then i is the terminator
            // of a truth interval
            firstFalseAfterTrue=i;
            IvZ ivAct(firstTrue,firstFalseAfterTrue-1);
            res&=ivAct;
            yetFoundTrue=false;
         }
      }
   }
   CPM_MZ
   return res;
}

template <class T>
```

```
bool V<T>::prnOn(ostream& str)const
{
   Z mL=3;
   Word loc=nameOf()&"::prnOn(...)";
   CPM_MA
   cpmwt((nameOf()&" begin").str());
   Root<IvZ>(iv_).prnOn(str);
   T* q=p_;
   for (Z i=b();i<=e();i++){
      if (CpmRoot::wrtTit){
         Word wi("// i="); // added 2012-01-19
         // same as in S<>
         wi&=cpm(i);
         bool bi=wi.prnOn(str);
         cpmassert(bi==true,loc);
      }
      if (!Root<T>(*q++).prnOn(str)){
         cpmwarning("failed to write component indexed "&cpm(i));
         cpmwarning("index range is from "&cpm(b())&" to "&cpm(e()));
         CPM_MZ
         return false;
      }
   }
   cpmwt((nameOf()&" end").str());
      // for large sz_ it would be difficult to find the
      // end of the vector data if these are written to a file
      // and a human reader wants to inspect them
   CPM_MZ
   return true;
}

template <class T>
bool V<T>::scanFrom(istream& str)
{
   Z mL=3;
   Word loc=nameOf()&"::scanFrom(...)";
   CPM_MA
   Root<IvZ> ivIn;
   bool suc = ivIn.scanFrom(str);
   if (!suc){
      cpmwarning(loc&": can't read IvZ");
      CPM_MZ
      return false;
   }
   IvZ iv=ivIn();
   Z n=iv.car();
   cpmmessage(mL,"dimension read as "&cpm(n));
   if (n>dimMax) cpmwarning(loc&": n>dimMax");
   V<T> res(iv);
   Root<T> riIn;
```

```
   for (Z i=res.b();i<=res.e();i++){
      suc = riIn.scanFrom(str);
      if (!suc){
         Word mes="failed to read component indexed "&cpm(i)&
             " index range is from "&cpm(res.b())&" to "&cpm(res.e());
         cpmwarning(mes);
         CPM_MZ
         return false;
      }
      res.cui(i) = riIn();
   }
   *this=res;
   CPM_MZ
   return true;
}

template <class T >
Z V<T>::com(V<T> const& s)const
// short vectors < longer vectors
{
   Z d1=dim(), d2=s.dim();
   if (d1<d2) return 1;
   else if (d1>d2) return -1;
   else{
      T* q=p_;
      T* qs=s.p_;
      for (Z i=0;i<d1;i++){
         Z ci=Root<T>(*q++).com(*qs++);
         if (ci!=0) return ci;
      }
      return 0;
   }
}

/************************ iterated templates *************************/
// describing multi-indexed quantities
// Notice that these all have automatically the member functions of V
// defined!

#define CPM_V1 V<T>
#define CPM_V2 V<V<T> >
#define CPM_V3 V<V<V<T> > >
#define CPM_V4 V<V<V<V<T> > > >

/************************* class VV **********************************/
template <class T>
class VV: public CPM_V2{ // matrices

   typedef CPM_V2 Base;
```

```
public:

// constructors

   VV(Z d1, Z d2):CPM_V2(d1,CPM_V1(d2)){}
   VV(Z d1, Z d2, T const& t):CPM_V2(d1, CPM_V1(d2,t)){}
      // all components are equal to t
   VV(void):CPM_V2(){}
   VV(CPM_V2 const& x):CPM_V2(x){}

   V<T> lin()const;
      // 'linear version of matrix'
      // by appending all rows

// dimension access
   Z dim1()const{ return Base::dim();}
   Z dim2()const{ return (*this)[1].dim();}

   virtual Z size()const // virtual in base V<...>
      { return dim1()==0 ? 0 : dim1()*dim2();}

// component access
   const T& operator()(Z i, Z j)const
      // returns T() for out of range indexes
   { return Base::read(i).read(j);}

   T& operator()(Z i, Z j)
   { return (*this)[i][j];}

   VV<T> trn()const
      //: transpose
      // Returns the transposed matrix
   {
      VV<T> res(dim2(),dim1());
      for (Z i=1;i<=dim1();i++){
         for (Z j=1;j<=dim2();++j){
            res[j][i] = (*this)[i][j];
         }
      }
      return res;
   }

   template <class Y>
   V<Y> each(void (*f)(T const&, Y&))const;
      //: each
      // collecting the application of f on every pixel in a linear array

   template <class Y>
   VV<Y> operator()(Y (*f)(T const&))const;
      //: operator()
```

```
      // creating a matrix with a transformed value range
};

template <class T>
template <class Y>
V<Y> VV<T>::each(void (*f)(T const&, Y&))const
{
   Z m1=dim1(),m2=dim2(),k=1,i1,i2;
   V<Y> res(m1*m2);
   for (i1=1;i1<=m1;i1++){
      for (i2=1;i2<=m2;i2++){
         f((*this)[i1][i2],res[k++]);
      }
   }
   return res;
}

template <class T>
template <class Y>
VV<Y> VV<T>::operator()(Y (*f)(T const&))const
{
   Z m=dim1(),n=dim2(),i,j;
   VV<Y> res(m,n);
   for (i=1;i<=m;i++){
      for (j=1;j<=n;j++){
         res[i][j]=f((*this)[i][j]);
      }
   }
   return res;
}

template <class T>
V<T> VV<T>::lin()const
{
   if (dim1()==0) return V<T>(0);
   else{
      V<T> res=(*this)[1];
      for (Z i=2;i<=dim1();i++) res&=(*this)[i];
      return res;
   }
}

/************************ class VVV***********************************/
// tensors of rank 3

template <class T>
class VVV: public CPM_V3{ // tensors of rank 3

      typedef CPM_V3 Base;
```

```
public:

// constructors

   VVV(Z d1, Z d2, Z d3, T  const& t):CPM_V3(d1,CPM_V2(d2,CPM_V1(d3,t))){}
   VVV(Z d1, Z d2, Z d3):CPM_V3(d1,CPM_V2(d2,CPM_V1(d3))){}
   VVV(void):CPM_V3(){}
   VVV(CPM_V3 const& x):CPM_V3(x){}

// dimension access
   Z dim1()const{ return Base::dim();}
   Z dim2()const{ return (*this)[1].dim();}
   Z dim3()const{ return (*this)[1][1].dim();}

   virtual Z size()const; // virtual in base V<...>

// component access

   const T & operator()(Z i, Z j, Z k)const
   // returns T() for out of range indexes
   { return Base::read(i).read(j).read(k);}

   T & operator()(Z i, Z j, Z k)
   { return (*this)[i][j][k];}
};

template <class T>
Z VVV<T>::size()const
{
   Z d1=dim1();
   if (d1==0) return d1;
   else{
      Z d2=dim2();
      if (d2==0) return d2;
      else{
         Z d3=dim3();
         if (d3==0) return d3;
         else return d1*d2*d3;
      }
   }
}

/********************** class VVVV ***********************************/
template <class T>
class VVVV: public CPM_V4{ // tensors of rank 4

      typedef CPM_V4 Base;

public:
```

```
// constructors

   VVVV(Z d1, Z d2, Z d3, Z d4, T const& t):
   CPM_V4(d1,CPM_V3(d2,CPM_V2(d3,CPM_V1(d4,t)))){}
   VVVV(Z d1, Z d2, Z d3, Z d4):
   CPM_V4(d1,CPM_V3(d2,CPM_V2(d3,CPM_V1(d4)))){}
   VVVV(void):CPM_V4(){}
   VVVV(CPM_V4 const& x):CPM_V4(x){}

// dimension access
   Z dim1()const{ return Base::dim();}
   Z dim2()const{ return (*this)[1].dim();}
   Z dim3()const{ return (*this)[1][1].dim();}
   Z dim4()const{ return (*this)[1][1][1].dim();}

   virtual Z size()const; // virtual in base V<...>

// component access

   T const& operator()(Z i, Z j, Z k, Z l)const
  // returns T() for out of range indexes
   { return Base::read(i).read(j).read(k).read(l);}

   T& operator()(Z i, Z j, Z k, Z l)
   { return (*this)[i][j][k][l];}
};

template <class T>
Z VVVV<T>::size()const
{
   Z d1=dim1();
   if (d1==0) return d1;
   else{
      Z d2=dim2();
      if (d2==0) return d2;
      else{
         Z d3=dim3();
         if (d3==0) return d3;
         else{
            Z d4=dim4();
            if (d4==0) return d4;
            else return d1*d2*d3*d4;
         }
      }
   }
}
#undef CPM_V4
#undef CPM_V3
#undef CPM_V2
#undef CPM_V1
```

```
}  // namespace CpmArrays

namespace CpmRoot{
// This is a pattern for partial specializations --- to which
// the specializations to class templates belong --- of the
// IO class template started in cpmnumbers.h and the
// Name class template started in cpmword.h.
// A corresponding definition is needed for all non-C+- classes which
// shall be used as template arguments of C+- containers and
// are expected to provide then the same functionality as
// corresponding C+- classes.

   template<class T>
   class IO< std::vector<T> >{ // partial specialization
    public:
      IO(){}
      bool o(std::vector<T> const& v, ostream& str)const
      { return CpmArrays::V<T>(v).prnOn(str);}
      bool i(std::vector<T>& v, istream& str)const
      {
         CpmArrays::V<T> vl;
         bool res=vl.scanFrom(str);
         v=vl.std();
         return res;
      }
   };

#if defined(CPM_NAMEOF)
   template<class T>
   class Name< std::vector<T> >{ // partial specialization
   public:
      Name(){}
      Word operator()(std::vector<T> const& vt )const
      { return Word("std::vector<"&CpmRoot::Name<T>()(T())&">");}
   };
   template<class T>
   class Name< std::set<T> >{ // partial specialization
   public:
      Name(){}
      Word operator()(std::set<T> const& vt )const
      { return Word("std::set<"&CpmRoot::Name<T>()(T())&">");}
   };
#endif
}
#endif
```

36 cpmv.cpp

```
//? cpmv.cpp
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#include <cpmv.h>

using CpmRoot::Z;
using CpmRoot::Word;
using CpmArrays::V;
using CpmArrays::IvZ;

Z CpmArrays::dimMax=100000000;
bool CpmArrays::ranChc=true;
bool CpmArrays::ranChcAlw=true;
bool CpmArrays::signal=false;

void CpmArrays::setDimMax(Z n)
{
   if (n<0)
      cpmerror(cpmwrite(n)&" not OK as value for CpmArrays::dimMax");
   dimMax=n;
}

Z CpmArrays::makeIndValNotMember(Z i, IvZ iv, Outside mode)
{
   if (iv.hasElm(i)) return i;
   if (mode==CYCLIC) return iv.cyc(i);
   else return iv.con(i);
}
      //: make index valid

Z CpmArrays::safeDim(Z n)
{
   if (n<0 || n>dimMax)
```

```
      cpmerror(cpmwrite(n)&" not OK as dimension of CpmArrays::V<>");
   return n;
}

V<Z> CpmArrays::IvZtoVofZ(IvZ const& iv)
{
   Z n=iv.car();
   V<Z> res(n);
   if (n==0) return res;
   Z i,val=iv.inf();
   for (i=res.b();i<=res.e();++i) res[i]=val++;
   return res;
}

V<Z> CpmArrays::VofIvZtoVofZ(V<IvZ> const& viv)
{
   V<Z> res(0);
   for (Z i=viv.b();i<=viv.e();++i) res&=IvZtoVofZ(viv[i]);
   return res;
}

V<Word> CpmArrays::comLine(int argc, char* argv[])
{
   V<Word> res(argc);
   for (Z i=res.b();i<=res.e();i++) res[i]=argv[i-1];
   return res;
}
```

37 cpmva.h

```
//? cpmva.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_VA_H_
#define CPM_VA_H_
/*
   Purpose: Defining a array class Va<T> which exploits the assumption
   that T supports arithmetics. See cpmv.h

   Recent history: 2012-01-22 function tim_(...) and mean() added
*/
#include <cpmvo.h>
#include <cpmtypes.h>

////////////////////// class Va<> ///////////////////////////

namespace CpmArrays{

   using CpmRoot::Z;
   using CpmRoot::R;
   using CpmRoot::Word;
   using CpmRoot::Root;
   using CpmRootX::inf;

template <class T>
   // We assume that T provides (explicitely or implicitely)
   // copy constructor, and assignement
   // or that T is a built-in type.
   // T > T, T < T, T += T, -T, T *= T, T /=T, ostream << T, istream >> T

class Va: public Vo<T>{ // version of Vo with arithmetic operations

   typedef Va<T> Type;
```

```
   typedef T ScalarType;
   typedef Vo<T> Base;

public:

explicit   Va(Z n=0):Vo<T>(n){}
      // note that Va(0) is defined and is different from Va()

explicit Va(IvZ ivz):Vo<T>(ivz){}

   Va(IvZ ivz, T const& t):Vo<T>(ivz,t){}

// constructors from explicit lists
explicit  Va(std::initializer_list<T> il ):Vo<T>(il){}
   // requires C++11
      // constructors from explicit lists such as
      // Va<Z> v{1,2,4,8};

   Va(Z n,T const& t):Vo<T>(n,t){}
      // initializes all n components with t

   Va(const V<T>& h):Vo<T>(h){}
   // 'down-cast'-constructor

   Va(const Vo<T>& h):Vo<T>(h){}
   // 'down-cast'-constructor

   T sum(void)const;
      // returns the sum over all components

   T sumPrd(V<T> const& v)const;
      // returns the sum over (*this)[i]*v[i]. I.e.
      // the scalar product without a conjugation operation for the first
      // factor

   R frac(T const& t)const;
      // returns nt/dim() where nt=card{ j | (*this)[j]==t}
      // For dim()==0 we return 0.

   R frac(const V<T>& v)const;
      // returns nt/dim() where nt=card{ j | (*this)[j]==v[k] for some k}
      // For dim()==0 or v.dim()==0 we return 0.

   virtual Word nameOf()const{
      Word wi="Va<";
      return wi&CpmRoot::Name<T>()(T())&">";
   }

   virtual V<T>* clone(void)const{ return new Va(*this);}
```

```
   template <class S>
   Va<T>& tim_(S const& s);
      //: times
      // using '*=' instead of 'tim_' causes confusion with existing calls
      // to *=.
      // Needed e.g. to multiply instances of Va<R3> by R as in
      // Va<R3> v=...;
      // v.tim_(3.14); // v*=3.14 would look more natural

   template <class S>
   Va<T> tim(S const& s)const;
      //: times
      // Non-mutating form of tim_

   T mean(void)const{ return sum()*cpminv(R(Base::dim()));}
      // returns the sum over all components, divided by
      // their number. This makes sense only if a multiplication
      // T*R is defined

   CPM_SUM_M
   CPM_PRODUCT_M
   CPM_DIFFERENCE
   CPM_DIVISION
   CPM_SCALAR_M
   CPM_DOT_PRODUCT_LEAN
      // does no longer contain a definition of dis. This comes now from
      // V<T>
   CPM_CONJ
   CPM_NORMALIZE
   CPM_IO

private:
   static T sumFunc(T const& t1, T const& t2){ return t1+t2;}
   static T prodFunc(T const& t1, T const& t2){ return t1*t2;}
  // static T prodRFunc(T const& t1, R const& r){ return t1*r;}
   static T negFunc(T const& t) {  return -t;}
   static T idFunc(T const& t) {  return t;}
   static T invFunc(T const& t) {  return Root<T>(t).inv();}
   static T zeroFunc(T const& t)
      {  return Root<T>(t).net(0);}
   static T unitFunc(T const& t)
      {  return Root<T>(t).net(1);}
   static T conjFunc(T const& t){ return Root<T>(t).con();}
   static T dotFunc(T const& t1, T const& t2)
     { return Root<T>(t1).con()*t2;}
   static void accFunc(T& z, T const& zAdd){ z+=zAdd;}
};

// class functions:
```

```
template <class T>
T Va<T>::sum(void)const
{
   return Base::fAcc1(idFunc,accFunc);
}

template <class T>
T Va<T>::operator | (Va<T> const& h)const
{
   return Base::fAcc2(dotFunc,accFunc,h);
}

template <class T>
T Va<T>::sumPrd(V<T> const& h)const
{
   return Base::fAcc2(prodFunc,accFunc,h);
}

template <class T>
CpmRoot::R Va<T>::frac(T const& t)const
{
   Z n=Base::dim();
   if (n==0) return 0;
   Z sum=0;
   for (Z i=1;i<=n;i++) if ((*this)[i]==t) sum++;
   return ((CpmRoot::R)(sum))/n;
}

template <class T>
CpmRoot::R Va<T>::frac(const V<T>& v)const
{
   Z n=Base::dim();
   if (n==0) return 0;
   Z nv=v.dim();
   if (nv==0) return 0;
   Z i,j,sum=0;
   for (i=1;i<=n;i++){
      T ti=(*this)[i];
      for (j=1;j<=nv;j++){
         if (ti==v[j]) sum++;
      }
   }
   return ((CpmRoot::R)(sum))/n;
}

// main task

template <class T>
Va<T> Va<T>::neg(void)const
{
```

```
   V<T> v_Res=Base::fa(negFunc);
   return Va<T>(v_Res);
}

template <class T>
Va<T> Va<T>::inv(void)const
{
   V<T> v_Res=Base::fa(invFunc);
   return Va<T>(v_Res);
}

template <class T>
Va<T> Va<T>::con(void)const
{
   V<T> v_Res=Base::fa(conjFunc);
   return Va<T>(v_Res);
}

template <class T>
Va<T> Va<T>::net(Z i)const
{
   V<T> v_Res= i==1 ? Base::fa(unitFunc) : Base::fa(zeroFunc);
   return Va<T>(v_Res);
}

template <class T>
Va<T>& Va<T>::operator +=(const Va<T>& s)
{
   Base::ff_(sumFunc,s);
   return *this;
}

template <class T>
Va<T>& Va<T>::operator +=(T const& t)
{
   Base::fc_(sumFunc,t);
   return *this;
}

template <class T>
Va<T>& Va<T>::operator *=(const Va<T>& s)
{
   Base::ff_(prodFunc,s);
   return *this;
}

template <class T>
Va<T>& Va<T>::operator *=(T const& t)
{
   Base::fc_(prodFunc,t);
```

```
   return *this;
}

template <class T>
template <class S>
Va<T>& Va<T>::tim_(S const& s)
{
   for (Z i=Base::b();i<=Base::e();++i) Base::cui(i)*=s;
   return *this;
}

template <class T>
template <class S>
Va<T> Va<T>::tim(S const& s)const
{
   Va<T> res=*this;
   return res.tim_(s);
}


template <class T>
bool Va<T>::prnOn(ostream& str)const
{
   return Base::prnOn(str);
}

template <class T>
bool Va<T>::scanFrom(istream& str)
{
   return Base::scanFrom(str);
}

////////////////////////// iterated templates ////////////////////////////
// describing multi-indexed quantities
///////////////////////////// class VVa //////////////////////////////////
// matrices without matrix product
// tensors of rank 2
// We derive VVa<T> from VVo<T> and not from Va<Va<T> >. In the
// latter case Va<T> would have to be the scalar type for VVa<T>
// which is not the case. I tried this 2005-08-19 but came deep
// 'in the woods'. Since thus the arithmetics interface is not
// yet fixed by the definition, we are free to define it anew in
// a more lean way as that of Va. Especially we avoid to
// define friend functions. Also simply taking over the interface
// of Va would not be adequate from the point of view of
// implementation efficiency.
// The implementation of VVa is such that it can be carried over to
// higher numbers of indexes iteratively

template <class T>
```

```
class VVa: public VVo<T>{ // version of VVo with arithmetic operations

   typedef VVo<T> Base;
   typedef VVa<T> Type;
   typedef T ScalarType;

public:
   CPM_LIN
      // notice that function con() declared here will be
      // implemented as conjugation of the matrix elements
      // only and not as accompanied by a matrix
      // transposition. The present implementation is natural
      // for an interpretation of VVa's as T-valued areal
      // distributions (as in images or wave functions)
      // Unlike the case for Va<T>, we do not include CPM_IO
      // in the interface of VVa<T>
// constructors
   VVa(Z d1, Z d2):Base(d1,d2){}
   VVa(Z d1, Z d2, T const& t):Base(d1,d2,t){}
      // all components are equal to t
   VVa(void):Base(){}
   VVa(V< V<T> >  const& x):Base(x){}
   VVa(VV<T> const& x):Base(x){}
   VVa(VVo<T> const& x):Base(x){}
};

template <class T>
VVa<T> VVa<T>::operator+(VVa<T> const& s)const
{
   Z d1=Base::dim(),d2=s.dim();
   Z d=inf<Z>(d1,d2);
   VVa<T> res(d,0);
   for (Z i=1;i<=d;i++){
      Va<T> ti=(*this)[i];
      Va<T> si=s[i];
      res[i]=ti+si;
   }
   return res;
}

template <class T>
VVa<T> VVa<T>::operator-(VVa<T> const& s)const
{
   Z d1=Base::dim(),d2=s.dim();
   Z d=inf<Z>(d1,d2);
   VVa<T> res(d,0);
   for (Z i=1;i<=d;i++){
      Va<T> ti=(*this)[i];
      Va<T> si=s[i];
      res[i]=ti-si;
```

```
   }
   return res;
}

template <class T>
VVa<T> VVa<T>::operator*(T const& r)const
{
   Z d=Base::dim();
   VVa<T> res(d,0);
   for (Z i=1;i<=d;i++){
      Va<T> ti=(*this)[i];
      res[i]=ti*r;
   }
   return res;
}

template <class T>
VVa<T> VVa<T>::operator-(void)const
{
   Z d=Base::dim();
   VVa<T> res(d,0);
   for (Z i=1;i<=d;i++){
      Va<T> ti=(*this)[i];
      res[i]=-ti;
   }
   return res;
}

template <class T>
VVa<T> VVa<T>::con(void)const
{
   Z d=Base::dim();
   VVa<T> res(d,0);
   for (Z i=1;i<=d;i++){
      Va<T> ti=(*this)[i];
      res[i]=ti.con();
   }
   return res;
}

template <class T>
T VVa<T>::operator|(VVa<T> const& s)const
{
   Z d1=Base::dim(),d2=s.dim();
   Z d=inf<Z>(d1,d2);
   T res=T();
   for (Z i=1;i<=d;i++){
      Va<T> ti=(*this)[i];
      Va<T> si=s[i];
      res+=(ti|si);
```

```
   }
   return res;
}

////////////////////////// class VVVa ////////////////////////////////////
// tensors of rank 3

template <class T>
class VVVa: public VVVo<T>{// version of VVVo with arithmetic operations

   typedef VVVo<T> Base;
   typedef VVVa<T> Type;
   typedef T ScalarType;

public:
   CPM_LIN
      // notice that function con() declared here will be
      // implemented as conjugation of the components.
      // The present implementation is natural
      // for an interpretation of VVVa's as T-valued spatial
      // distributions (as in wave functions).
      // Unlike the case for Va<T>, we do not include CPM_IO
      // in the interface of VVVa<T>
// constructors
   VVVa(Z d1,Z d2,Z d3):Base(d1,d2,d3){}
   VVVa(Z d1,Z d2,Z d3,T const& t):Base(d1,d2,d3,t){}
      // all components are equal to t
   VVVa(void):Base(){}
   VVVa( V< V< V<T> > >  const& x):Base(x){}
   VVVa(VVV<T> const& x):Base(x){}
   VVVa(VVVo<T> const& x):Base(x){}
};

template <class T>
VVVa<T> VVVa<T>::operator+(VVVa<T> const& s)const
{
   Z d1=Base::dim(),d2=s.dim();
   Z d=inf<Z>(d1,d2);
   VVVa<T> res(d,0,0);
   for (Z i=1;i<=d;i++){
      VVa<T> ti=(*this)[i];
      VVa<T> si=s[i];
      res[i]=ti+si;
   }
   return res;
}

template <class T>
VVVa<T> VVVa<T>::operator-(VVVa<T> const& s)const
{
```

```
   Z d1=Base::dim(),d2=s.dim();
   Z d=inf<Z>(d1,d2);
   VVVa<T> res(d,0,0);
   for (Z i=1;i<=d;i++){
      VVa<T> ti=(*this)[i];
      VVa<T> si=s[i];
      res[i]=ti-si;
   }
   return res;
}

template <class T>
VVVa<T> VVVa<T>::operator*(T const& r)const
{
   Z d=Base::dim();
   VVVa<T> res(d,0,0);
   for (Z i=1;i<=d;i++){
      VVa<T> ti=(*this)[i];
      res[i]=ti*r;
   }
   return res;
}

template <class T>
VVVa<T> VVVa<T>::operator-(void)const
{
   Z d=Base::dim();
   VVVa<T> res(d,0,0);
   for (Z i=1;i<=d;i++){
      VVa<T> ti=(*this)[i];
      res[i]=-ti;
   }
   return res;
}

template <class T>
VVVa<T> VVVa<T>::con(void)const
{
   Z d=Base::dim();
   VVVa<T> res(d,0,0);
   for (Z i=1;i<=d;i++){
      VVa<T> ti=(*this)[i];
      res[i]=ti.con();
   }
   return res;
}

template <class T>
T VVVa<T>::operator|(VVVa<T> const& s)const
{
```

```
   Z d1=Base::dim(),d2=s.dim();
   Z d=inf<Z>(d1,d2);
   T res=T();
   for (Z i=1;i<=d;i++){
      VVa<T> ti=(*this)[i];
      VVa<T> si=s[i];
      res+=(ti|si);
   }
   return res;
}

////////////////////////// class VVVVa ////////////////////////////////////
// tensors of rank 4

template <class T>
class VVVVa: public VVVVo<T>{//version of VVVVo with arithmetic operations

   typedef VVVVo<T> Base;
   typedef VVVVa<T> Type;
   typedef T ScalarType;

public:
   CPM_LIN
      // notice that function con() declared here will be
      // implemented as conjugation of the components.
      // The present implementation is natural
      // for an interpretation of VVVVa's as T-valued spatial
      // distributions (as in wave functions).
      // Unlike the case for Va<T>, we do not include CPM_IO
      // in the interface of VVVVa<T>
// constructors
   VVVVa(Z d1,Z d2,Z d3,Z d4):Base(d1,d2,d3,d4){}
   VVVVa(Z d1,Z d2,Z d3,Z d4,T const& t):Base(d1,d2,d3,d4,t){}
      // all components are equal to t
   VVVVa(void):Base(){}
   VVVVa( V< V< V< V<T> > > > const& x):Base(x){}
   VVVVa(VVVV<T> const& x):Base(x){}
   VVVVa(VVVVo<T> const& x):Base(x){}
};

template <class T>
VVVVa<T> VVVVa<T>::operator+(VVVVa<T> const& s)const
{
   Z d1=Base::dim(),d2=s.dim();
   Z d=inf<Z>(d1,d2);
   VVVVa<T> res(d,0,0,0);
   for (Z i=1;i<=d;i++){
      VVVa<T> ti=(*this)[i];
      VVVa<T> si=s[i];
      res[i]=ti+si;
```

```
   }
   return res;
}

template <class T>
VVVVa<T> VVVVa<T>::operator-(VVVVa<T> const& s)const
{
   Z d1=Base::dim(),d2=s.dim();
   Z d=inf<Z>(d1,d2);
   VVVVa<T> res(d,0,0,0);
   for (Z i=1;i<=d;i++){
      VVVa<T> ti=(*this)[i];
      VVVa<T> si=s[i];
      res[i]=ti-si;
   }
   return res;
}

template <class T>
VVVVa<T> VVVVa<T>::operator*(T const& r)const
{
   Z d=Base::dim();
   VVVVa<T> res(d,0,0,0);
   for (Z i=1;i<=d;i++){
      VVVa<T> ti=(*this)[i];
      res[i]=ti*r;
   }
   return res;
}

template <class T>
VVVVa<T> VVVVa<T>::operator-(void)const
{
   Z d=Base::dim();
   VVVVa<T> res(d,0,0,0);
   for (Z i=1;i<=d;i++){
      VVVa<T> ti=(*this)[i];
      res[i]=-ti;
   }
   return res;
}

template <class T>
VVVVa<T> VVVVa<T>::con(void)const
{
   Z d=Base::dim();
   VVVVa<T> res(d,0,0,0);
   for (Z i=1;i<=d;i++){
      VVVa<T> ti=(*this)[i];
      res[i]=ti.con();
```

```
   }
   return res;
}

template <class T>
T VVVVa<T>::operator|(VVVVa<T> const& s)const
{
   Z d1=Base::dim(),d2=s.dim();
   Z d=inf<Z>(d1,d2);
   T res=T();
   for (Z i=1;i<=d;i++){
      VVVa<T> ti=(*this)[i];
      VVVa<T> si=s[i];
      res+=(ti|si);
   }
   return res;
}

} // namespace
#endif
```

38 cpmviewport.h

```
//? cpmviewport.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_VIEWPORT_H_
#define CPM_VIEWPORT_H_
/*

   class Viewport is the Cpm interface to the graphical
   capabilities of the hardware.

*/
#include <cpmv.h>

namespace CpmGraphics{

   using CpmRoot::Z;
   using CpmRoot::R;
   using CpmRoot::L;
   using CpmRoot::Word;
   using CpmArrays::V;
   using CpmArrays::IvZ;
   using namespace CpmStd;
      // see cpmnumbers.h, only I/O related stuff from std.
      // I had using namespace std before. This had the bad effect that
      // one of my function names ('ignore' in this case), even if defined
      // in an anonymous namespace, once clashed with std::ignore.
      // So, to avoid clashes with the ever expanding namespace std I will
      // never rely on std as a whole.

#if defined(WIN32)
   const Word Platform("Windows");
#else
   const Word Platform("Cygwin or Linux");
```

```
#endif

////////////////////////////////// struct xy ////////////////////////////
// Simple data structure for graphical points. As ever in graphics:
// x-axis horizontal from left to right (thus perfectly normal)
// y-axis downwards. Meaningful coordinates belong to {0,1,2,...}.
   struct xy{ // graphical points
      Z x,y;
      xy(Z a=0, Z b=0):x(a),y(b){}
      xy operator+(xy p)const{ return xy(x+p.x,y+p.y);}
      xy operator-(xy p)const{ return xy(x-p.x,y-p.y);}
      xy operator-()const{ return xy(-x,-y);}
      xy& operator+=(xy p){ x+=p.x;y+=p.y;return *this;}
      xy& operator-=(xy p){ x-=p.x;y-=p.y;return *this;}
      bool operator==(xy p)const{ return x==p.x && y==p.y;}
   };

// simple data structure for graphical rectangles

   struct xyxy{ // graphical rectangles
      Z x1,y1,x2,y2; // (x1,y1) left upper corner,
         // (x2,y2) right lower corner

      xyxy(Z a=0,Z b=0,Z c=0,Z d=0):x1(a),y1(b),x2(c),y2(d){}
         // construction from corner coordinates

      xyxy(xy p1, xy p2):x1(p1.x),y1(p1.y),x2(p2.x),y2(p2.y){}
         // construction from corners

      xyxy(xy p, Z w, Z h):x1(p.x),y1(p.y),x2(x1+w-1),y2(y1+h-1){}
         // construction from left upper corner, width, and height

      Z w()const{ return x2-x1+1;}

      Z h()const{ return y2-y1+1;}

      R aspRat()
      //: aspect ratio
      { Z h_=h(); cpmassert(h_>=1,"xyxy::aspRat()"); return R(w())/h_;}

      bool isVoid()const{ return x1<0;}
         // since the meaningfull graphical rectangles have no
         // negative components, this is an easy way to encode
         // exceeptional behavior

      xyxy operator+(xy p)const
         { return xyxy(x1+p.x,y1+p.y,x2+p.x,y2+p.y);}
      xyxy operator-(xy p)const
         { return xyxy(x1-p.x,y1-p.y,x2-p.x,y2-p.y);}
      xyxy& operator+=(xy p){ return *this=*this+p;}
```

```
xyxy& operator-=(xy p){ return *this=*this-p;}

xyxy operator|(xyxy r)const;
   // smallest xyxy that contains both *this and r
   // kind of union

xyxy& operator|=(xyxy r){ return *this=(*this)|r;}
   // | as a mutating operation

xyxy operator&(xyxy r)const;
   // returns the set section of *this and r.
   // The result res satisfies res.isVoid()==true
   // if this section is the void set.

xyxy& operator&=(xyxy r){ return *this=(*this)&r;}
   // & as a mutating operation

xy operator[](Z i)const{ return i<=1 ? xy(x1,y1) : xy(x2,y2);}
   // For an instance r of xyxy r[1] is the upper left corner
   // and r[2] is the lower right corner.

xy cor(Z i)const
//: corner
// Returns the i-th corner. Here corners are numbered counter
// clock-wise starting with the left upper corner as 1. All
// non-matching arguments give corner 1. Thus no exceptions!
{
   if (i==2) return xy(x1,y2);
   if (i==3) return xy(x2,y2);
   if (i==4) return xy(x2,y1);
   return xy(x1,y1);
}

bool hasElm(xy const& p)const
//: has element
// Makes a xyxy a set of xy's
{
   return IvZ(x1,x2).hasElm(p.x)&& IvZ(y1,y2).hasElm(p.y);
}

xyxy scaleToFit(xyxy r)const;
//: scale to fit
// Returns the largest rectangle that
// 1. has the aspect ratio of r
// 2. has the same left-upper corner as *this
// 3. is contained in *this.

friend ostream& operator<<(ostream& os, xyxy const& r)
{
   os<<" "<<r.x1<<", "<<r.y1<<", "<<r.x2<<", "<<r.y2<<endl;
```

```
         return os;
      }
   };

   struct rgb;
   class Img24;

////////////////////////// class ColRef ////////////////////////////////

const float i255=1./255;

   class ColRef{ //lean 24-bit color-values
      // light 24-bit representation of color-values
      // that easily converts (back and forth) to the low-level color
      // data

      friend struct rgb;
      friend class Img24;
      static Z x(Z z);
      typedef ColRef Type;
   public:
      static Z lowerLimit;
      static Z upperLimit;
      L r,g,b; // all values valid, so public OK
      CPM_IO
      CPM_ORDER
      ColRef():r((L)lowerLimit),g((L)lowerLimit),b((L)lowerLimit){}
      ColRef(Z r_, Z g_, Z b_):r((L)x(r_)),g((L)x(g_)),b((L)x(b_)){}
      ColRef(L r_, L g_, L b_):r(r_),g(g_),b(b_){}
      L getR()const{ return r;}
      L getG()const{ return g;}
      L getB()const{ return b;}
      L operator[](Z i)const
         { if (i==3) return b; else if (i==2) return g; else return r;}
      float glR()const{ return i255*r;}
      float glG()const{ return i255*g;}
      float glB()const{ return i255*b;}
      bool isGray()const{ return r==g && g==b;}
      ColRef add(ColRef const& cr)const
      { return ColRef((Z)r+(Z)cr.r,(Z)g+(Z)cr.g,(Z)b+(Z)cr.b);}
      Word nameOf()const{ return "ColRef";}
   };

////////////////////////// struct rgb ////////////////////////////////

struct rgb { // Z-valued red green blue
   // a simple struct to provide a convenient interface for
   // functions draw. This class will be a base class for deriving
   // a more functional class Color in namespace CpmImaging.
   Z r,g,b;
```

```
rgb():r(ColRef::lowerLimit),g(ColRef::lowerLimit),
   b(ColRef::lowerLimit){}

rgb(Z r_, Z g_, Z b_, bool norm=true):r(r_),g(g_),b(b_)
   {if (norm) normalize();}

rgb(R r_, R g_, R b_, bool norm=true):
r(cpmrnd(r_)),g(cpmrnd(g_)),b(cpmrnd(b_))
{if (norm) normalize();}

explicit rgb(ColRef cr):r(cr.r),g(cr.g),b(cr.b){}

virtual Word nameOf()const{ return Word("rgb");}

operator ColRef()const{ return ColRef(r,g,b);}

rgb operator~()const
   // returning the complementary color
{
   Z up=ColRef::upperLimit; return rgb(up-r,up-g,up-b);
}

void normalize(){ r=ColRef::x(r); g=ColRef::x(g); b=ColRef::x(b);}
   // ensuring the right value range by simply cutting

static rgb mix(rgb c1, rgb c2, R w1, R w2, R gamma);
   // mixing colors according to function mixingRule

R gray()const{return (r+g+b)/3.;}

bool isGray()const{ return r==g && g==b;}
   //: is gray

bool isBlack()const{ return r==0 && g==0 && b==0;}
   //:: is black

bool isWhite()const{ return r==255 && g==255 && b==255;}
   //:: is white

rgb operator*(R fac)const
{ return rgb(cpmrnd(fac*r),cpmrnd(fac*g),cpmrnd(fac*g));}

static Z mixingRule(Z z1, Z z2, R w1, R w2, R gamma);
   // defines mixing of colors according to given weights
   // w1+w2=1 is not assumed
   // ( w1=w2=1 corresponds to addition of light)

void write(ostream& str)const
{ str<<endl<<"rgb: r="<<r<<" g="<<g<<" b="<<b<<endl;}
```

```
   friend void operator<<(ostream& str, rgb const& c)
   { str<<c.r; str<<c.g; str<<c.b;}

   friend void operator>>(istream& str, rgb & c)
   { str>>c.r; str>>c.g; str>>c.b;}
};

////////////////////////////// class Font ////////////////////////////////
// Presently only Helvetica 12 needed from this font system

class Font{
// only Helvetica 12 needed from this font system of GLUT
   void* name;
   Z h;
      // height in pixels
   Z mw;
      // maximum width in pixels (estimation based on height)
   Z d;
      // suitable line separation taking decent ('Unterlaenge') into
      // account
public:
   explicit Font(Z type=4);
   explicit Font(R height);
      // creates an instance of a font height that comes closest
      // to the value given by the argument. See the simple
      // implementation
      // code for the details

   Z getHeight()const{ return h;}
   Z getWidth()const{ return mw;}
      // maximum width in pixels (typically width of W in proportional
      // fonts
   Z getLineSep()const{ return d;}
   void* getName()const{ return name;}
};

////////////////////////// class Rec ////////////////////////////////////
// Pixel rectangle rec_ which always fits Viewport::win()
// and linear image buffer mem_ of arbitrary size.
// The display function vis of this class shows mem_
// always whithin rec_. This function is Cpm's means
// to bring pixel-based graphical information to screen.
// Present implementation relies on OpenGL (only if
// CPM_NOGRAPHICS is not set). If CPM_NOGRAPHICS is
// defined we need no graphics libraries from the system
// and can generate all graphics as PPM-images (on an
// pixel rectangle that can be set arbitrarily
// by editing cpmconfig.ini.
```

```
class Rec{ // pixel rectangle which always fits Viewport::win()
   xyxy rec_;
   Z n_;
      // dimension of mem_
   L* mem_;
      // should be unsigned char = L
      // Data element of this type needed in function vis()
      // by OpenGL. This is linearized data in two-fold manner:
      // The rectangular image area is un-fold to a linear chain,
      // and the r g b pixel data are represented as a list of
      // characters.

public:
   explicit Rec(xyxy r=xyxy(0,0,0,0));
      // Constructor which fills the image data from the one and only
      // initialized Viewport, more precisely from the data
      // within the set section of r and the Viewport rectangle
      // Viewport::win()
      // Present implementation relies on OpenGL
   explicit Rec(V< V<ColRef> > const& bh);
      // constructs a Rec from the matrix data bh. The corresponding
      // rectangle rec_ is created such that it fits Viewport::win().
   Rec(xyxy r, L c);
      // The pixels within the rectangle r are set equal to c
      // On display this is always a gray (black to white) frame.
      // Notice that colorful colors would be more difficult to
      // handle here.
      // Rectangle is created such that it fits Viewport::win().
   Rec(Rec const& r);
   ~Rec();
   Rec& operator=(Rec const& r);
   static bool db; // double buffering
   Z dim()const{ return n_;}
      // number of bytes = 3*number of pixels
   Z h()const{ return rec_.h();}
   Z w()const{ return rec_.w();}
   Z x1()const{ return rec_.x1;}
   Z y1()const{ return rec_.y1;}
   Z x2()const{ return rec_.x2;}
   Z y2()const{ return rec_.y2;}
   xyxy rec()const{ return rec_;}
   xy operator[](Z i)const{ return rec_[i];}
   xy cor()const{ return rec_[1];}
      //: corner
      // actually left upper corner
   Rec& operator+=(xy shift);
      // result is always within Viewport::win()
   Rec operator+(xy shift)const{ Rec res=*this; res+=shift; return res;}
   Rec& operator-=(xy shift);
      // result is always within Viewport::win()
```

```
   Rec operator-(xy shift)const{ Rec res=*this; res-=shift; return res;}
   void vis()const;
      //: visualize
      // Displays the image data in mem_ on the area rec_ (which always
      // is a subset of the viewpoint window).
      // On this function all bitmap imaging in C+- is based.
      // Present implementation relies on OpenGL. Function glDrawPixels
      // needs a L* as the last argument. So it would not be convenient to
      // replace the mem_ data element of the present class by one of type
      // V<L>.
   V< V<ColRef> > fold()const;
      // returns a matrix-like memory version of mem_
};

////////////////////////// class Viewport ////////////////////////////////
/*
class Viewport is a lean interface to the
systems graphical capabilities. It also defines and activates a
statusbar supporting a continuous message stream from a running
program going to a dedicated area of the screen.

In order for the OpenGL version to work (which is the only one that
survived, I had working implementations based on MSFC and on FLTK)
one has to do some initialisation in function main() as is done in
cpmapplication.cpp.

The class as it stands is a rather 'blind viewport'. Apart from filling
rectangles with chosen color and setting text in GLUT's native fonts
it allows  n o t to perform actions which can be seen on screen.
The viewport becomes more active only with class Img24 (see cpmimg24.h)
which provides a data element of type Rec (its name is displayBuffer) and
tools to fill and display it. Each call of an constructor of class
CpmGraphics::Frame activates Img24.
See in particular CpmGraphics::FramesetStaticSize(Z width, Z height)
which is called in int main(int argc, char *argv[]) when this function
is defined in cpmapplication.cpp
*/

class Viewport{ // Lean interface to the system's graphical capabilities.
   // The Viewport class has no non-static data member, so
   // every member function could be declared static without changing
   // their behavior. However, it is more convenient to say
   //
   // Viewport vp;
   // vp.addText(10,10,"hello world");
   //
   // than
   //
   // Viewport::addText(10,10,"hello world");
   //
```

```
   // Notice that there is no way to have more than one different
   // instances of Viewport. So we can control a single graphical area,
   // which in my normal applications is one 'application window'.
   // Classes Frame, Graph, and Frames organize as many rectangular
   // subframes of this basic window as we want.
   static bool initialized;
   static Z applWindowWidth;
   static Z applWindowHeight;
   static Z fullWindowHeight;
   static xyxy window;
   static xyxy fullWindow;
   static R gamma;
      // a gamma-value for the systems video screen
      // presently set to 2.2 (usual in Windows systems)
      // in code. Would not be illogical to read this in from
      // cpmsystemdependencies.h.
      // This value of gamma will be used as default for the
      // image class Img24.
   static ColRef colText;
   static xyxy seg1;
   static xyxy seg2;
   static xyxy seg3;
   static xyxy seg4;
   static ColRef cs1;
   static ColRef cs2;
   static ColRef cs3;
   static ColRef cs4;
   static Z heightStatusBar;
   static Z size;
   static Z rank;

public:
   static void setColText(Z r, Z g, Z b){ colText=ColRef(r,g,b);}
   static void setColText(ColRef cr){ colText=cr;}
   static void setColText(rgb crgb){ colText=(ColRef)crgb;}
   static ColRef getColText(){ return colText;}

   static xyxy win(){ return window;}
      // Window available for regular display activities (that do
      // avoid writing to the status bar).
   static xyxy fullWin(){ return fullWindow;}
      // Graphics window including the status bar.

   static Z getHeightStatusBar(){ return heightStatusBar;}
   static void onStatusBar(const Word& w, Z i);
   static void clrPan();
      //: clear panes
   static void fill(xyxy rec, ColRef cr);
      // Fills rectangle rec with solid color cr
   static void placeWord(Z tx, Z ty, Word const& w,
```

```
   const Font& font, Z wMax=500);
   // wMax sets the maximum number of pixels for the length of the
   // printed picture of w. If w is longer it gets cut ( e.g. for
   // disciplined writing on status bars.
static xyxy textLine(Z tx, Z ty, Word const& w,
   ColRef crText, R h);
   // writing w in color crText.
   // Only the pixels making up the characters are written so that
   // previous screen content is only minimally shadowed by the text.
   // The text may not be easy to read, however.
   // h is a proposal for the font hight (the actually chosen
   // hight depends on the available fonts). Introduced
   // 2005-04-14
   // The return value is a enclosing box for the text - a bit larger
   // than the minimum enclosing box.

Viewport(Z w, Z h, Word const& title);
Viewport(){}

static R getGamma(){ return gamma;}
static void setGamma(R g){ gamma=g;}
static bool isInitialized(){return initialized!=0;}
   // does not allow to change status !!
static Z pelX(){ return applWindowWidth;}
static Z pelY(){ return applWindowHeight;}

Z getCol()const{return applWindowWidth;}
   // number of columns
   // horizontal extension of the graphically accessible
   // application window in pixels. The range of the first pixel
   // access index (mostely named x or i) is {0,1,...,getCol()-1}
Z getWidth()const{return applWindowWidth;}
   // legalizing another conventional name

Z getLin()const{ return applWindowHeight;}
   // number of lines (rows)
   // vertical extension of the graphically accessible
   // application window in pixels. The range of the second pixel
   // access index (mostely named y or j) is {0,1,...,dim2()-1}
Z getHeight()const{ return applWindowHeight;}
   // legalizing another conventional name

Z getFullHeight()const{return fullWindowHeight;}

Z dim1()const{return applWindowWidth;}
Z dim2()const{return fullWindowHeight;}
   // The screen area under control of Viewport consists of
   // dim1()*dim2() writable and readable pixels. In the case
   // of OpenGL-implementation a part of the lines are used as
   // forming a statusbar. So these should not be used for
```

```
      // displaying images and graphics (but they perfectly
      // can from a syntactic point of view).

   Z getRank()const{ return rank;}
   Z getSize()const{ return size;}

   xyxy addText(Z x1, Z y1, Word const& text, R fontHeight=12, Z font=1);
      // (x1,y1) is the upper left corner of the writing box.
      // The return value is a box which encloses the text and is a bit
      // larger than minimal.
      // Thus the last argument remains passive since we select the font
      //  the height of which comes closest to fontHeight.

      // Here we are the names of the available fonts together with a
      // number which does not matter here.
      // font = 1 : GLUT_BITMAP_TIMES_ROMAN_10
      // font = 2 : GLUT_BITMAP_HELVETICA_10
      // font = 3 : GLUT_BITMAP_HELVETICA_12
      // font = 4 : GLUT_BITMAP_8_BY_13
      // font = 5 : GLUT_BITMAP_9_BY_15
      // font = 6 : GLUT_BITMAP_HELVETICA_18
      // font = 7 : GLUT_BITMAP_TIMES_ROMAN_24
};

}// namespace

#endif
```

39 cpmviewport.cpp

```
//? cpmviewport.cpp
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#include <cpmsystemdependencies.h>
   // This file is not included in any header file in order
   // to make only those translation units dependent on these
   // macros which really depend on them.
   // Important that this precedes all other includes.
#include <cpmviewport.h>
//#include <cpmtypes.h> // experiment, needed for cpmdebug

using CpmRoot::Z;
using CpmRoot::L;
using CpmRoot::R;
using CpmRoot::Word;
using CpmRoot::getByte;

using CpmSystem::Message;
//using CpmSystem::R_;

using CpmArrays::V;
using CpmArrays::LEAN;

using CpmGraphics::Rec;
using CpmGraphics::xy;
using CpmGraphics::xyxy;
using CpmGraphics::Viewport;
using CpmGraphics::rgb;
using CpmGraphics::ColRef;

using namespace std;

Z CpmGraphics::Viewport::heightStatusBar=16;
```

```
////////////////////////// struct xyxy ////////////////////////////////////
namespace {
Z infZ(Z i, Z j){ return i<=j ? i : j;}
Z supZ(Z i, Z j){ return i<=j ? j : i;}
R infR(R i, R j){ return i<=j ? i : j;}
R supR(R i, R j){ return i<=j ? j : i;}
   // to avoid need of namespace CpmRootX
}

xyxy xyxy::operator|(xyxy r)const
{
   Z rx1=infZ(x1,r.x1);
   Z ry1=infZ(y1,r.y1);
   Z rx2=supZ(x2,r.x2);
   Z ry2=supZ(y2,r.y2);
   return xyxy(rx1,ry1,rx2,ry2);
}

xyxy xyxy::operator&(xyxy r)const
{
   if (x2 < r.x1) return xyxy(-1,-1,-1,-1);
   if (r.x2 < x1) return xyxy(-1,-1,-1,-1);
      // if upper end of one is smaller than the lower
      // end of the other, then the projections of
      // *this and r to the x-axis are disjoint
   if (y2 < r.y1) return xyxy(-1,-1,-1,-1);
   if (r.y2 < y1) return xyxy(-1,-1,-1,-1);
      // same for x replaced by y
   Z rx1=supZ(x1,r.x1);
   Z ry1=supZ(y1,r.y1);
   Z rx2=infZ(x2,r.x2);
   Z ry2=infZ(y2,r.y2);
   return xyxy(rx1,ry1,rx2,ry2);
}

xyxy xyxy::scaleToFit(xyxy r)const
{
   Word loc("xyxy::scaleToFit(xyxy)");
   cpmassert(!r.isVoid()&&!isVoid(),loc);
   R w1=w(), h1=h(), w2=r.w(), h2=r.h();
   R sw=R(w1)/w2; // scale factor to adjust width
   R sh=R(h1)/h2; // scale factor to adjust height
   Z wN,hN; // width and hight of the result
   R s=infR(sw,sh);
   wN=cpmtoz(s*w2);
   hN=cpmtoz(s*h2);
   return (*this)&xyxy(cor(1),wN,hN);
 }

////////////////////////// class Rec //////////////////////////////////////
```

```
bool Rec::db=false;

Rec& Rec::operator+=(xy shift)
{
   rec_+=shift;
   rec_&=Viewport::win();
   return *this;
}
Rec& Rec::operator-=(xy shift)
{
   (rec_-=shift)&=Viewport::win(); return *this;
}
// the following implementation follows that of V<> in cpmv.h

Rec::~Rec(){ /*printf(" ~Rec() called ");*/ delete[] mem_;}

Rec::Rec(xyxy r): rec_(r&Viewport::win()),
   n_(rec_.w()*rec_.h()*3),mem_(new L[n_])
// together with the usage in Rec::vis, this is all we need from OpenGL
{
#if !defined(CPM_NOGRAPHICS)
   glFlush();
   Z x1=rec_.x1, y1=rec_.y1, w=rec_.w(), h=rec_.h(), wBytes=3*w;
   L* p=mem_;
   Z gly=Viewport::fullWin().h()-y1-h;
   Z j=h;
   while (j--){
     L* buff=new L[wBytes];
     glReadPixels(x1,gly++,w,1,GL_RGB,GL_UNSIGNED_BYTE,buff);
     Z ib=wBytes;
     L* q=buff;
     while (ib--) *p++=*q++;
     delete[] buff;
   }
#endif
}

Rec::Rec(xyxy r, L c): rec_(r&Viewport::win()),
   n_(rec_.w()*rec_.h()*3),mem_(new L[n_])
{
   L* p=mem_;
   Z j=n_;
   while (j--)*p++=c;
}
      // The pixels within the rectangle r are set equal to c
      // On display this is always a gray (black to white) frame.
      // Notice that colorful colors would be more difficult to
      // handle here
```

```
Rec& Rec::operator=(Rec const& r)
{
   if (this==&r) return *this;
   rec_=r.rec_;
   Z nMem=n_;
   n_=r.n_;
   if (n_!=nMem){ //  avoid superfluous delete-new action
      delete[] mem_;
      mem_=new L[n_];
   }
   L* q=r.mem_;
   L* it=mem_;
   Z i=n_;
   while (i--) *it++ = *q++;
   return *this;
}

Rec::Rec(Rec const& r):rec_(r.rec_),n_(r.n_)
{
   L* itp=r.mem_;
   L* q=new L[n_];
   L* it=q;
   Z i=n_;
   while (i--) *it++=*itp++;
   mem_=q;
}

Rec::Rec(V< V<ColRef> > const& bh)
{
   Z h1=bh.dim();
   Z w1=bh.fir().dim();
   xyxy r1(xy(0,0),w1,h1);
   rec_=r1&Viewport::win();
   Z height=rec_.h();
   Z width=rec_.w();
   n_=height*width*3;
   mem_=new L[n_];
   L* p=mem_;
   for (Z i=height-1;i>=0;i--){
      for (Z j=0;j<width;j++){
         ColRef val=bh[i][j];
         *p++=val[1];
         *p++=val[2];
         *p++=val[3];
       }
   }
}

V< V<ColRef> >  Rec::fold()const
{
```

```
   L* p=mem_;
   Z w1=w(),h1=h();
   V<V<ColRef> > res(h1,V<ColRef>(w1,LEAN),0);
   for (Z i=h1-1;i>=0;--i){
      for (Z j=0;j<w1;++j){
         L r=*p++;
         L g=*p++;
         L b=*p++;
         res[i][j]=ColRef(r,g,b);
      }
   }
   return res;
}

void Rec::vis()const
// screen representation of a linear block of pixel values.
// Basicly this is all we need from OpenGL
{
#if !defined(CPM_NOGRAPHICS)
//   Z mL=1;
//   Word loc("Rec::vis()");
//   CPM_MA
   Z h1=rec_.h();
   Z w1=rec_.w();
   glRasterPos2i(rec_.x1,rec_.y1+h1);
   glDrawPixels(w1,h1,GL_RGB,GL_UNSIGNED_BYTE,mem_);
   glFlush();
   if (Rec::db) glutSwapBuffers();
      // since 2020-05-02 we enable double buffered
//   CPM_MZ
#endif
}

////////////////////// class ColRef ////////////////////////

Z ColRef::lowerLimit=0;
Z ColRef::upperLimit=255;
   // no surprises

Z ColRef::x(Z z)
{
   if (z>ColRef::upperLimit) return ColRef::upperLimit;
   else if (z<ColRef::lowerLimit) return ColRef::lowerLimit;
   else return z;
}

bool ColRef::prnOn(ostream& str)const
{
   cpmwt("ColRef");
   cpmp(r);
```

```
   cpmp(g);
   cpmp(b);
   return true;
}

bool ColRef::scanFrom(istream& str)
{
   cpms(r);
   cpms(g);
   cpms(b);
   return true;
}

Z ColRef::com(ColRef const& obj)const
{
   cpmc(r);
   cpmc(g);
   cpmc(b);
   return 0;
}

//////////////////// class rgb ////////////////////////////

Z CpmGraphics::rgb::mixingRule(Z z1, Z z2, R w1, R w2, R gamma)
{
   if (gamma==0)
      return ColRef::upperLimit;
   else if (gamma==1)
      return cpmtoz(w1*z1+w2*z2);
   else if (gamma==2)
      return cpmtoz(sqrt(w1*z1*z1+w2*z2*z2));
   else if (gamma>0 && gamma<10){
      R p1=w1*cpmpow(R(z1),gamma);
      R p2=w2*cpmpow(R(z2),gamma);
      R gammaInv=1./gamma;
      return cpmtoz(cpmpow(p1+p2,gammaInv));
   }
   else
      return (z1>=z2 ? z1 : z2);
}

rgb CpmGraphics::rgb::mix(rgb c1, rgb c2, R w1, R w2, R gamma)
{
   Z a=mixingRule(c1.r,c2.r,w1,w2,gamma);
   Z b=mixingRule(c1.g,c2.g,w1,w2,gamma);
   Z c=mixingRule(c1.b,c2.b,w1,w2,gamma);
   rgb res(a,b,c);
   res.normalize();
   return res;
}
```

```
////////////////////////// class Viewport ////////////////////////////

// common initializations of static variables

ColRef CpmGraphics::Viewport::colText=ColRef();
R CpmGraphics::Viewport::gamma=2.3;
// best guess from a Kodak tool GAMMA.TIF for my office monitor
// is 2.3, also for my TFT-Monitor this seems to be the best
xyxy CpmGraphics::Viewport::window;
xyxy CpmGraphics::Viewport::fullWindow;
Z CpmGraphics::Viewport::size=1;
Z CpmGraphics::Viewport::rank=1;

#if !defined(CPM_NOGRAPHICS)
using CpmGraphics::Font;
bool CpmGraphics::Viewport::initialized=false;
Z CpmGraphics::Viewport::applWindowWidth=0;
Z CpmGraphics::Viewport::applWindowHeight=0;
Z CpmGraphics::Viewport::fullWindowHeight=0;

namespace{

Z getX()
{
   GLint x4[4];
   glGetIntegerv(GL_CURRENT_RASTER_POSITION,x4);
   return x4[0];
}

} // namespace

CpmGraphics::Font::Font(Z type)
{
   if (type==1){
      name=GLUT_BITMAP_TIMES_ROMAN_10;
      h=10;
   }
   else if (type==2){
      name=GLUT_BITMAP_HELVETICA_10;
      h=10;
   }
   else if (type==3){
      name=GLUT_BITMAP_HELVETICA_12;
      h=12;
   }
   else if (type==4){
      name=GLUT_BITMAP_8_BY_13;
      h=13;
   }
```

```
   else if (type==5){
      name=GLUT_BITMAP_9_BY_15;
      h=15;
   }
   else if (type==6){ // GLUT_BITMAP_HELVETICA_18 is very poorly designed
      // or implemented and should not be used
      // name=GLUT_BITMAP_HELVETICA_18;
      // h=18;
      name=GLUT_BITMAP_9_BY_15;
      h=15;
   }
   else {
      name=GLUT_BITMAP_TIMES_ROMAN_24;
      h=24;
   }
   d=cpmrnd(h*0.2);
   mw=cpmrnd(h*0.6); // as in 9 by 15 font
      // only an approximation - needed since glutBitmapWidth(...)
      // couldn't be made working so far
}

CpmGraphics::Font::Font(R fh)
{
   if (fh<=10.0){
      name=GLUT_BITMAP_HELVETICA_10;
      h=10;
   }
   else if (fh<=11.0){
      name=GLUT_BITMAP_TIMES_ROMAN_10;
      h=10;
   }
   else if (fh<=12.5){
      name=GLUT_BITMAP_HELVETICA_12;
      h=12;
   }
   else if (fh<=14.0){
      name=GLUT_BITMAP_8_BY_13;
      h=13;
   }
   else if (fh<=16.5){
      name=GLUT_BITMAP_9_BY_15;
      h=15;
   }
   else if (fh<=21.0){ // see comment to Font(6)
   //   name=GLUT_BITMAP_HELVETICA_18;
    //  h=18;
      name=GLUT_BITMAP_9_BY_15; // experiment due to L'
      h=15;
   }
   else if (fh<=24.5) {
```

```
      name=GLUT_BITMAP_TIMES_ROMAN_24;
      h=24;
   }
   else{
      name=GLUT_BITMAP_TIMES_ROMAN_24;
      h=24;
   }
   d=cpmrnd(h*0.2);
   mw=cpmrnd(h*0.6); // as in 9 by 15 font
      // only an approximation - needed since glutBitmapWidth(...)
      // couldn't be made working so far
}

// basic Cpm graphics library should be independent from CpmSystem and
// thus not to have to include cpmsystem.h

xyxy CpmGraphics::Viewport::seg1;
xyxy CpmGraphics::Viewport::seg2;
xyxy CpmGraphics::Viewport::seg3;
xyxy CpmGraphics::Viewport::seg4;

// background colors of the panes on the status bar
ColRef CpmGraphics::Viewport::cs1(L(255),L(255),L(50));
ColRef CpmGraphics::Viewport::cs2(L(210),L(255),L(110));
ColRef CpmGraphics::Viewport::cs3(L(180),L(180),L(190));
ColRef CpmGraphics::Viewport::cs4(L(165),L(165),L(255));

void Viewport::fill(xyxy r, ColRef cr)
{
   glColor3f(cr.glR(),cr.glG(),cr.glB());
   glRecti(r.x1,r.y1,r.x2+1,r.y2+1);
   glFlush();
}

Viewport::Viewport(Z w, Z h, const Word& title)
{
      if (initialized) return;
      Word loc("Viewport::Viewport(Y,Y,Word)");
#if defined(CPM_USE_MPI)
      size=CpmMPI::Cpm_com.getSize();
      rank=CpmMPI::Cpm_com.getRank();
#else
      size=1;
      rank=1;
#endif
      ColRef crBac(L(230),L(255),L(180));
      applWindowWidth=w;
      fullWindowHeight=h;
      fullWindow=xyxy(xy(0,0),applWindowWidth,fullWindowHeight);
      applWindowHeight=fullWindowHeight-heightStatusBar;
```

```
      window=xyxy(xy(0,0),applWindowWidth,applWindowHeight);
      if (rank==1){
         Word mes="OpenGL window got for applWindowWidth = ";
         mes&=cpm(applWindowWidth);
         mes&=" , applWindowHeight= ";
         mes&=cpm(applWindowHeight);
         Word mes2="Platform is "&Platform;
         cout<<mes<<endl;
         cout<<mes2<<endl;
      }
      R cp1=Message::pane1(); // from cpmsystemdependencies
      R cp2=Message::pane2();
      R cp3=Message::pane3();
      R cp4=Message::pane4();
      R sumPanes=cp1+cp2+cp3+cp4;
      if (sumPanes==0){
         cp1=2;
         cp2=2;
         cp3=1;
         cp4=1;
         sumPanes=cp1+cp2+cp3+cp4;
      }
      R awsp=applWindowWidth/sumPanes;
      Z w1=cpmrnd(awsp*cp1);
      Z w2=cpmrnd(awsp*cp2);
      Z w3=cpmrnd(awsp*cp3);
      Z w4=applWindowWidth-w1-w2-w3;
      Z yL=applWindowHeight;
      Z xL=0;
      seg1=xyxy(xy(xL,yL),w1,heightStatusBar);
      xL+=w1;
      seg2=xyxy(xy(xL,yL),w2,heightStatusBar);
      xL+=w2;
      seg3=xyxy(xy(xL,yL),w3,heightStatusBar);
      xL+=w3;
      seg4=xyxy(xy(xL,yL),w4,heightStatusBar);
      initialized=true;
      xy p0;
      xyxy all(p0,applWindowWidth,applWindowHeight);
      fill(all,crBac);
      R fontHeight=14;
      Font font(fontHeight);
      Z tx=font.getWidth();
      Z ty=font.getHeight();
      ColRef crText(L(0),L(0),L(255));
      textLine(tx,ty,title,crText,fontHeight);
}

void Viewport::onStatusBar(const Word& w, Z i)
{
```

```
   static Z th=cpmrnd(0.75*heightStatusBar);
   R fH=12.0;
   static Font font(fH); // Helvetica 12
   if (!isInitialized() ) return;
   ColRef cr;
   xyxy seg;
   if (i==1){
      cr=cs1;
      seg=seg1;
   }
   else if (i==2){
      cr=cs2;
      seg=seg2;
   }
   else if (i==3){
      cr=cs3;
      seg=seg3;
   }
    else if (i==4){
      cr=cs4;
      seg=seg4;
   }
   fill(seg,cr);
   Z shift=2;
   placeWord(seg.x1+shift, seg.y1+th, w, font, seg.w()-shift);
 }

void Viewport::clrPan()
{
   fill(seg1,cs1);
   fill(seg2,cs2);
   fill(seg3,cs3);
   fill(seg4,cs4);
}

void Viewport::placeWord(Z tx, Z ty, const Word& w, const Font& f,
   Z wMax)
{
   glColor3f(colText.glR(),colText.glG(),colText.glB());
   void* FONT=f.getName();
   Z x=tx,y=ty;
   Z xLim=tx+wMax;
   Z xM=f.getWidth();
   xLim-=xM;
   Z n=w.dim();
   glRasterPos2i(x,y);
   for (Z i=1;i<=n;++i){
      glutBitmapCharacter(FONT,w[i]);
      x=getX();
      if (x>xLim) break;
```

```
   }
   glFlush();
}

xyxy Viewport::textLine(Z tx, Z ty, const Word& w,
      ColRef crText, R h)
      // writing w on a line previously filledwith crBac
{
   glColor3f(colText.glR(),colText.glG(),colText.glB());
   Font f(h);
   void* FONT=f.getName();
   Z x=tx,y=ty;
   Z xM=f.getWidth();
   Z yM=f.getHeight();
   Z decender=(Z)(0.25*yM+0.5);
   Z i,n=w.dim();
   Z more=2; // makes the background rectangle a bit larger than the text
   glColor3f(crText.glR(),crText.glG(),crText.glB());
   glRasterPos2i(x,y);
   for (i=1;i<=n;++i){
       glutBitmapCharacter(FONT,w[i]);
   }
   Z txF=getX();
   glFlush();
   return xyxy(tx-more,ty-yM+decender-more,txF+more,ty+decender+more);
 }

xyxy Viewport::addText(Z x, Z y, Word const& line,
   R fontHeight, Z fontNumber)
{
   fontNumber; // not used
   return textLine(x,y,line,colText,fontHeight);
}

#else
// e.g. if CPM_NOGRAPHICS is defined
// trivial implementation; since then graphics
// works through a RAM-based data structure Img24

bool CpmGraphics::Viewport::initialized=true;

Viewport::Viewport(Z w, Z h, const Word& title){}
void Viewport::onStatusBar(const Word& w, Z i){}

Z CpmGraphics::Viewport::applWindowWidth=10;
Z CpmGraphics::Viewport::applWindowHeight=10;
Z CpmGraphics::Viewport::fullWindowHeight=10;

void CpmGraphics::Viewport::placeWord(Z tx, Z ty, const Word& w,
   const Font& f, Z wMax){tx;ty;w;f;wMax;}
```

```
void CpmGraphics::Viewport::fill(xyxy r, ColRef cr){r;cr;}

xyxy Viewport::textLine(Z tx, Z ty, const Word& w,
      ColRef crText, R h)
{
   tx;ty;w;crText;h;
   return xyxy();
}

xyxy Viewport::addText(Z x, Z y, Word const& text, R size, Z fontNumber)
{
   x;y;text;size;fontNumber;
   return xyxy();
}

void Viewport::clrPan()
{}

#endif
```

40 cpmvo.h

```
//? cpmvo.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_VO_H_
#define CPM_VO_H_
/*
   Purpose: Defining a array class Vo<T> which exploits the assumption\
    that
   T supports ordering. See cpmv.h

*/
#include <cpmv.h>
#include <algorithm>

namespace CpmArrays{

////////////////////////// class Vo ////////////////////////////////////

template <class T>
   // We assume that T provides (explicitly or implicitly)
   // copy constructor, and assignment or that T is a built-in type.
   // T < T and T > T assumed.

class Vo : public V<T>{ // version of V with order-related operations
   // derivation which implements the order interface

   typedef Vo<T> Type;
   typedef V<T> Base;
public:

explicit Vo(Z n=0):V<T>(n){}

explicit Vo(IvZ ivz):V<T>(ivz){}
```

```
   Vo(IvZ ivz, T const& t):V<T>(ivz,t){}

// constructors from explicit lists
explicit  Vo(std::initializer_list<T> il ):V<T>(il){}
   // requires C++11
      // constructors from explicit lists such as
      // Vo<Z> v{1,2,4,8};

   Vo(Z n, T const& t):V<T>(n,t){}

   Vo(const V<T>& h):V<T>(h){}
         // 'down-cast'-constructor

   virtual V<T>* clone(void)const{ return new Vo(*this);}

   virtual Word nameOf()const
      //: name of
   {
      Word wi="Vo<";
      return wi&CpmRoot::Name<T>()(T())&">";
   }

   V<Z> permutationForIncreasingOrder(void)const;
      // Returned is an array per such that i<j enforces
      //  (*this)[per[i]]<(*this)[per[j]]
      //       or
      // (*this)[per[i]]==(*this)[per[j]]
      // Assumes that valid indexing starts with 1 (i.e. b()==1)

   V<Z> permutationForDecreasingOrder(void)const
      // analogous to previous function
   { return permutationForIncreasingOrder().rev();}

   void order(void);
      // Permutes the indexes such that
      // (*this)[i]<(*this)[j] || (*this)[i]==(*this)[j]
      // whenever i<j. The components (instances of T) are allowed to
      // be large objects since excessive copying is avoided.

   Vo<T>& sort_(void){ order(); return *this;}
      //: sort

   void sortInc_();

   V<Z> perForDecOrd(void)const
      //: permutation for decreasing order
   { return permutationForDecreasingOrder();}

   V<Z> perForIncOrd(void)const
```

```
   //: permutation for increasing order
{ return permutationForIncreasingOrder();}

Vo<T> increasingCopy(void)const;
   // returns a increasingly ordered version of (*this) without
   // changing *this

Vo<T> incCopy(void)const{ return increasingCopy();}
   //: increasing copy

Vo<T> strictlyIncreasingVersion(void)const;
   // returns a stricly increasingly ordered version of (*this)
   // which leaves out multiply appearing components
   // (again, without changing *this)

Vo<T> strIncVer(void)const{ return strictlyIncreasingVersion();}
   //: strictly increasing version

Vo<T> decreasingCopy(void)const;
   // returns a decreasingly ordered version of (*this) without
   // changing *this

Vo<T> decCopy(void)const{ return decreasingCopy();}
   //: decreasing copy

Z find(T const&, bool ordered=true)const;
   //: find
   // if the return value is 0, there is no i=1,...n
   // such that (*this)[i]==t. Otherwise, the return value i is
   // such that (*this)[i]==t.
   // If ordered==true, this assumes that (*this) is ordered in the
   // way achieved by function order(), else no assumptions on
   // ordering. The components are compared against t starting from 1
   // (to dim()). The first occurrence of t gets reported.

Vo<T> purge(const V<T>& s)const;
   // returned is a list which is obtained from *this by eliminating
   // all components which equal to one of the components of s. Beside
   // of this removal, the order of the components in *this is
   // retained.

Vo<T> pur(const V<T>& s)const{ return purge(s);}
   //: purge

Z locate(T const&)const;
   // returned is a value j such that t satisfies
   //    (*this)[j] <= x < (*this)[j+1].
   // (*this) must be monotonic increasing
   // j=b()-1 or j=e()+1 is returned to indicate that t is out of
   // range.
```

```
   Z loc(T const& t)const{ return locate(t);}
      //: locate

   Vo<T> enqueue(T const&, bool asc=true)const;
      // assumes that (*this) is monotonic. Then t will be inserted in
      // the orderly correct position into of a copy of (*this). This copy
      // remains monotonic and gets returned.
      // The second argument comes only in action for dim()==1, since
      // then no direction is provided by *this and a direction is needed
      // for the result (for dim()==0, the result has dimm()==1 and no
      // direction).

   Vo<T> enq(T const& t, bool asc=true)const{ return enqueue(t,asc);}
      //: enqueue

   T sup()const;
      //: supremum
      // returns the largest component of *this; faster than getting the
      // largest element by ordering

   T inf()const;
      //: infimum
      // returns the smallest component of *this; faster than getting the
      // largest element by ordering

   Z indSup()const;
      //: index (of) supremum
      // returns the smallest i for which (*this)[i]=(*this).sup()

   Z indInf()const;
      //: index (of) infimum
      // returns the smallest i for which (*this)[i]=(*this).inf()

   T2<Z> indInfSup()const;
      //: returns indInf(), indSup() as a pair in a single pass

   T2<T> infSup()const;
      //: infimum supremum
      // Let res be the return value, then res.first==inf(),
      // res.second==sup(). Fast single-pass operation
};


// Function indexx from Press et al. changed into a function template
// p. 339  (role of M: see p. 333; algorithm is based on Quicksort, see
// sort(), anyway the choice of M may influence performance but never
// puts a limitation on the size of the vector to be ordered.

namespace{
```

```
   inline void swapZ(Z& i, Z& j){ Z ii=i; i=j; j=ii;}
}

template <class T>
V<Z> Vo<T>::permutationForIncreasingOrder(void)const
// Indexes an array v[1..n], i.e. outputs the array indx[1..n] such that
// v[indx[j]] is in ascending order for j=1,2,...,n. The input
// quantities n and v are not changed. Especially, v-elements are
// not copied back and forth which is essential if T is a large class
{
   const Z mL=4;
   cpmmessage(mL,"Vo<T>::permutation...(): started");
   const Z M=7;
   const Z NSTACK=50;

   Z nVec=Base::dim();
   Vo<Z> indx(nVec);
   Z i,indxt,ir=nVec,j,k,l=1,jstack=0;
   T a;

   Vo<Z> istack(NSTACK);

   for (j=1;j<=nVec;j++) indx[j]=j;
   for (;;) {
      if (ir-l < M) {
         for (j=l+1;j<=ir;j++) {
            indxt=indx[j];
            a=(*this)[indxt];
            for (i=j-1;i>=1;i--) {
               if ((*this)[indx[i]] < a) break;
               if ((*this)[indx[i]] == a) break;
               indx[i+1]=indx[i];
            }
            indx[i+1]=indxt;
         }
         if (jstack == 0) break;
         ir=istack[jstack--];
         l=istack[jstack--];
      } else {
         k=(l+ir)/2; // was >> 1 instead of /2 before
         swapZ(indx[k],indx[l+1]);
         if ((*this)[indx[l+1]] > (*this)[indx[ir]]) {
            swapZ(indx[l+1],indx[ir]);
         }
         if ((*this)[indx[l]] > (*this)[indx[ir]]) {
            swapZ(indx[l],indx[ir]);
         }
         if ((*this)[indx[l+1]] > (*this)[indx[l]]) {
            swapZ(indx[l+1],indx[l]);
         }
```

```
         i=l+1;
         j=ir;
         indxt=indx[l];
         a=(*this)[indxt];
         for (;;) {  // the i<nVec and j>1 test added by UM, otherwise in
               // patological cases of < and > relations we get a range
               // error
            do i++; while (i<nVec && (*this)[indx[i]] < a);
            do j--; while (j>1 && (*this)[indx[j]] > a);
            if (j < i) break;
            swapZ(indx[i],indx[j]);
         }
         indx[l]=indx[j];
         indx[j]=indxt;
         jstack += 2;
         if (jstack > NSTACK) cpmerror("NSTACK too small in ordering()");
         if (ir-i+1 >= j-l) {
            istack[jstack]=ir;
            istack[jstack-1]=i;
            ir=j-1;
         } else {
            istack[jstack]=j-1;
            istack[jstack-1]=l;
            l=i;
         }
      }
   }
   cpmmessage(mL,"Vo<T>::permutation...(): done");
   return indx;
}

// Function locate from Press et al. changed into a function template

//template <class T>
//Z Vo<T>::locate(const T& x)const
//{
//   Z mL=3;
//   Word loc("Z Vo<T>::locate(const T& x)const");
//   CPM_MA
//   Z jl=Base::b()-1;
//   Z ju=Base::e()+1;
//   bool ascnd=(Base::last() > Base::fir());
//   while (ju-jl > 1) {
//      Z jm=(ju+jl)/2;// Press et al. have >>1 istead of /2
//      if ((x > (*this)[jm]) == ascnd)
//         jl=jm;
//      else
//         ju=jm;
//   }
//   CPM_MZ
```

```
//   if (jl<Base::e()&& x==(*this)[jl+1]) return jl+1;
//      // Press et al. only assert that "x is between (*this)[j] and
//      // (*this)[j+1]" if j is the return value of the function
//      // So we have to specially treat the case that it is at the
//      // upper end
//   else return jl;
//}

template <class T>
Z Vo<T>::locate(T const& x)const
{
   Z mL=3;
   Word loc("Z Vo<T>::locate(const T& x)const");
   CPM_MA
   Z nc=Base::sz_;
   cpmassert(nc>0,loc);
   Z jl=Base::b()-1;
   Z ju=Base::e()+1;

   T xf=Base::fir();
   T xl=Base::last();
   if (x<xf){
      CPM_MZ
      return jl;
   }
   if (x>xl){
      CPM_MZ
      return ju;
   }
   T* q=std::upper_bound(&Base::p_[0],&Base::p_[nc],x);
   Z ir=Base::b()+(q-Base::p_)-1;
   CPM_MZ
   return ir;
}

// functions based on order relations in T

template <class T>
Vo<T> Vo<T>::increasingCopy(void)const
{
   Vo<Z> per=permutationForIncreasingOrder();
   Z i, n=Base::dim();
   Vo<T> y(n);
   for (i=1;i<=n;i++) y[i]=(*this)[per[i]];
   return y;
}

namespace{
   template <class T>
   bool fEqual(T const& t1, T const& t2){ return t1==t2;}
```

```
}

template <class T>
Vo<T> Vo<T>::strictlyIncreasingVersion(void)const
{
   Vo<T> res=increasingCopy();
   return res.condense(fEqual).c1();
}

template <class T>
Vo<T> Vo<T>::decreasingCopy(void)const
{
   Vo<Z> per=permutationForIncreasingOrder();
   Z i, n=Base::dim();
   Vo<T> y(n);
   for (i=1;i<=n;i++) y[1+n-i]=(*this)[per[i]];
   return y;
}

template <class T>
void Vo<T>::sortInc_()
{
   Base::cow_();
   std::sort(&Base::p_[0], &Base::p_[Base::sz_]);
}

//template <class X>
//void V<X>::sortInc_(B b)
//{
//   class cx{ // compare X
//      Z z_;
//   public:
//      cx(B bb):z_(bb ? -1 : 1){}
//      bool operator()(X const& x1, X const& x2 )
//      { return x1.c(x2)==z_;}
//   };
//   cx cp(b);
//   cow_();
//   std::sort(&p_[0], &p_[nc()],cp);
//}

template <class T>
void Vo<T>::order(void){ sortInc_();}
//{
//   V<Z> per=permutationForIncreasingOrder();
//   Z i, n=Base::dim();
//   Vo<T> y(n);
//   for (i=1;i<=n;i++) y[i]=(*this)[per[i]];
//   for (i=1;i<=n;i++) (*this)[i]=y[i];
//}
```

```
//template <class T >
//Z Vo<T>::find(T const& t, bool ordered)const
//{
//   Z n=Base::dim();
//   if (n==0) return 0;
//   if (ordered){
//      // fast method
//      Z jl=0, jm, ju=n+1;
//      while (ju-jl > 1) {
//         jm=(ju+jl)/2; // was >> 1 instead of /2 before
//         if (t < (*this)[jm])
//            ju=jm;
//         else
//            jl=jm;
//      }
//      if (jl<=n){
//         if (jl<n && t==(*this)[jl+1]) return jl+1;
//         if (jl>0 && t==(*this)[jl]) return jl;
//      }
//      return 0;
//   }
//   else{
//      Z iRes=0; // correct result if the next loop doesn't find t
//      for (Z i=1;i<=n;i++){
//         if ((*this)[i]==t){ iRes=i; break;}
//      }
//      return iRes;
//   }
//}

template <class T >
Z Vo<T>::find(T const& t, bool ordered)const
{
   cpmassert(ordered==true,"Z Vo<T>::find(T,bool)");
   Z n=Base::dim();
   if (n==0) return 0;
   Z nc=Base::sz_;
   T* q=std::find(&Base::p_[0],&Base::p_[nc],t);
   if (q==&Base::p_[nc]) return 0;
   return Base::b()+(q-Base::p_);
}

template <class T >
Vo<T> Vo<T>::purge(const V<T>& s)const
{
   Z i,j,n=Base::dim(),ns=s.dim();
   V<bool> vs(n,true);
   for (i=1;i<=n;i++){
      T t=(*this)[i];
```

```
      for (j=1;j<=ns;j++){
         if( t==s[j] ){ vs[i]=false; break; }
      }
   }
   return Base::select(vs);
}

template <class T >
T Vo<T>::sup()const
{
   Z n=Base::dim();
   cpmassert(n>0,"T Vo<T>::sup()const");
   T t,res=Base::fir();
   for (Z i=Base::b()+1;i<=Base::e();i++){
      t=Base::cui(i);
      if (t>res) res=t;
   }
   return res;
}

template <class T >
Z Vo<T>::indSup()const
{
   Z n=Base::dim();
   cpmassert(n>0,"Vo<T>::indSup()const");
   T t,valMax=(*this)[1];
   Z res=1;
   for (Z i=2;i<=n;i++){
      t=(*this)[i];
      if (t>valMax){ valMax=t; res=i;}
   }
   return res;
}

template <class T >
T Vo<T>::inf()const
{
   Z n=Base::dim();
   cpmassert(n>0,"T Vo<T>::inf()const");
   T t,res=Base::fir();
   for (Z i=Base::b()+1;i<=Base::e();i++){
      t=Base::cui(i);
      if (t<res) res=t;
   }
   return res;
}

template <class T >
Z Vo<T>::indInf()const
// needed as fast as possible
```

```
{
   Z n=Base::dim();
   cpmassert(n>0,"Z Vo<T>::indInf()const");
   Z res=0;
   T t,valMin=Base::cui(res);

   for (Z i=1;i<n;i++){
      t=Base::cui(i);
      if (t<valMin){ valMin=t; res=i;}
   }
   return ++res;
}

template <class T >
T2<Z> Vo<T>::indInfSup()const
{
   Z n=Base::dim();
   if (n<=0) cpmerror("Vo<T>::indInfSup(): dim()<=0");
   T t,valMin=(*this)[1],valMax=valMin;
   Z resMin=1;
   Z resMax=1;
   for (Z i=2;i<=n;i++){
      t=(*this)[i];
      if (t<valMin){ valMin=t; resMin=i;}
      if (t>valMax){ valMax=t; resMax=i;}
   }
   return T2<Z>(resMin,resMax);
}

/*********************************
template <class T >
T2<T> Vo<T>::infSup()const
{
   Z n=Base::dim();
   if (n<=0) cpmerror("Vo<T>::infSup(): dim()<=0");
   T t,valMin=(*this)[1],valMax=valMin;
   for (Z i=2;i<=n;i++){
      t=(*this)[i];
      if (t<valMin){ valMin=t; }
      if (t>valMax){ valMax=t; }
   }
   return T2<T>(valMin,valMax);
}
*********************************/

template <class T >
T2<T> Vo<T>::infSup()const
{
   Z n=Base::dim();
   if (n<=0) cpmerror("Vo<T>::infSup(): dim()<=0");
```

```
   T t,valMin=Base::fir(),valMax=valMin;
   for (Z i=Base::b()+1;i<=Base::e();i++){
      t=Base::cui(i);
      if (t<valMin){ valMin=t; }
      if (t>valMax){ valMax=t; }
   }
   return T2<T>(valMin,valMax);
}

template <class T >
Vo<T> Vo<T>::enqueue(T const& t, bool asc)const
{
   Z mL=3;
   Word loc("Vo<T> Vo<T>::enqueue(T const& t, bool asc)const");
   CPM_MA
   Z d=Base::dim();
   if (d==0) return Vo<T>(1,t);
   if (d==1){
      Vo<T> res(2);
      T t1=(*this)[1];
      if (asc){
         if (t1<t){
            res[1]=t1;
            res[2]=t;
         }
         else{
            res[1]=t;
            res[2]=t1;
         }
      }
      else{
         if (t1<t){
            res[1]=t;
            res[2]=t1;
         }
         else{
            res[1]=t1;
            res[2]=t;
         }
      }
      return res;
   }
   Z i=locate(t);
   CPM_MZ
   return Base::ins(i+1,t);
}
////////////////////////// iterated templates ////////////////////////////
// describing multi-indexed quantities
////////////////////////////// class VVo /////////////////////////////////
// matrices
```

```
template <class T>
class VVo: public VV<T>{// version of VV with order-related operations

   typedef VV<T> Base;
   typedef VVo<T> Type;

public:
   CPM_ORDER
// constructors
   VVo(Z d1, Z d2):Base(d1,d2){}
   VVo( Z d1, Z d2, T const& t):Base(d1,d2,t){}
      // all components are equal to t
   VVo(void):Base(){}
   VVo(VV<T> const& x):Base(x){}
   VVo( V< V<T> > const& x):Base(x){}
};

template <class T >
Z VVo<T>::com(VVo<T> const& s)const
{
   Z d1=Base::dim(), d2=s.dim();
   if (d1<d2) return 1;
   else if (d1>d2) return -1;
   else{
      for (Z i=1;i<=d1;i++){
         Vo<T> ti=(*this)[i];
         Vo<T> si=s[i];
         Z ci=CpmRoot::comT<T>(ti,si);
         if (ci!=0) return ci;
      }
      return 0;
   }
}

////////////////////////// class VVVo ////////////////////////////////////
// tensors of rank 3

template <class T>
class VVVo: public VVV<T>{// version of VVV with order-related operations

   typedef VVV<T> Base;
   typedef VVVo<T> Type;

public:
   CPM_ORDER
// constructors
   VVVo(Z d1,Z d2,Z d3):Base(d1,d2,d3){}
   VVVo(Z d1,Z d2,Z d3,T const& t):Base(d1,d2,d3,t){}
      // all components are equal to t
```

```
   VVVo(void):Base(){}
   VVVo(VVV<T> const& x):Base(x){}
   VVVo( V< V< V<T> > > const& x):Base(x){}
};

template <class T >
Z VVVo<T>::com(VVVo<T> const& s)const
{
   Z d1=Base::dim(), d2=s.dim();
   if (d1<d2) return 1;
   else if (d1>d2) return -1;
   else{
      for (Z i=1;i<=d1;i++){
         VVo<T> ti=(*this)[i];
         VVo<T> si=s[i];
         Z ci=CpmRoot::comT<T>(ti,si);
         if (ci!=0) return ci;
      }
      return 0;
   }
}

////////////////////////// class VVVVo ////////////////////////////////
// tensors of rank 4

template <class T>
class VVVVo:public VVVV<T>{//version of VVVV with order-related operations

   typedef VVVV<T> Base;
   typedef VVVVo<T> Type;

public:
   CPM_ORDER
// constructors
   VVVVo(Z d1,Z d2,Z d3,Z d4):Base(d1,d2,d3,d4){}
   VVVVo(Z d1,Z d2,Z d3,Z d4,T const& t):Base(d1,d2,d3,d4,t){}
      // all components are equal to t
   VVVVo(void):Base(){}
   VVVVo(VVVV<T> const& x):Base(x){}
   VVVVo( V< V< V< V<T> > > > const& x):Base(x){}
};

template <class T >
Z VVVVo<T>::com(VVVVo<T> const& s)const
{
   Z d1=Base::dim(), d2=s.dim();
   if (d1<d2) return 1;
   else if (d1>d2) return -1;
   else{
      for (Z i=1;i<=d1;i++){
```

```
         VVVo<T> ti=(*this)[i];
         VVVo<T> si=s[i];
         Z ci=CpmRoot::comT<T>(ti,si);
         if (ci!=0) return ci;
      }
      return 0;
   }
}

} // namespace

#endif
```

41 cpmvr.h

```
//? cpmvr.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_VR_H_
#define CPM_VR_H_
/*

Purpose: Defining a vector template providing the full
functionality of the rich interface

*/
#include <cpmva.h>

////////////////////////////// class Vr<> ////////////////////////////////////

namespace CpmArrays{

   using CpmRoot::R;
   using CpmRoot::Root;

template <class T>
   // We assume that T provides (explicitely or implicitely)
   // copy constructor, and assignement
   // or that T is a built-in type.

class Vr: public Va<T>{ // version of Va with a rich interface

public:
   typedef Vr<T> Type;
   typedef Va<T> Base;
   typedef T ScalarType; // was missing till 2001-10-12
      // detected by the gnu compiler
```

```
   explicit Vr(Z n=0):Va<T>(n){}
      // note that Vr(0) is defined and is different from Vr()

   explicit Vr(IvZ ivz):Va<T>(ivz){}
      // note that Vr(0) is defined and is different from Vr()

   Vr(IvZ ivz, T const& t):Va<T>(ivz,t){}

   // constructors from explicit lists
   explicit  Vr(std::initializer_list<T> il ):Va<T>(il){}
   // requires C++11
      // constructors from explicit lists such as
      // Vr<Z> v{1,2,4,8};

   Vr(Z n, T const& t):Va<T>(n,t){}
      // initializes all n components with t

   Vr( V<T> const& h):Va<T>(h){}
      // 'down-cast'-constructor

   Vr( Vo<T> const& h):Va<T>(h){}
      // 'down-cast'-constructor

   Vr(Va<T> const& h):Va<T>(h){}
      // 'down-cast'-constructor

   virtual V<T>* clone(void)const{ return new Vr<T>(*this);}

   CPM_TEST
   CPM_DESCRIPTORS
      // dis from Va
};

// member functions

template <class T>
Word Vr<T>::nameOf(void)const
{
   Word w1="Vr<";
   Word w2=Root<T>(T()).nameOf();
   return w1&w2&">";
}

template <class T>
Word Vr<T>::toWord(void)const
{
   if (Base::isVoid()) return "void list";
   Word res="( ";
   Z i;
   for (i=Base::b();i<Base::e();i++){
```

```
      Word wi="("&Root<Z>(i).toWord()&","&Root<T>((*this)[i]).toWord()&"),\
       ";
      res&=wi;
   }
   i=Base::e();
   Word we="("&Root<Z>(i).toWord()&","&Root<T>((*this)[i]).toWord()&")";
   res&=we;
   res&=" )";
   return res;
}

template <class T>
Z Vr<T>::hash(void)const
{
   Z res=0;
   for (Z i=Base::b(); i<=Base::e(); i++){
      res+=Root<T>((*this)[i]).hash();
   }
   return res;
}

template <class T>
Vr<T> Vr<T>::test(Z cpl)const
{
   Z it=0;
   it=Root<Z>(it).test(cpl);
   T xt;
   xt=Root<T>(xt).test(cpl);
   return Vr<T>(it,xt);
}

template <class T>
Vr<T> Vr<T>::ran(Z j)const
   // all randomized versions of *this thus have the same dimension as
   // *this
{
   Vr<T> res(Base::dom());
   T tempX;
   if (j==0){
      for (Z i=Base::b(); i<=Base::e(); i++){
         tempX=(*this)[i];
         res[i]=Root<T>(tempX).ran(j);
      }
   }
   else{
      Z j0=1000; // this is done to avoid an obvious correlation
         // among the coordinates of x.ran(i)
         // and x.ran(i+1)
      Z jIncr=Root<Z>(j0).ran(j);
      Z jc=j;
```

```
      for (Z i=Base::b(); i<=Base::e(); i++){
         tempX=(*this)[i];
         jc+=jIncr;
         res[i]=Root<T>(tempX).ran(jc);
      }
   }
   return res;
}

// We do not implement the iterated templates since they are now
// endowed with sufficient functionality already as Va, VVa, VVVa,
// and VVVVa

} // namespace

#endif
```

42 cpmword.h

```
//? cpmword.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_WORD_H_
#define CPM_WORD_H_
/*
   Description: Declares class Word which is now essentially identical
      to string of the standard library. It is formed from it according
      to the advice in BS3, end of Chapter 20.3.

      Word is a minimal string class to which Cpm classes refer for
         type identification and for messaging
*/
#include <cpminterfaces.h>
#if !defined(CPM_NAMEOF)
   #include <typeinfo>
#endif

namespace CpmRoot{

   using namespace CpmStd;

   class Word;

   Word toWord(R x, Z prc);

   Word toWord(unsigned int i, const char* ff);
      // ff standard format string
   Word toWord(unsigned long int i, const char* ff);
   Word toWord(int i, const char* ff);
   Word toWord(long int i, const char* ff);
   Word toWord(R r, const char* ff);
```

```
   ofstream& to_ofstream(Word const& w);
   ifstream& to_ifstream(Word const& w);
   string toString(Word const& w);

// concatenation of words described by operator &
   Word operator &(Word const&, Word const&);
   Word operator &(const char* cp, Word const& w);
   Word operator &(Word const& w, const char* cp);
   Word operator &(char cp[], Word const& w);
   Word operator &(Word const& w, char cp[]);

// for convenience same is allowed to be denoted +
   Word operator +(Word const&, Word const&);
   Word operator +(const char* cp, Word const& w);
   Word operator +(Word const& w, const char* cp);
   Word operator +(char cp[], Word const& w);
   Word operator +(Word const& w, char cp[]);

////////////////////// class Word ////////////////////////////////////

   class Word{ //  wrapping charcter strings, rich functionaliy
      // Instances of Word are dynamically allocated strings of
      // characters like those of string (on which class the present
      // implementation is based). Input and Output works as expected only
      // if we have a word in the narrower sense that the character
      // string does not contain 'whitespace'.
      // Due to functions toBool(), toZ(), toR() a Word is a universal
      // carrier of numerical information. For instance the content
      // of a command line may be represented as a list of Words (an
      // instance of V<Word> in C+- )

   string rep_;

   static bool scnLin;
      // modifies the function scanFrom() in a way that also character
      // strings containing blanks can be read in to yield a single Word.
      // Such a facility is needed if Word is to be used as the data type
      // of names of more complex entities like organizations, theories,
      // etc.: Certainly 'set theory' and 'Eastman Kodak Company' should
      // be allowed names.

   static Z endOfWord; // end of Word
      // 0 nothing, 1 blank, else newline after w.prnOn()

   typedef Word Type;
   typedef Word CloneBase;

public:

 //  Word nameOf()const{ return "Wf";}
```

```
static void setScanLine(bool sl){ scnLin=sl;}
   //: set scan line
   // see private static data member scnLin
static bool getScanLine(){ return scnLin;}
   //: get scan line
   // see private static data member scnLin

static void setEndOfWord(Z i){ endOfWord=i;}
   //: set end of word
   // see private static data member endOfWord
static Z getEndOfWord(){ return endOfWord;}
   //: get end of word
   // see private static data member endOfWord

explicit Word(char c):rep_(1,c){}
// a Word consisting of just one character c

explicit Word(Z i, char c='a'):rep_(i,c){}
// a Word consisting of i copies of character c
// useful for getting new words by changing letters at given positions

Word(const char* charPtr):rep_(charPtr){}
   // construction of a Word from char*, and thus from literals
   // such as Word w("convenient programming")
   // Although this defines a conversion from const char* to
   // Word, in a statement  cout<<"This is ..."<<endl;  the
   // interpretation of "This is ..." as const char* applies,
   // since exact matches are preferred over matches which employ
   // user-defined conversions. This is important, since
   // cout<<Word("This is ...")<<endl; may append some separator
   // to the input string depending on the value of the
   // static data member endOfWord.

explicit Word(string const& s):rep_(s){}
   // construction of a Word from string

explicit Word(ostringstream const& ost):rep_(ost.str()){}
   // construction of a Word from a string stream
   // This is the most convenient way to create detailed texts held as
   // a Word, as in the following code fragment:
   // ostringstream ost;
   // ost<<...<<...endl<<...;
   // Word text(ost);

Word():rep_(){}
   // default constructor, creates the void word which consists of
   // 0 characters

const char* operator-(void)const{ return rep_.c_str();}
```

```
   // explicit conversion from Word to const char*
   // The result is owned by the Word and the user should not try to
   // delete it.

const char* c_str(void)const{ return rep_.c_str();}
   // to make Word and string as much alike as possible

string str()const{ return rep_;}
   //: string

string toStr()const{ return rep_;}
   //: to string

string std()const{ return rep_;}
   //. standard

Word skipChr(char c)const;
   //: skip character
   // Returns a Word which results from *this by eliminating c
   // everywhere.

void skipSpc_();
   //: skip space, mutating function
   // eliminates all characters which by the function isspace(char)
   // of <cctype> is classified as space

void skipSurSpc_();
   //: skip surrounding space
   // eliminates all space characters (whitespace) as above which
   // surround a chain of space separated traditional words.
   // So the result is a robust representation for a textual content
   // of a line of text, since unvisible parts at the beginning and
   // the end get eliminated
   // added 2004-10-12

Word firstWord()const;
   // returns the first whitespace separated part of the string rep_
   // (which may contain whitespace). It would be nice to
   // return a complete list of Words here, but we prefer not to
   // build on any array class in the present basic class.
   // The result may be the void Word.

Word firstWord_();
   // returns the first whitespace separated part of the string rep_
   // (which may contain whitespace) and changes *this into
   // the part of the word which remains.
   // The ending '_' is reminescent of Ruby's '!' as a indicator
   // for mutating functions. Calling this function in succession
   // allows to create a complete list of words in a line without
   // having a list- or array-class available at this point.
```

```
// we have to define later a general template function
// template <class T1, class T2>
// operator & (const T1&, const T2&)
// This implies that in (non-template !) functions operator & only exact
// matches will be accepted (see Stroustrup p. 589), certainly not the
// ones that can be achieved by userdefined conversions. So, we have to
// take care of char* and char[] in this place

// concatenation of words described by operator &
   Word& operator &=(Word const& w);
   Word& operator &=(const char* cp);
   friend Word operator &(Word const&, Word const&);
   friend Word operator &(const char* cp, Word const& w);
   friend Word operator &(Word const& w, const char* cp);
   friend Word operator &(char cp[], Word const& w);
   friend Word operator &(Word const& w, char cp[]);

// for convenience same is allowed to be denoted +. This is convenient
// since + binds stronger than & and especially stronger than <<
   Word& operator +=(Word const& w);
   Word& operator +=(const char* cp);
   friend Word operator +(Word const&, Word const&);
   friend Word operator +(const char* cp, Word const& w);
   friend Word operator +(Word const& w, const char* cp);
   friend Word operator +(char cp[], Word const& w);
   friend Word operator +(Word const& w, char cp[]);

// access functions
   Z dim(void)const{ return Z(rep_.length());}
      //: dimension
      // is 0 for the void Word
   Z b(void)const{ return 1;}
      // b for begin, first index of a letter
   Z e(void)const{ return dim();}
      // e for end, last index of a letter
      // A loop over the letters of Word w takes the form
      // for (Z i=w.b();i<=w.e();++i){ ... = w[i];}
      // This is correct also for the void word.
      // This looping style is also valid for all C+- array classes.
   Z size(void)const{ return dim();}
      //: size
   Z length(void)const{ return dim();}
      // for uniformity with std library types

   bool isVoid()const { return dim()==0;}
      //: is void
      //  short answer on whether the dimension is zero

   friend Z length(Word const& w){ return w.dim();}
```

```
      // number of characters (no addition for a terminator)

   char operator[](Z i)const;
      // save reading of characters. Valid indexes are 1...dim()

   char& operator[](Z i);
      // save overwriting of characters in a word.

   bool valInd(Z i)const{ return i>0 && i<=dim();}
      //: valid index
      // returns the validity of i as an index of *this

   Word cut(Z i)const;
      //: cut
      // returned is a Word which results from *this by removing i
      // components from the end. If i<0 we remove -i components from
      // the beginning.

// functions dealing Words as filenames
   Word resize(Z newDim)const;
      // returns a Word res such that res.dim()=newDim. If newDim is
      // smaller than dim(), the end of *this will be cut away.
      // If newDim is larger, blanks will be added.

   Word remExt(bool extended=false)const;
      //: remove extension
      // Intention of the function: returns what is left if any
      // file name extension (if there is one) is removed from
      // *this. So for *this=="myprog.ini" we return "myprog".
      // Actually we read and copy *this from the beginning until we
      // find a character which signifies the beginning of the extension.
      // In nomal mode this is a dot, in extended mode this is any capital
      // letter. The extended mode arrose from the need to distinguish
      // executables made by different compilers by a suffix such as
      // palaMS.exe, palaMinGW.exe, pala_ms.exe. Since the application
      // framework in cpmapplication.cpp deduces the program name,
      // and thus the name of the ini-file to search, from
      // the name of the executable, the above-mentioned executables would
      // try to read ini-files palaMS.ini, palaMinGW.ini, and
      // pala_ms.ini. They should, however, read simply pala.ini.
      // This is achieved by removing the extension from the executables
      // name for a true value of the argument.
      // The presently implemented notion of extension in 'extended mode'
      // is, that it is everything whatis not a lower-case letter or a
      // digit. This reflects .... obsolete

   Word remApp()const;
      //: remove appendix
      // Here, we remove the maximum contiguous string formed out of
      // capital letters and underscores at the end of *this. So name
```

```
      // appendices reflecting compilers or compiler options should be
      // chosen to consist of capital letters and/or underscores. So
      // not all of the functions in the previous function comment work:
      // Instead of palaMinGW, pala_ms one should use palaMINGW and
      // pala_MS

   Word appBefExt(Word const& app)const;
      //: append before extension
      // Word("myprog.ini").appBefExt("2") yields "myprog2.ini"

   Word baseName()const;
      //: base name
      // Only the characters right to the right-most "\" or "/"
      // are returned as a Word. If *this is a file name including
      // a directory path, then the actual file name gets extracted and
      // returned. (corresponds do File:basename of Ruby)

   bool readable()const;
      // returns true if *this is the name of a file that can be
      // opened and gives a valid istream to read from.
      // The stream will be closed after the test and not returned.

   Word makeFileName()const;
      //: make file name
      // returns a version of *this in which all characters which may not
      // be acceptable in file names are replaced by "_"
// end of functions dealing Words as filenames

   Word rev()const;
      //: reverse
      // returns the reversed version of *this, i.e. the one read from
      // the end

   Word& rev_(){ return *this=rev();}
      //: reverse
      // mutating form of reversal

   Word slc_(char c, bool firstSep=true);
      //: slice
      // If c does not occur in *this, we return the original *this and
      // change *this to Word("").
      // One could do it the other way round, then the implementation
      // of remExt in terms of the present function would need to handle
      // the case that no dot is present (as for executables in Linux)
      // would need special treatment.
      // Now the interesting case:
      // If c appears in *this, we fix the position where it
      // appears in the leftmost position for firstSep==true
      // and in the rightmost position else.
      // we return the part of the word which is left of c
```

```
      // and change *this to the part right of c. Then we have for
      // Word w=...;
      // Word wMem=w;
      // char c=...; // such that c occurs in w !
      // Word wl=w.slc_(c);
      // the property
      // wl&Word(c)&w==wMem;

   Word tail(Z i)const;
      //: tail
      // if i>=dim() we return *this. Else we remove as many components
      // from the beginning that only i remain (' the last i components of
      // the original Word') and return the result.

   Z find(char c)const;
      //: find
      // returns the first position (1.....dim()) of a c in the
      // Word *this. If the character c isn't there we get 0

    Z find(Word const& w)const;
      //: find
      // returns the first position (1.....dim()) of a
      // subword w in the Word *this. If the Word w isn't there
      // we get 0

   static Word write(R n, Z fieldLength=0);
   static Word write(Z n, Z fieldLength=0);
   static Word write(N n, Z fieldLength=0);
        // If fieldLength=0 the natural length of the number is fully
      // outputted. Else the length is casted or extended to fieldLength.
   static Word write(string h);
   static Word write(bool b);

   static Word ascii();
      // All ASCII characters implemented in in the fonts
      // used by CpmGraphics::Frame. These characters are concatenated
      // in a Word. Helpful for font inspection and development.

   R toR()const;
      // outputs a R obtained by converting *this
   Z toZ()const;
      // outputs a Z obtained by converting *this
   bool toBool()const;
      // outputs a bool obtained by converting *this

// these declarations are repeated outside the class
// and have only influence on the implementation

   friend Word toWord(unsigned int i, const char* ff);
   friend Word toWord(unsigned long int i, const char* ff);
```

```
   friend Word toWord(int i, const char* ff);
   friend Word toWord(long int i, const char* ff);
   friend Word toWord(R r, const char* ff);

   friend ofstream& to_ofstream(Word const& w);
   friend ifstream& to_ifstream(Word const& w);
   friend string toString(Word const& w);

   bool readLine(istream& in){return CpmRoot::readLine(rep_,in);}
      // making *this equal to the content of the next non-comment
      // line in stream in

   CPM_IO
      // scanFrom(in);  makes *this equal to the next noncomment
      // whitespace separated sequence of characters in stream in.
      // Thus the code
      //
      //    Word w("This is a word");
      //    w.prnOn(aStream);
      //    Word w2;
      //    w2.scanFrom(aStream)
      //
      // will yield w2=="This"
      // and not w2=="This is a word"! This is sometimes what one wants,
      // sometimes not.
   CPM_ORDER
   CPM_TEST_ALL // implementation borrowed from string
// Declarations of CPM_DESCRIPTORS together with implementation
   Word toWord()const{ return *this;}
   Word nameOf()const{ return "Word";}
};

inline string toString(Word const& w){ return w.rep_;}

#define cpmwrite CpmRoot::Word::write
#define cpm CpmRoot::Word::write

// conversion of selected basic types to string
// as a basis for doing the conversion to Word in a consistent
// manner.
// The parameter prc (precision) makes no sense for integer types
// Most conversions from built-in types to Word, which have not
// a char*-typed formatting string as one argument, are based on these
// toStr-functions. An exception are the functions write which have
// precisely defined writing field length which is needed for writing
// e.g. to tables or quietly on the status bar.
// prc<=0 means that no precision gets set in the code, so that we then
// rely on the default precision set by the compiler.
   string toStr(Z x);
   string toStr(N x);
```

```
   string toStr(L x);
   string toStr(bool x);
   string toStr(R const& x, Z prc=0);
/*
In cpmnumbers.h the following list of interface service class
templates was defined
    IO<T>, Comp<T>, Inv<T>, AbsSqr<T>, Abs<T>, Dis<T>,
    Test<T>, Ran<T>, Hash<T>, Conj<T>, Neutrals<T>
and the following two completing members were annonced
    Name<T>, ToWord<T>.
These will be defined now:
*/
///////////////// class Name<> ////////////////////////////////////////
// We need a template class istead of a template function since we need
// specializations also for templates such as std::vector<> (see cpmv.h).
// Whether one has CPM_NAMEOF defined or not --- the C+- basic
// built-in types always have the C+- name (i.e 'R', not 'double')
// For the more complex types, the names created by typeinfo
// (which look acceptable) are used for CPM_NAMEOF not defined.

   template<class T>
   class Name{ // tool class template for defining the nameOf function
   public:
      Name(){}
      Word operator()(T const& t)const
      {
#if defined(CPM_NAMEOF)
         return t.nameOf();
#else
         return Word(typeid(t).name());
#endif
      }
   };

   template<>
   class Name<Z>{ // Z: integers
   public:
      Name(){}
      Word operator()(Z const& t)const
         { t; return Word("Z");}
   };

   template<>
   class Name<N>{ // N: natural numbers
   public:
      Name(){}
      Word operator()(N const& t)const
         { t; return Word("N");}
   };
```

```
   template<>
   class Name<R>{ // multiple precision real numbers
   public:
      Name(){}
      Word operator()(R const& t)const
         { t; return Word("R");}
   };

   template<>
   class Name<L>{ // L: letter
   public:
      Name(){}
      Word operator()(L const& t)const
         { t; return Word("L");}
   };

   template<>
   class Name<bool>{ // specialization
   public:
      Name(){}
      Word operator()(bool const& t)const
         { t; return Word("bool");}
   };

   template<>
   class Name<char>{ // specialization
   public:
      Name(){}
      Word operator()(char const& t)const
         { t; return Word("char");}
   };

   template<>
   class Name<string>{ // specialization
   public:
      Name(){}
      Word operator()(string const& t)const
         { t; return Word("string");}
   };

   template<class T>
   Word nameOf(T const& t){ return Name<T>()(t); }
      //: name of
   // convenient syntax for getting a name of the type of
   // a quantity. Definition is such that one may also give
   // Cpm-names to class templates. See cpmv.h
   // for treating template <class T> std::vector<T> in this way.

////////////////////// class ToWord<> ///////////////////////////////////
```

```
template<class T>
class ToWord{
public:
   ToWord(){}
   Word operator()(T const& t)const{ return t.toWord();}
};

template<>
class ToWord<R>{
public:
   ToWord(){}
   Word operator()(R const& t)const{ return Word(toStr(t));}
};

template<>
class ToWord<Z>{
public:
   ToWord(){}
   Word operator()(Z const& t)const{ return Word(toStr(t));}
};

template<>
class ToWord<N>{
public:
   ToWord(){}
   Word operator()(N const& t)const{ return Word(toStr(t));}
};

template<>
class ToWord<L>{
public:
   ToWord(){}
   Word operator()(L const& t)const{ return Word(toStr(t));}
};

template<>
class ToWord<bool>{
public:
   ToWord(){}
   Word operator()(bool const& t)const{ return Word(toStr(t));}
};

template<>
class ToWord<string>{
public:
   ToWord(){}
   Word operator()(string const& t)const{ return Word(t);}
};

template<class T>
```

```
   Word toWord(T const& t){ return ToWord<T>()(t); }
      //: to word
      // Presently not important, MPI-functionalty relies on
      // prnOn and scanFrom

   ////////////////////////////// class Root<> ////////////////////////////

   // A final step towards unification of infrastructure functions:
   // The Root class template provides member functions which the
   // argument type T may not define. Even if type T is only accessible
   // through its declaration, we may specialize the 13 basic
   // Root - related interface service class templates
   //
   //    IO<T>, Comp<T>, Inv<T>, AbsSqr<T>, Abs<T>, Dis<T>, Test<T>,
   //    Ran<T>, Hash<T>, Conj<T>, Neutrals<T>, ToWord<T>, Name<T>
   //
   // for this type T and thus make Root<T> equipped with these member
   // functions.
   // How this class simplifies the implementation of template classes
   // can be seen from the implementation of the interfaces
   // CPM_IO and CPM_ORDER in cpmv.h
   // Notice that in addition to the template argument type T, there are
   // the following types involved in the definition of Root<>:
   //
   // bool
   // std::istream
   // std::ostream
   // CpmRoot::Z
   // CpmRoot::R
   // CpmRoot::Word
   //
   // Notice also, that the true nature of CpmRoot::Z and CpmRoot::R
   // depends on the macros CPM_LONG and CPM_MP in cpmdefinitions.h.
   // This is an overview of all 26 member functions of Root<T>:
/*
      Root(T const& t = T());
      T operator()(void)const;
      bool prnOn(ostream& str)const;
      bool scanFrom(istream& str);
      Z com(T const& t)const;
      bool operator == ( T const& t)const;
      bool operator != (T const& t)const;
      bool operator < (T const& t)const;
      bool operator > (T const& t)const;
      bool operator <= (T const& t)const;
      bool operator >= (T const& t)const;
      T inf(T const& t)const;
      T sup(T const& t)const;
      Z hash()const;
      R absSqr()const;
```

```
      R abs()const;
      R dis(T const& t)const;
      T inv()const;
      T operator!()const;
      T net(Z i=0)const;
      T con()const;
      T operator~()const;
      T test(Z tvs)const;
      T ran(Z j=0)const;
      Word nameOf()const
      Word toWord()const
*/

   template<class T>
   class Root{ // Provides the basic functions for dealing with CpmRoot's
     // basic types as member functions (or operators) of a class
     // template.
      T a_;
   public:
      explicit Root(T const& t = T()):a_(t){}
      T operator()(void)const{ return a_;}
      bool prnOn(ostream& str)const{ return IO<T>().o(a_,str);}
         //: print on
         // we don't redefine ostream& << T
         // Root<T>(t).prnOn(out);
         // is the general idiom for writing a T-typed object to stream.
         // Notice that the I/O-macros cpmp(X) and cpms(X) are implemented
         // based on functions prnOnT and scanFromT and that class Root
         // can hardly be used for this purpose (macros are always
         // special).
      bool scanFrom(istream& str){ return IO<T>().i(a_,str);}
         //: scan from
         // we don't redefine istream& >> T
         // Root<T> tIn; tIn.scanFrom(in); T t = tIn();
         // is the general idiom for reading a T-typed object t from
         // stream.
         // A more compact form of reading t from str is available with
         // the following function template scanFrom(T&, istream&).
      Z com(T const& t)const{ return Comp<T>()(a_,t);}
         //: compare
      bool operator == ( T const& t)const{ return 0==com(t);}
      bool operator != (T const& t)const{ return 0!=com(t);}
      bool operator < (T const& t)const{ return 0 < com(t);}
      bool operator > (T const& t)const{ return 0 > com(t);}
      bool operator <= (T const& t)const{ return 0 <= com(t);}
      bool operator >= (T const& t)const{ return 0 >= com(t);}
      T inf(T const& t)const{ return 0 < com(t) ? a_ : t;}
         //: infimum
      T sup(T const& t)const{ return 0 > com(t) ? a_ : t;}
         //: supremum
```

```
      Z hash()const{ return Hash<T>()(a_);}
         //: hash
      R absSqr()const{ return AbsSqr<T>()(a_);}
         //: absolute (value) squared
      R abs()const{ return Abs<T>()(a_);}
         //: absolute (value)
      R dis(T const& t)const{ return Dis<T>()(a_,t);}
         //: distance
      T inv()const{return Inv<T>()(a_);}
         //: inverse
      T operator!()const{return Inv<T>()(a_);}
      T net(Z i=0)const{ return Neutrals<T>()(a_,i);}
         //: neutrals
      T con()const{ return Conj<T>()(a_);}
         //: conjugate
      T operator~()const{ return Conj<T>()(a_);}
      T test(Z tvs)const{ return Test<T>()(a_,tvs);}
         //: test, //. tvs: test value size ~ complexity
      T ran(Z j=0)const{ return Ran<T>()(a_,j);}
         //: random
      Word nameOf()const{ return Name<T>()(a_);}
         //: name of
      Word toWord()const{ return ToWord<T>()(a_);}
         //: to word
   };

   template<typename T>
   bool scanFrom(T& t, istream& str)
   {
      Root<T> rt(t);
      bool b=rt.scanFrom(str); // reading the 'value' of rt from str
      if (b) t = rt();
      return b;
   }

} // namespace

#define cpmnam CpmRoot::nameOf
#define cpmtow CpmRoot::toWord

#endif
```

43 cpmword.cpp

```
//? cpmword.cpp
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#include <cpmtypes.h>
#include <stdio.h>
#include <cctype>
#include <iomanip>

using namespace CpmStd;
using CpmRoot::Word;
using CpmRoot::Z;
using CpmRoot::N;
using CpmRoot::R;
using CpmRoot::L;
using CpmRoot::toDouble;
using CpmRoot::skipLeadingWhitespace;
using CpmRoot::skipTrailingWhitespace;

bool Word::scnLin=true;
   // true is dangerous for the presenly implemented movie file
   // production. Don't change without careful analysis of the
   // consequences

Z Word::endOfWord=2; // was 1
   // 0 nothing, 1 blank, else endl
   // choice 2 is the most stable since a endl, in addition to '\n'
   // also puts flush

bool Word::prnOn(ostream& str)const
{
   if (endOfWord==0) str<<rep_;
   else if (endOfWord==1) str<<rep_<<" ";
   else              str<<rep_<<endl; // now the default case
```

```
   //return (str!=0);
   return !str ? false : true;
}

bool Word::scanFrom(istream& str)
{
   if (scnLin) CpmRoot::readLine(rep_,str);
   else        CpmRoot::read(rep_,str);
   //return (str!=0);
   return !str ? false : true;
}

Z Word::find(char c)const
{
   Z res=0;
   Z n=dim();
   for (Z i=0;i<n;++i){
      if (rep_[i]==c){
         res=i+1;
         break;
      }
   }
   return res;
}

Z Word::find(const Word& w)const
{
   string::size_type res=rep_.find(w.rep_);
   if (res==string::npos) return 0;
   else return 1+(Z)res;
}

Word Word::resize(Z newDim)const
{
   Z n1=dim();
   Z n2= newDim<0 ? 0 : newDim;
   char blank=' ';
   Word res(n2,blank);
   Z nMin = n1<=n2 ? n1 : n2;
   for (Z i=1;i<=nMin;++i) res[i]=(*this)[i];
   return res;
}

Word Word::slc_(char c, bool firstSep)
{
   Z d=dim();
   if (d==0) return *this;
   Word temp= firstSep ? *this : rev();
   Z ic=temp.find(c);
   if (ic==0){ Word res=*this; *this=""; return res;}
```

```
   if (d==1 && ic==1){ // Then *this consists of a single c
      *this="";
      return "";
   }
   Word wl(ic-1);
   Word wr(d-ic);
   Z i,j;
   for (i=1;i<ic;++i) wl[i]=temp[i];
   for (i=ic+1,j=1;i<=d;++i,++j) wr[j]=temp[i];
   if (firstSep){
      *this=wr;
      return wl;
   }
   else{
      *this=wl.rev();
      return wr.rev();
   }
}

Word Word::appBefExt(Word const& app)const
{
   Word dot(".");
   Z posDot=find(dot);
   if (posDot==0) return *this&app;
   else{
      Z nameL=posDot-1;
      Z extL=dim()-posDot+1;
      Word name=resize(nameL);
      Word ext=tail(extL);
      name&=app;
      name&=ext;
      return name;
   }
}

Word Word::baseName()const
{
   Word res;
   string::size_type i1=rep_.find_last_of("/");
   string::size_type i2=rep_.find_last_of("\\");
   string repx=rep_;
   if (i1==string::npos && i2==string::npos){
      ; // nothing to do
   }
   else if (i1==string::npos){ // then i2 is a valid string position that
      // holds a backslash
      repx.erase(0,i2+1);
   }
   else if (i2==string::npos){// then i2 is a valid string position that
      // holds a slash
```

```
      repx.erase(0,i1+1);
   }
   else{
      string::size_type imax=i1;
      if (i2>i1) imax=i2;
         // then right of imax there is np slash or backslash
      repx.erase(0,imax+1);
   }
   return Word(repx);
}

char Word::operator[](Z i)const
{
   return rep_.at(i-1); // checked access accorcing to the error handling
      // facilities of the standard library
}

char& Word::operator[](Z i)
{
   return rep_.at(i-1);// checked access accorcing to the error handling
      // facilities of the standard library
}

Word Word::cut(Z i)const
{
   string res=rep_;
   if (i>=0){
      res.erase(res.length()-i,i);
   }
   else{
      res.erase(0,-i);
   }
   return Word(res);
}

Word Word::tail(Z n)const
{
   Z d=dim();
   if (d<=n) return *this;
   Z diff=d-n; // notice diff>=1
   Word res(n);
   for (Z i=0;i<n;i++) res.rep_[i]=rep_[i+diff];
      // notice that for i==n-1, we have i+diff=n-1+d-n=d-1
      // so that we just stop with the last letter of rep_.
      // No essential efficiency compromises. Should be fast.
      // We do not use the substring functionality of the standard
      // library since my compiler does not understand Stroustrup's name
      // Basic_substring<char> even after using namespace std;
      // Also forming a new template class for this minor purpose
      // seems not be justified; also the floppy description of the
```

```
      // substring constructors on p.596 of BSC3 would require some
      // testing for being sure what the function do in case of
      // non-matching arguments.
   return res;
}

Word CpmRoot::operator &(const Word& w1, const Word& w2)
{
   return Word(w1.rep_+w2.rep_);
}

Word CpmRoot::operator +(const Word& w1, const Word& w2)
{
     return Word(w1.rep_+w2.rep_);
}

Word Word::write(bool x)
{
   return CpmRoot::toWord(x);
}

Word Word::write(string x)
{
   return Word(x);
}

#define CPM_SC\
   ostringstream ost;\
   if (length>0) ost<<std::setfill('0')<<std::setw(length)<<n;\
   else ost<<n;\
   return Word(ost.str())

Word Word::write(R n, Z length){CPM_SC;}
Word Word::write(Z n, Z length){CPM_SC;}
Word Word::write(N n, Z length){CPM_SC;}

#undef CPM_SC

R Word::toR()const
{
   stringstream sstr;
   sstr<<rep_;
   R res;
   sstr>>res;
   return res;
}

Z Word::toZ()const
{
   stringstream sstr;
```

```
   sstr<<rep_;
   Z res;
   sstr>>res;
   return res;
}

bool Word::toBool()const
{
   if (rep_=="true" || rep_=="TRUE" ||
      rep_=="wahr" || rep_=="1" || rep_=="W")
      return true;
   else
      return false;
}

namespace{
   const int fieldLength=32;
   // privat name to file
}

#if defined(_WINDOWS)
   #define CPM_SPF sprintf_s
#else
   #define CPM_SPF sprintf
#endif

#define CPM_SC\
   char p[fieldLength];\
   CPM_SPF(p,ff,x);\
   return Word(p)

Word CpmRoot::toWord(N x, const char* ff){CPM_SC;}
Word CpmRoot::toWord(Z x, const char* ff){CPM_SC;}
Word CpmRoot::toWord(R y, const char* ff){ double x=cpmtod(y); CPM_SC;}

#undef CPM_SPF
#undef CPM_SC

Word CpmRoot::operator &(const char* cp, const Word& w)
{ return Word(cp)&w;}
Word CpmRoot::operator &(const Word& w, const char* cp)
{ return w&Word(cp);}
Word CpmRoot::operator &(char cp[], const Word& w)
{ return Word(cp)&w;}
Word CpmRoot::operator &(const Word& w, char cp[])
{ return w&Word(cp);}
Word& Word::operator &=(const Word& w)
{ return *this=(*this)&w;}
Word& Word::operator &=(const char* cp)
{ return *this=(*this)&Word(cp);}
```

```
// for convenience same is allowed to be denoted +

Word CpmRoot::operator +(const char* cp, const Word& w)
{ return Word(cp)&w;}
Word CpmRoot::operator +(const Word& w, const char* cp)
{ return w&Word(cp);}
Word CpmRoot::operator +(char cp[], const Word& w)
{ return Word(cp)&w;}
Word CpmRoot::operator +(const Word& w, char cp[])
{ return w&Word(cp);}
Word& Word::operator +=(const Word& w)
{ return *this=(*this)&w;}
Word& Word::operator +=(const char* cp)
{ return *this=(*this)&Word(cp);}

ofstream& CpmRoot::to_ofstream(const Word& w)
{
    ofstream*  outptr=new ofstream(w.rep_.c_str());
    return *outptr;
}

ifstream& CpmRoot::to_ifstream(const Word& w)
{
    ifstream*  inptr=new ifstream(w.rep_.c_str());
    return *inptr;
}

bool Word::readable()const
{
   ifstream ifs(rep_.c_str());
   if (ifs) return true;
   else return false;
}

Word Word::firstWord()const
{
   istringstream ist(rep_);
   string st;
   ist>>st;
   return Word(st);
}

Word Word::firstWord_()
{
   istringstream ist(rep_);
   string st;
   ist>>st;
   rep_=string();
   char c;
```

```
   while (ist>>c) rep_+=c;
   return Word(st);
}

Z Word::com(const Word& w)const
{
   if (rep_<w.rep_) return 1;
   else if (rep_>w.rep_) return -1;
   else return 0;
}

Word Word::makeFileName()const
{
   string::const_iterator i;
   ostringstream res;
   for (i=rep_.begin(); i!=rep_.end(); ++i){
      if (isalnum(*i)) res<<*i;  else res<<'_';
   }
   return Word(res);
}

void Word::skipSpc_()
{
   string::const_iterator i;
   ostringstream res;
   for (i=rep_.begin(); i!=rep_.end(); ++i){
      if (!isspace(*i)) res<<*i;
   }
   rep_=res.str();
}

Word Word::skipChr(char c)const
{
   string::const_iterator i;
   ostringstream res;
   for (i=rep_.begin(); i!=rep_.end(); ++i){
      if (*i!=c) res<<*i;
   }
   return Word(res);
}

namespace{
   bool isOK(char c, bool extended)
   {
      if (!extended) return c!='.';
      else return c!='_'&& (islower(c)||isdigit(c));
   }
}

// we should remove any contiguous array of upper case characters
```

```
// (here '_' is considered an upper case character).

Word Word::remExt(bool extended)const
{
   string::const_iterator i;
   ostringstream res;
   for (i=rep_.begin(); i!=rep_.end(); ++i){
      if (isOK(*i,extended)) res<<*i;
      else break;
   }
   return Word(res);
}

Word Word::remApp()const
{
   Z d=dim();
   char l=(*this)[d];
   while(isupper(l)||l=='_'){
      d--;
      l=(*this)[d];
   }
   return resize(d);
}

void Word::skipSurSpc_()
{
   string line=skipLeadingWhitespace(rep_);
   rep_=skipTrailingWhitespace(line);
}

R Word::abs(void)const{ return CpmRoot::absT<string>(rep_);}
R Word::absSqr(void)const{ return CpmRoot::absSqrT<string>(rep_);}
Word Word::con(void)const{ return Word(CpmRoot::conT<string>(rep_));}
Word Word::inv(void)const{ return Word(CpmRoot::invT<string>(rep_));}
Word Word::rev(void)const{ return Word(CpmRoot::invT<string>(rep_));}

Word Word::test(Z complexity)const
{
   string s;
   string res=CpmRoot::testT<string>(s,complexity);
   return Word(res);
}

Word Word::ran(Z j)const
{
   string res=CpmRoot::ranT<string>(rep_,j);
         // rep_ was res till 2016-03-10
   return Word(res);
}
```

```
Z Word::hash()const
{
   return CpmRoot::hashT<string>(rep_);
}

R Word::dis(Word const& x)const
{
   return CpmRoot::disT<string>(rep_,x.rep_);
}

Word Word::net(Z i)const
{
   return Word(CpmRoot::netT<string>(rep_,i));
}

// functions CpmRoot::toStr

#define CPM_SC\
   ostringstream ost;\
   if (prc>0) ost.precision(prc);\
   ost<<x;\
   return ost.str()

string CpmRoot::toStr(R const& x, Z prc){CPM_SC;}
#if defined(CPM_ULM)
   string CpmRoot::toStr(rwrap const& x, Z prc){CPM_SC;}
#endif
#undef CPM_SC

#define CPM_SC ostringstream ost; ost<<x; return ost.str()
string CpmRoot::toStr(Z x){ CPM_SC; }
string CpmRoot::toStr(N x){ CPM_SC; }
string CpmRoot::toStr(L x){ CPM_SC; }
string CpmRoot::toStr(bool x){ CPM_SC; }
#undef CPM_SC

const char asciiList[]={
   '0','1','2','3','4','5','6','7','8','9',
   'a','b','c','d','e','f','g','h','i','j','k',
   'l','m','n','o','p','q','r','s','t','u','v',
   'w','x','y','z',
   'A','B','C','D','E','F','G','H','I','J','K',
   'L','M','N','O','P','Q','R','S','T','U','V',
   'W','X','Y','Z',
   '\x20','\x21','\x22','\x23','\x24','\x25','\x26','\x27',
   '\x28','\x29','\x2a','\x2b','\x2c','\x2d','\x2e','\x2f',
   '\x3a','\x3b','\x3c','\x3d','\x3e','\x3f','\x40',
   '\x5b','\x5c','\x5d','\x5e','\x5f','\x60',
   '\x7b','\x7c','\x7d','\x7e','\x7f'
};
```

```
Word Word::ascii()
{
   ostringstream ost;
   ost<<'0'<<'1'<<'2'<<'3'<<'4'<<'5'<<'6'<<'7'<<'8'<<'9'<<
   'a'<<'b'<<'c'<<'d'<<'e'<<'f'<<'g'<<'h'<<'i'<<'j'<<'k'<<
   'l'<<'m'<<'n'<<'o'<<'p'<<'q'<<'r'<<'s'<<'t'<<'u'<<'v'<<
   'w'<<'x'<<'y'<<'z'<<
   'A'<<'B'<<'C'<<'D'<<'E'<<'F'<<'G'<<'H'<<'I'<<'J'<<'K'<<
   'L'<<'M'<<'N'<<'O'<<'P'<<'Q'<<'R'<<'S'<<'T'<<'U'<<'V'<<
   'W'<<'X'<<'Y'<<'Z'<<
   '\x20'<<'\x21'<<'\x23'<<'\x24'<<'\x25'<<'\x26'<<
   '\x28'<<'\x29'<<'\x2a'<<'\x2b'<<'\x2c'<<'\x2d'<<'\x2e'<<'\x2f'<<
   '\x3a'<<'\x3b'<<'\x3c'<<'\x3d'<<'\x3e'<<'\x3f'<<'\x40'<<
   '\x5b'<<'\x5c'<<'\x5d'<<
   '\x7b'<<'\x7c'<<'\x7d'<<'\x7e';
   return Word(ost.str());
}
```

44 cpmx.h

```
//? cpmx.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_X_H_
#define CPM_X_H_
/*

Purpose: Defining lean cartesian products of 2-8 factors
   to be used similar to pair<> of the standard library.
   Needed for enabling functions to return longer value
   lists than just pairs.
   Non-constant reference arguments as a work-around are considered
   bad style here.
   If one needs cartesian products very often as function arguments or as
   function return values, this might be an indication that it would
   natural to define one or more classes which aggregate these components
   into a meaningful object.

   All these cartesian producs define IO and order operations in the
   same manner as V1 and V do this. This is a considerable simplification
   compared to my old approach to have templates Xno, Xnio for those
   template arguments which implement order operations and IO operations.

   Notice that access to components can be done via references and
   values. Notice also that, other than for pair, first,second,...
   are functions and not data. So the following is correct:
   X2<R,Z> a(4,6);
   Z i=a.second(); // not i=a.second;
      // i.e. i=6
   However,
   Z i=a.c2();
   looks nicer to me
```

```
   Addition February 2009: For pairs, triplets, and quartets there are now
   also homotypic versions. Component access is given here through public
   data members x1, x2, x3, x4.
   Also here, IO and order relations are implemented.
*/
#include <cpmword.h>
#include <vector>
#include <tuple>

namespace CpmArrays{

   using CpmRoot::Word;
   using CpmRoot::Z;
   using namespace CpmStd;
   using std::tuple; // 2015-06-11 conversion to std::tupel added.
   using std::get;

// Classes introduced by this file with fixed names:

//       X2, X3, X4, X5, X6, X7, X8
//       T2, T3, T4

// All template arguments are assumed to have (not necessarily to define
// explicitely) default constructor, copy constructor, and operator =

// Lists of 2,3,4,5,6,7 or 8 components (hetero-typic lists)
//   and
// homo-typic lists with 2,3, or 4 components.
// Components of Xi accessible by member functions c1(),c2(),...
// Components of Ti accessible by member functions c1(),c2(),... and by
// conventional indexing (first valid index is 1)
// (indexing added 2011-03-22)
////////////////////////////// class X2<> ////////////////////////////

template <class Y1, class Y2>
class X2{ // heterotypic pairs
   // Lean Cartesian product of Y1 with Y2
protected:
   Y1 x1;
   Y2 x2;
   typedef X2<Y1,Y2> Type;

public:
   CPM_IO
   CPM_ORDER
   Z dim()const{ return 2;}
 // constructors
   X2(Y1 const& x1_, Y2 const& x2_):x1(x1_),x2(x2_){}
   X2(void):x1(),x2(){}
   X2(X2<Y1,Y2> const& z): x1(z.x1), x2(z.x2){}
```

```
   X2(tuple<Y1,Y2> const& t): x1(get<0>(t)),x2(get<1>(t)){}

// 'modern' access functions. Similar to operator[]
// getting the second component of object a is done like
//
// ...=a.c2();
//
// setting is done like
//
// a.c2()=...;
//
   Y1& c1(){ return x1;}
   Y2& c2(){ return x2;}
   Y1 const& c1()const{ return x1;}
   Y2 const& c2()const{ return x2;}
   Y1 first()const{ return x1;}
   Y2 second()const{ return x2;}

   Word nameOf()const
   {
       return Word("X2< "&
          CpmRoot::Name<Y1>()(Y1())&
          ","&
          CpmRoot::Name<Y2>()(Y2())&
          " >");
   }

   tuple<Y1,Y2> toTup()const{ return tuple<Y1,Y2>{x1,x2};}

};

template <class Y1, class Y2>
Z X2<Y1,Y2>::com(X2<Y1,Y2> const& s)const
{
   if (c1()<s.c1()) return 1;
   if (c1()>s.c1()) return -1;
   if (c2()<s.c2()) return 1;
   if (c2()>s.c2()) return -1;
   return 0;
}

template <class Y1, class Y2>
bool X2<Y1,Y2>::prnOn(ostream& str)const
{
   cpmp(x1);
   cpmp(x2);
   return true;
}

template <class Y1, class Y2>
```

```
bool X2<Y1,Y2>::scanFrom(istream& str)
{
   cpms(x1);
   cpms(x2);
   return true;
}

////////////////////////////// class T2<> ////////////////////////////

template <class X>
class T2{ // homotypic pairs
   // Lean Cartesian product of X with itself.
public:
   X x1, x2;
   typedef T2<X> Type;
   CPM_IO
   CPM_ORDER
   Z dim()const{ return 2;}
 // constructors
   T2(X const& x1_, X const& x2_):x1(x1_),x2(x2_){}
   T2(void):x1(),x2(){}
   T2(T2<X> const& z): x1(z.x1), x2(z.x2){}
   X& operator[](Z i){ if (i>=2) return x2; else return x1;}
   X const& operator[](Z i)const{ if (i>=2) return x2; else return x1;}
   X& fir(){ return x1;}
      //: first
   X& last(){ return x2;}
      //: last
   Z b()const { return 1;}
      //. begin
      // Returns the first valid index.
   Z e()const { return 2;}
      //. end
      // Returns the last valid index.
   X& c1(void){ return x1;}
   X& c2(void){ return x2;}
   X const& c1(void)const{ return x1;}
   X const& c2(void)const{ return x2;}
   Word nameOf()const
   {
      return Word("T2< "&CpmRoot::Name<X>()(x1)&" >");
   }
};

template <class Y>
Z T2<Y>::com(T2<Y> const& s)const
{
   if (x1<s.x1) return 1;
   if (x1>s.x1) return -1;
   if (x2<s.x2) return 1;
```

```
   if (x2>s.x2) return -1;
   return 0;
}

template <class Y>
bool T2<Y>::prnOn(ostream& str)const
{
   cpmp(x1);
   cpmp(x2);
   return true;
}

template <class Y>
bool T2<Y>::scanFrom(istream& str)
{
   cpms(x1);
   cpms(x2);
   return true;
}

////////////////////////////// class X3<> //////////////////////////

// for more factors:

template <class Y1, class Y2, class Y3>
class  X3{ // heterotypic triplets
protected:
   Y1 x1;
   Y2 x2;
   Y3 x3;
   typedef X3<Y1,Y2,Y3> Type;
public:
   CPM_IO
   CPM_ORDER
   Z dim()const{ return 3;}
   X3(Y1 const& x1_, Y2 const& x2_, Y3 const& x3_):x1(x1_),x2(x2_),
      x3(x3_){}
   X3(X2<Y1,Y2> const& y_, Y3 const& x3_):x1(y_.c1()),x2(y_.c2()),
      x3(x3_){}
   X3(void):x1(),x2(),x3(){}
   X3(X3<Y1,Y2,Y3> const& z): x1(z.x1), x2(z.x2), x3(z.x3){}
   X3(tuple<Y1,Y2,Y3> const& t): x1(get<0>(t)),x2(get<1>(t)),
      x3(get<2>(t)){}

   Y1& c1(void){ return x1;}
   Y2& c2(void){ return x2;}
   Y3& c3(void){ return x3;}

   Y1 const& c1(void)const{ return x1;}
   Y2 const& c2(void)const{ return x2;}
```

```
   Y3 const& c3(void)const{ return x3;}

   Y1 first()const{ return x1;}
   Y2 second()const{ return x2;}
   Y3 third()const{ return x3;}

   Word nameOf()const
   {
       return Word("X3< "&
          CpmRoot::Name<Y1>()(Y1())&
          ","&
          CpmRoot::Name<Y2>()(Y2())&
          ","&
          CpmRoot::Name<Y3>()(Y3())&
          " >");
   }

   tuple<Y1,Y2,Y3> toTup()const{ return tuple<Y1,Y2,Y3>{x1,x2,x3};}

};

template <class Y1, class Y2, class Y3>
Z X3<Y1,Y2,Y3>::com(X3<Y1,Y2,Y3> const& s)const
{
   if (c1()<s.c1()) return 1;
   if (c1()>s.c1()) return -1;
   if (c2()<s.c2()) return 1;
   if (c2()>s.c2()) return -1;
   if (c3()<s.c3()) return 1;
   if (c3()>s.c3()) return -1;
   return 0;
}

template <class Y1, class Y2, class Y3>
bool X3<Y1,Y2,Y3>::prnOn(ostream& str)const
{
   cpmp(x1);
   cpmp(x2);
   cpmp(x3);
   return true;
}

template <class Y1, class Y2, class Y3>
bool X3<Y1,Y2,Y3>::scanFrom(istream& str)
{
   cpms(x1);
   cpms(x2);
   cpms(x3);
   return true;
}
```

```
////////////////////////////// class T3<> ////////////////////////////

template <class X>
class T3{ // homotypic triplets
   // Lean Cartesian product of X with itself.
public:
   X x1, x2, x3;
   typedef T3<X> Type;
   CPM_IO
   CPM_ORDER
   Z dim()const{ return 3;}
 // constructors
   T3(X const& x1_, X const& x2_, X const& x3_):x1(x1_),x2(x2_),x3(x3_){}
   T3(void):x1(),x2(),x3(){}
   T3(T3<X> const& z): x1(z.x1), x2(z.x2),x3(z.x3){}

   X& operator[](Z i){ if (i>=3) return x3; else if (i==2) return x2;
    else return x1;}
   X const& operator[](Z i)const{ if (i>=3) return x3; else if (i==2)
    return x2; else return x1;}
   X& fir(){ return x1;}
      //: first
   X& last(){ return x3;}
      //: last
   Z b()const { return 1;}
      //. begin
      // Returns the first valid index.
   Z e()const { return 3;}
      //. end
      // Returns the last valid index.

   X& c1(void){ return x1;}
   X& c2(void){ return x2;}
   X& c3(void){ return x3;}

   X const& c1(void)const{ return x1;}
   X const& c2(void)const{ return x2;}
   X const& c3(void)const{ return x3;}

   Word nameOf()const
   {
      return Word("T3< "&CpmRoot::Name<X>()(x1)&" >");
   }
};

template <class Y>
Z T3<Y>::com(T3<Y> const& s)const
{
   if (x1<s.x1) return 1;
```

```
   if (x1>s.x1) return -1;
   if (x2<s.x2) return 1;
   if (x2>s.x2) return -1;
   if (x3<s.x3) return 1;
   if (x3>s.x3) return -1;
   return 0;
}

template <class Y>
bool T3<Y>::prnOn(ostream& str)const
{
   cpmp(x1);
   cpmp(x2);
   cpmp(x3);
   return true;
}

template <class Y>
bool T3<Y>::scanFrom(istream& str)
{
   cpms(x1);
   cpms(x2);
   cpms(x3);
   return true;
}

////////////////////////////// class X4<> ////////////////////////////

template <class Y1, class Y2, class Y3, class Y4>
class  X4{ // heterotypic 4-tuples
protected:
   Y1 x1;
   Y2 x2;
   Y3 x3;
   Y4 x4;
   typedef X4<Y1,Y2,Y3,Y4> Type;

public:
   CPM_IO
   CPM_ORDER
   Z dim()const{ return 4;}
   X4(Y1 const& x1_, Y2 const& x2_, Y3 const& x3_, Y4 const& x4_):
      x1(x1_),x2(x2_),x3(x3_),x4(x4_){}
   X4(X3<Y1,Y2,Y3> const& y_, Y4 const& x4_):
      x1(y_.c1()),x2(y_.c2()),x3(y_.c3()),x4(x4_){}
   X4(void):x1(),x2(),x3(),x4(){}
   X4(X4<Y1,Y2,Y3,Y4> const& z): x1(z.x1), x2(z.x2), x3(z.x3),x4(z.x4){}
   X4(tuple<Y1,Y2,Y3,Y4> const& t): x1(get<0>(t)),x2(get<1>(t)),
      x3(get<2>(t)),x4(get<3>(t)){}
```

```
   Y1& c1(void){ return x1;}
   Y2& c2(void){ return x2;}
   Y3& c3(void){ return x3;}
   Y4& c4(void){ return x4;}

   Y1 const& c1(void)const{ return x1;}
   Y2 const& c2(void)const{ return x2;}
   Y3 const& c3(void)const{ return x3;}
   Y4 const& c4(void)const{ return x4;}

   Y1 first()const{ return x1;}
   Y2 second()const{ return x2;}
   Y3 third()const{ return x3;}
   Y4 fourth()const{ return x4;}

   Word nameOf()const
   {
       return Word("X4< "&
          CpmRoot::Name<Y1>()(Y1())&
          ","&
          CpmRoot::Name<Y2>()(Y2())&
          ","&
          CpmRoot::Name<Y3>()(Y3())&
          ","&
          CpmRoot::Name<Y4>()(Y4())&
          " >");
   }

   tuple<Y1,Y2,Y3,Y4> toTup()const
   {
      return tuple<Y1,Y2,Y3,Y4>{x1,x2,x3,x4};
   }

};

template <class Y1, class Y2, class Y3, class Y4>
Z X4<Y1,Y2,Y3,Y4>::com(X4<Y1,Y2,Y3,Y4> const& s)const
{
   if (c1()<s.c1()) return 1;
   if (c1()>s.c1()) return -1;
   if (c2()<s.c2()) return 1;
   if (c2()>s.c2()) return -1;
   if (c3()<s.c3()) return 1;
   if (c3()>s.c3()) return -1;
   if (c4()<s.c4()) return 1;
   if (c4()>s.c4()) return -1;
   return 0;
}

template <class Y1, class Y2, class Y3, class Y4>
```

```
bool X4<Y1,Y2,Y3,Y4>::prnOn(ostream& str)const
{
   cpmp(x1);
   cpmp(x2);
   cpmp(x3);
   cpmp(x4);
   return true;
}

template <class Y1, class Y2, class Y3, class Y4>
bool X4<Y1,Y2,Y3,Y4>::scanFrom(istream& str)
{
   cpms(x1);
   cpms(x2);
   cpms(x3);
   cpms(x4);
   return true;
}

////////////////////////////// class T4<> ////////////////////////////////

template <class X>
class T4{ // homotypic quartets
   // Lean Cartesian product of X with itself.
public:
   X x1, x2, x3, x4;
   typedef T4<X> Type;
   CPM_IO
   CPM_ORDER
   Z dim()const{ return 4;}
 // constructors
   T4(X const& x1_, X const& x2_, X const& x3_, X const& x4_):x1(x1_),
      x2(x2_),x3(x3_),x4(x4_){}
   T4(void):x1(),x2(),x3(),x4(){}
   T4(T4<X> const& z): x1(z.x1), x2(z.x2),x3(z.x3),x4(z.x4){}

   X& operator[](Z i)
   { if (i>=4) return x4; else if (i==3) return x3;
      else if (i==2) return x2; else return x1;}

   X const& operator[](Z i)const
   { if (i>=4) return x4; else if (i==3) return x3;
      else if (i==2) return x2; else return x1;}

   X& fir(){ return x1;}
      //: first
   X& last(){ return x4;}
      //: last
   Z b()const { return 1;}
      //. begin
```

```
      // Returns the first valid index.
   Z e()const { return 4;}
      //. end
      // Returns the last valid index.

   X& c1(void){ return x1;}
   X& c2(void){ return x2;}
   X& c3(void){ return x3;}
   X& c4(void){ return x4;}

   X const& c1(void)const{ return x1;}
   X const& c2(void)const{ return x2;}
   X const& c3(void)const{ return x3;}
   X const& c4(void)const{ return x4;}

   Word nameOf()const
   {
      return Word("T4< "&CpmRoot::Name<X>()(x1)&" >");
   }
};

template <class Y>
Z T4<Y>::com(T4<Y> const& s)const
{
   if (x1<s.x1) return 1;
   if (x1>s.x1) return -1;
   if (x2<s.x2) return 1;
   if (x2>s.x2) return -1;
   if (x3<s.x3) return 1;
   if (x3>s.x3) return -1;
   if (x4<s.x4) return 1;
   if (x4>s.x4) return -1;
   return 0;
}

template <class Y>
bool T4<Y>::prnOn(ostream& str)const
{
   cpmp(x1);
   cpmp(x2);
   cpmp(x3);
   cpmp(x4);
   return true;
}

template <class Y>
bool T4<Y>::scanFrom(istream& str)
{
   cpms(x1);
   cpms(x2);
```

```
   cpms(x3);
   cpms(x4);
   return true;
}

////////////////////////////// class X5<> //////////////////////////

template <class Y1, class Y2, class Y3, class Y4, class Y5>
class  X5{ // heterotypic 5-tuples
protected:
   Y1 x1;
   Y2 x2;
   Y3 x3;
   Y4 x4;
   Y5 x5;
   typedef X5<Y1,Y2,Y3,Y4,Y5> Type;
public:
   CPM_IO
   CPM_ORDER
   Z dim()const{ return 5;}
   X5(Y1 const& x1_, Y2 const& x2_, Y3 const& x3_,
      Y4 const& x4_, Y5 const& x5_):x1(x1_),x2(x2_),x3(x3_),
      x4(x4_),x5(x5_){}
   X5(X4<Y1,Y2,Y3,Y4> const& y_, Y5 const& x5_):
      x1(y_.c1()),x2(y_.c2()),x3(y_.c3()),
      x4(y_.c4()),x5(x5_){}
   X5(void):x1(),x2(),x3(),x4(),x5(){}
   X5(X5<Y1,Y2,Y3,Y4,Y5> const& z): x1(z.x1), x2(z.x2),
      x3(z.x3),x4(z.x4),x5(z.x5){}

   X5(tuple<Y1,Y2,Y3,Y4,Y5> const& t): x1(get<0>(t)),x2(get<1>(t)),
      x3(get<2>(t)),x4(get<3>(t)),x5(get<4>(t)){}

   Y1& c1(void){ return x1;}
   Y2& c2(void){ return x2;}
   Y3& c3(void){ return x3;}
   Y4& c4(void){ return x4;}
   Y5& c5(void){ return x5;}

   Y1 const& c1(void)const{ return x1;}
   Y2 const& c2(void)const{ return x2;}
   Y3 const& c3(void)const{ return x3;}
   Y4 const& c4(void)const{ return x4;}
   Y5 const& c5(void)const{ return x5;}

   Y1 first()const{ return x1;}
   Y2 second()const{ return x2;}
   Y3 third()const{ return x3;}
   Y4 fourth()const{ return x4;}
   Y5 fifth()const{ return x5;}
```

```
   Word nameOf()const
   {
       return Word("X5< "&
          CpmRoot::Name<Y1>()(Y1())&
          ","&
          CpmRoot::Name<Y2>()(Y2())&
          ","&
          CpmRoot::Name<Y3>()(Y3())&
          ","&
          CpmRoot::Name<Y4>()(Y4())&
          ","&
          CpmRoot::Name<Y5>()(Y5())&
          " >");
   }

   tuple<Y1,Y2,Y3,Y4,Y5> toTup()const
   {
      return tuple<Y1,Y2,Y3,Y4,Y5>{x1,x2,x3,x4,x5};
   }
};

template <class Y1, class Y2, class Y3, class Y4, class Y5>
Z X5<Y1,Y2,Y3,Y4,Y5>::com(X5<Y1,Y2,Y3,Y4,Y5> const& s)const
{
   if (c1()<s.c1()) return 1;
   if (c1()>s.c1()) return -1;
   if (c2()<s.c2()) return 1;
   if (c2()>s.c2()) return -1;
   if (c3()<s.c3()) return 1;
   if (c3()>s.c3()) return -1;
   if (c4()<s.c4()) return 1;
   if (c4()>s.c4()) return -1;
   if (c5()<s.c5()) return 1;
   if (c5()>s.c5()) return -1;
   return 0;
}

template <class Y1, class Y2, class Y3, class Y4, class Y5>
bool X5<Y1,Y2,Y3,Y4,Y5>::prnOn(ostream& str)const
{
   cpmp(x1);
   cpmp(x2);
   cpmp(x3);
   cpmp(x4);
   cpmp(x5);
   return true;
}

template <class Y1, class Y2, class Y3, class Y4, class Y5>
```

```
bool X5<Y1,Y2,Y3,Y4,Y5>::scanFrom(istream& str)
{
   cpms(x1);
   cpms(x2);
   cpms(x3);
   cpms(x4);
   cpms(x5);
   return true;
}

////////////////////////////// class X6<> ////////////////////////////

template <class Y1, class Y2, class Y3, class Y4, class Y5, class Y6>
class  X6{ // heterotypic 6-tuples
protected:
   Y1 x1;
   Y2 x2;
   Y3 x3;
   Y4 x4;
   Y5 x5;
   Y6 x6;
   typedef X6<Y1,Y2,Y3,Y4,Y5,Y6> Type;
public:
   CPM_IO
   CPM_ORDER
   Z dim()const{ return 6;}

   X6(Y1 const& x1_, Y2 const& x2_, Y3 const& x3_,
      Y4 const& x4_, Y5 const& x5_, Y6 const& x6_):x1(x1_),x2(x2_),
      x3(x3_), x4(x4_),x5(x5_),x6(x6_){}

   X6(X5<Y1,Y2,Y3,Y4,Y5> const& y_, Y6 const& x6_):
      x1(y_.c1()),x2(y_.c2()),x3(y_.c3()),
      x4(y_.c4()),x5(y_.c5()),x6(x6_){}

   X6(void):x1(),x2(),x3(),x4(),x5(),x6(){}

   X6(X6<Y1,Y2,Y3,Y4,Y5,Y6> const& z): x1(z.x1), x2(z.x2),
      x3(z.x3),x4(z.x4),x5(z.x5),x6(z.x6){}

   X6(tuple<Y1,Y2,Y3,Y4,Y5,Y6> const& t): x1(get<0>(t)),x2(get<1>(t)),
      x3(get<2>(t)),x4(get<3>(t)),x5(get<4>(t)),x6(get<5>(t)){}

   Y1& c1(void){ return x1;}
   Y2& c2(void){ return x2;}
   Y3& c3(void){ return x3;}
   Y4& c4(void){ return x4;}
   Y5& c5(void){ return x5;}
   Y6& c6(void){ return x6;}
```

```
   Y1 const& c1(void)const{ return x1;}
   Y2 const& c2(void)const{ return x2;}
   Y3 const& c3(void)const{ return x3;}
   Y4 const& c4(void)const{ return x4;}
   Y5 const& c5(void)const{ return x5;}
   Y6 const& c6(void)const{ return x6;}

   Y1 first()const{ return x1;}
   Y2 second()const{ return x2;}
   Y3 third()const{ return x3;}
   Y4 fourth()const{ return x4;}
   Y5 fifth()const{ return x5;}
   Y6 sixth()const{ return x6;}

   Word nameOf()const
   {
       return Word("X6< "&
          CpmRoot::Name<Y1>()(Y1())&
          ","&
          CpmRoot::Name<Y2>()(Y2())&
          ","&
          CpmRoot::Name<Y3>()(Y3())&
          ","&
          CpmRoot::Name<Y4>()(Y4())&
          ","&
          CpmRoot::Name<Y5>()(Y5())&
          ","&
          CpmRoot::Name<Y6>()(Y6())&
          " >");
   }

   tuple<Y1,Y2,Y3,Y4,Y5,Y6> toTup()const
   {
      return tuple<Y1,Y2,Y3,Y4,Y5,Y6>{x1,x2,x3,x4,x5,x6};
   }
};

template <class Y1, class Y2, class Y3, class Y4, class Y5, class Y6>
Z X6<Y1,Y2,Y3,Y4,Y5,Y6>::com(X6<Y1,Y2,Y3,Y4,Y5,Y6> const& s)const
{
   if (c1()<s.c1()) return 1;
   if (c1()>s.c1()) return -1;
   if (c2()<s.c2()) return 1;
   if (c2()>s.c2()) return -1;
   if (c3()<s.c3()) return 1;
   if (c3()>s.c3()) return -1;
   if (c4()<s.c4()) return 1;
   if (c4()>s.c4()) return -1;
   if (c5()<s.c5()) return 1;
   if (c5()>s.c5()) return -1;
```

```
   if (c6()<s.c6()) return 1;
   if (c6()>s.c6()) return -1;
   return 0;
}

template <class Y1, class Y2, class Y3, class Y4, class Y5, class Y6>
bool X6<Y1,Y2,Y3,Y4,Y5,Y6>::prnOn(ostream& str)const
{
   cpmp(x1);
   cpmp(x2);
   cpmp(x3);
   cpmp(x4);
   cpmp(x5);
   cpmp(x6);
   return true;
}

template <class Y1, class Y2, class Y3, class Y4, class Y5, class Y6>
bool X6<Y1,Y2,Y3,Y4,Y5,Y6>::scanFrom(istream& str)
{
   cpms(x1);
   cpms(x2);
   cpms(x3);
   cpms(x4);
   cpms(x5);
   cpms(x6);
   return true;
}

////////////////////////////// class X7<> ////////////////////////////

template <class Y1, class Y2, class Y3, class Y4, class Y5,
   class Y6, class Y7>
class  X7{ // heterotypic 7-tuples
protected:
   Y1 x1;
   Y2 x2;
   Y3 x3;
   Y4 x4;
   Y5 x5;
   Y6 x6;
   Y7 x7;
   typedef X7<Y1,Y2,Y3,Y4,Y5,Y6,Y7> Type;
public:
   CPM_IO
   CPM_ORDER
   Z dim()const{ return 7;}
   X7(Y1 const& x1_, Y2 const& x2_, Y3 const& x3_,
      Y4 const& x4_, Y5 const& x5_, Y6 const& x6_, Y7 const& x7_):
      x1(x1_),x2(x2_),x3(x3_),x4(x4_),x5(x5_),x6(x6_),x7(x7_){}
```

```
X7(X6<Y1,Y2,Y3,Y4,Y5,Y6> const& y_, Y7 const& x7_):
   x1(y_.c1()),x2(y_.c2()),x3(y_.c3()),
   x4(y_.c4()),x5(y_.c5()),x6(y_.c6()),x7(x7_){}

X7(void):x1(),x2(),x3(),x4(),x5(),x6(),x7(){}
X7(X7<Y1,Y2,Y3,Y4,Y5,Y6,Y7> const& z): x1(z.x1), x2(z.x2),
   x3(z.x3),x4(z.x4),x5(z.x5),x6(z.x6),x7(z.x7){}

X7(tuple<Y1,Y2,Y3,Y4,Y5,Y6,Y7> const& t): x1(get<0>(t)),x2(get<1>(t)),
   x3(get<2>(t)),x4(get<3>(t)),x5(get<4>(t)),x6(get<5>(t)),
   x7(get<6>(t)){}


Y1& c1(void){ return x1;}
Y2& c2(void){ return x2;}
Y3& c3(void){ return x3;}
Y4& c4(void){ return x4;}
Y5& c5(void){ return x5;}
Y6& c6(void){ return x6;}
Y7& c7(void){ return x7;}

Y1 const& c1(void)const{ return x1;}
Y2 const& c2(void)const{ return x2;}
Y3 const& c3(void)const{ return x3;}
Y4 const& c4(void)const{ return x4;}
Y5 const& c5(void)const{ return x5;}
Y6 const& c6(void)const{ return x6;}
Y7 const& c7(void)const{ return x7;}

Y1 first()const{ return x1;}
Y2 second()const{ return x2;}
Y3 third()const{ return x3;}
Y4 fourth()const{ return x4;}
Y5 fifth()const{ return x5;}
Y6 sixth()const{ return x6;}
Y7 seventh()const{ return x7;}

Word nameOf()const
{
    return Word("X7< "&
       CpmRoot::Name<Y1>()(Y1())&
       ","&
       CpmRoot::Name<Y2>()(Y2())&
       ","&
       CpmRoot::Name<Y3>()(Y3())&
       ","&
       CpmRoot::Name<Y4>()(Y4())&
       ","&
       CpmRoot::Name<Y5>()(Y5())&
```

```
            ","&
            CpmRoot::Name<Y6>()(Y6())&
            ","&
            CpmRoot::Name<Y7>()(Y7())&
            " >");
   }

   tuple<Y1,Y2,Y3,Y4,Y5,Y6,Y7> toTup()const
   {
      return tuple<Y1,Y2,Y3,Y4,Y5,Y6,Y7>{x1,x2,x3,x4,x5,x6,x7};
   }

};

template <class Y1, class Y2, class Y3, class Y4, class Y5,
   class Y6, class Y7>
Z X7<Y1,Y2,Y3,Y4,Y5,Y6,Y7>::com(X7<Y1,Y2,Y3,Y4,Y5,Y6,Y7> const& s)const
{
   if (c1()<s.c1()) return 1;
   if (c1()>s.c1()) return -1;
   if (c2()<s.c2()) return 1;
   if (c2()>s.c2()) return -1;
   if (c3()<s.c3()) return 1;
   if (c3()>s.c3()) return -1;
   if (c4()<s.c4()) return 1;
   if (c4()>s.c4()) return -1;
   if (c5()<s.c5()) return 1;
   if (c5()>s.c5()) return -1;
   if (c6()<s.c6()) return 1;
   if (c6()>s.c6()) return -1;
   if (c7()<s.c7()) return 1;
   if (c7()>s.c7()) return -1;
   return 0;
}

template <class Y1, class Y2, class Y3, class Y4, class Y5,
   class Y6, class Y7>
bool X7<Y1,Y2,Y3,Y4,Y5,Y6,Y7>::prnOn(ostream& str)const
{
   cpmp(x1);
   cpmp(x2);
   cpmp(x3);
   cpmp(x4);
   cpmp(x5);
   cpmp(x6);
   cpmp(x7);
   return true;
}

template <class Y1, class Y2, class Y3, class Y4, class Y5,
```

```
   class Y6, class Y7>
bool X7<Y1,Y2,Y3,Y4,Y5,Y6,Y7>::scanFrom(istream& str)
{
   cpms(x1);
   cpms(x2);
   cpms(x3);
   cpms(x4);
   cpms(x5);
   cpms(x6);
   cpms(x7);
   return true;
}

////////////////////////////// class X8<> ////////////////////////////

template <class Y1, class Y2, class Y3, class Y4, class Y5,
   class Y6, class Y7, class Y8>
class  X8{ // heterotypic 8-tuples
protected:
   Y1 x1;
   Y2 x2;
   Y3 x3;
   Y4 x4;
   Y5 x5;
   Y6 x6;
   Y7 x7;
   Y8 x8;
   typedef X8<Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8> Type;
public:
   CPM_IO
   CPM_ORDER
   Z dim()const{ return 8;}
   X8(Y1 const& x1_, Y2 const& x2_, Y3 const& x3_,
      Y4 const& x4_, Y5 const& x5_, Y6 const& x6_, Y7 const& x7_,
      Y8 const& x8_):
      x1(x1_),x2(x2_),x3(x3_),x4(x4_),x5(x5_),x6(x6_),x7(x7_),x8(x8_){}

   X8(X7<Y1,Y2,Y3,Y4,Y5,Y6,Y7> const& y_, Y8 const& x8_):
      x1(y_.c1()),x2(y_.c2()),x3(y_.c3()),
      x4(y_.c4()),x5(y_.c5()),x6(y_.c6()),x7(y_.c7()),x8(x8_){}

   X8(void):x1(),x2(),x3(),x4(),x5(),x6(),x7(),x8(){}
   X8(X8<Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8> const& z): x1(z.x1), x2(z.x2),
      x3(z.x3),x4(z.x4),x5(z.x5),x6(z.x6),x7(z.x7),x8(z.x8){}

   X8(tuple<Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8> const& t): x1(get<0>(t)),
      x2(get<1>(t)),x3(get<2>(t)),x4(get<3>(t)),x5(get<4>(t)),
      x6(get<5>(t)),x7(get<6>(t)),x8(get<7>(t)){}

   Y1& c1(void){ return x1;}
```

```
Y2& c2(void){ return x2;}
Y3& c3(void){ return x3;}
Y4& c4(void){ return x4;}
Y5& c5(void){ return x5;}
Y6& c6(void){ return x6;}
Y7& c7(void){ return x7;}
Y8& c8(void){ return x8;}

Y1 const& c1(void)const{ return x1;}
Y2 const& c2(void)const{ return x2;}
Y3 const& c3(void)const{ return x3;}
Y4 const& c4(void)const{ return x4;}
Y5 const& c5(void)const{ return x5;}
Y6 const& c6(void)const{ return x6;}
Y7 const& c7(void)const{ return x7;}
Y8 const& c8(void)const{ return x8;}

Y1 first()const{ return x1;}
Y2 second()const{ return x2;}
Y3 third()const{ return x3;}
Y4 fourth()const{ return x4;}
Y5 fifth()const{ return x5;}
Y6 sixth()const{ return x6;}
Y7 seventh()const{ return x7;}
Y8 eighth()const{ return x8;}

Word nameOf()const
{
    return Word("X8< "&
       CpmRoot::Name<Y1>()(Y1())&
       ","&
       CpmRoot::Name<Y2>()(Y2())&
       ","&
       CpmRoot::Name<Y3>()(Y3())&
       ","&
       CpmRoot::Name<Y4>()(Y4())&
       ","&
       CpmRoot::Name<Y5>()(Y5())&
       ","&
       CpmRoot::Name<Y6>()(Y6())&
       ","&
       CpmRoot::Name<Y7>()(Y7())&
       ","&
       CpmRoot::Name<Y8>()(Y8())&
       " >");
}

tuple<Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8> toTup()const
{
   return tuple<Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8>{x1,x2,x3,x4,x5,x6,x7,x8};
```

```
   }

};

template <class Y1, class Y2, class Y3, class Y4, class Y5,
   class Y6, class Y7, class Y8>
Z X8<Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8>::com(
   X8<Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8> const& s)const
{
   if (c1()<s.c1()) return 1;
   if (c1()>s.c1()) return -1;
   if (c2()<s.c2()) return 1;
   if (c2()>s.c2()) return -1;
   if (c3()<s.c3()) return 1;
   if (c3()>s.c3()) return -1;
   if (c4()<s.c4()) return 1;
   if (c4()>s.c4()) return -1;
   if (c5()<s.c5()) return 1;
   if (c5()>s.c5()) return -1;
   if (c6()<s.c6()) return 1;
   if (c6()>s.c6()) return -1;
   if (c7()<s.c7()) return 1;
   if (c7()>s.c7()) return -1;
   if (c8()<s.c8()) return 1;
   if (c8()>s.c8()) return -1;
   return 0;
}

template <class Y1, class Y2, class Y3, class Y4, class Y5,
   class Y6, class Y7, class Y8>
bool X8<Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8>::prnOn(ostream& str)const
{
   cpmp(x1);
   cpmp(x2);
   cpmp(x3);
   cpmp(x4);
   cpmp(x5);
   cpmp(x6);
   cpmp(x7);
   cpmp(x8);
   return true;
}

template <class Y1, class Y2, class Y3, class Y4, class Y5,
   class Y6, class Y7, class Y8>
bool X8<Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8>::scanFrom(istream& str)
{
   cpms(x1);
   cpms(x2);
   cpms(x3);
```

```
   cpms(x4);
   cpms(x5);
   cpms(x6);
   cpms(x7);
   cpms(x8);
   return true;
}

// In a context where X1,X2,X3,X4,X5,X6,X7,X8 are used as
// names for template arguments one could get in trouble
// by a statement
//
// using CpmArrays::X2;
//
// thus one should avoid this. Then the full name CpmArrays::X2 would
// be necessary  which is too long a name for this innocent construct.
// By the following definition we may use cpmX2, which follows the same
// notational scheme as cpmerror, cpmwrite, ...

#define cpmX2 CpmArrays::X2
#define cpmX3 CpmArrays::X3
#define cpmX4 CpmArrays::X4
#define cpmX5 CpmArrays::X5
#define cpmX6 CpmArrays::X6
#define cpmX7 CpmArrays::X7
#define cpmX8 CpmArrays::X8

#define cpmT2 CpmArrays::T2
#define cpmT3 CpmArrays::T3
#define cpmT4 CpmArrays::T4

} // CpmArrays

#endif
```

45 cpmzinterval.h

```
//? cpmzinterval.h
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#ifndef CPM_ZINTERVAL_H_
#define CPM_ZINTERVAL_H_
/*
    Description:
      class for 'intervals' in Z

    Recent history:
    2012-01-03 functions setUnn, setSec, setDif, leftOf, rigOf added,
      #include <cpmx.h> added

*/

#include <cpmsystem.h>
#include <cpmx.h>
   // needed for functions that return a pair of IvZ objects

namespace CpmArrays{

   using CpmRoot::Z;
   using CpmRoot::R;
   using CpmRoot::rnd;
   using CpmRoot::Word;

////////////////////// class IvZ ////////////////////////////////////
// class IvZ will be used in the definition of the basic array template
// V<> which, in turn, will be used to define the set template S<>.
// So it is not possible at this point to relate IvZ with S<Z>
// although clearly each instance of IvZ determines uniquely
// an instance of S<Z>. The C++ classes IvZ and S<Z> are very
// different in implementation and efficiency. For instance
```

```
//    IvZ iv(1,10000);
// defines a 'set' of 10000 integers by memorizing just two
// integers (1 and 10000 in this case). Whereas
//    S<Z> s(10000);
// (which calls a private constructor and thus may appear only in
// member function definition code) allocates memory for
// 10000 integers. When V<> will have been defined, we can give
// as V<IvZ> a storage-economic representation of arbitrary
// finite subsets of Z.
// That IvZ 'represents' subsets of Z expresses itself in
// the member function
//    bool hasElm(Z const& i)const
// which S<Z> defines with just the same signature.

class IvZ{
   // 'intervals of integers', i.e. contiguous finite subsets of Z
   // The void set is included; this is important since T-valued arrays
   // indexed by i's ranging over some instance of IvZ will be used
   // for defining our basic array template class V<T>. V<T> has to have
   // void arrays, which correspond to a void index set.
   // No virtual functions for efficency.
// independent data
   Z c_,l_;
      // c for 'cardinality' arbitrary non-negative integer,
      // 0 for the void set.
      // l for 'left', arbitrary integer

// dependent quantity
   Z r_;
      // r for 'right'
      // r_= l_ + c_ - 1
      // the right-most element of *this if *this is not void, 0 else.

// initializing the dependent data
   void ini_(){
      if (c_<=0){ c_=0; l_=0; r_=-1;}
      else r_=l_+c_-1;
   }

// enabling declaration macros
   typedef IvZ Type;
public:
// infra-structure by declaration macros
   CPM_IO
      // stream interaction
   CPM_ORDER
      // order-related functions
      // The void interval preceeds every non-void interval.
      // All void intervals are equal.
      // For non-void intervals we use lexicographic order of the pairs
```

```
      // (l_,r_).
   CPM_TEST_X
      // functions ran, test, hash, dis, abs, absSqr

   Word nameOf()const{ return "IvZ";}
      //: name of
      // class name of *this

   Word toWord()const;
      //: to word
      // text form of *this

// constructors

   explicit IvZ(Z n=0):c_(n),l_(1){ini_();}
      // The 'standard set' with n elements.
      // For n=0 it is the default constructor, which makes the void set.
      // Here, by definition, the smallest element of any non-void
      // standard set is 1 (not 0!).

   IvZ(Z i, Z j)
   {
      if (i<=j){ c_=j-i+1; l_=i; }
      else { c_=i-j+1; l_=j; }
      r_=l_+c_-1;
   }
      // the minimum contiguous subset of Z which contains both
      // i and j; no order restrictions to i,j. Notice that the
      // void set can't be obtained that way.

   IvZ(Z c, Z b, Z dummy):c_(c),l_(b){dummy; ini_();}
      // constructor which gives the cardinality and the first
      // element.

   IvZ(R a, R b);
      // construction of the smallest IvZ which contains a and b.

// basic properties
   bool isVoid()const{ return c_==0;}
      //: is void
      //':' means that the name is formed according to the CPM standard
      // scheme.

   Z car()const{ return c_;}
      //: cardinality
      // cardinality of the set given by *this

   Z width()const{ return c_==0 ? 0 : c_-1;}
      //:: width
      //'::' means that the name is longer than the one formed according
```

```
      // to the CPM standard abbreviation scheme. In most cases
      // no abbreviation at all.
      // distance between the last point and the first set.
      // Thus diameter of the set. Obviously the diameter
      // of a singlet set is zero, as is the diameter of a void set.

// The intention behind inf() and sup() is that
// iv==IvZ(iv.inf(),iv.sup()) for every non-void iv \in IvZ
// Thus inf() and sup() have to warn if called for a void set.
   Z inf()const;
      //: infimum

   Z sup()const;
      //: supremum

   Z mean()const{ return l_ + c_/2; }
      //: mean
      // IvZ(1,5).mean() = 3 = median of (1,2,3,4,5)
      // Iv(1,4).mean() = 3 lowerst integer which is >= median of
      // (1,2,3,4)

   Z b()const{ return l_;}
      //. begin
      //'.' means that the name is shorter than the one formed according
      // to the CPM standard abbreviation scheme. In most cases
      // abbreviation to the first letter of the full name.

   Z n()const{ return r_+1;}
      //. next
      // relative to the numbers belonging to *this
      // this is the next number with respect to standard order

   Z e()const{ return r_;}
      //. end
      // actually the last number belonging to *this
      // if *this is non-void as a set.
      // Looping over an IvZ ivz can allways be done as
      // for (Z i=ivz.b();i<=ivz.e();++i) ... ivz[i]....
      // or, equally well
      // for (Z i=ivz.b();i<ivz.n();++i) ... ivz[i]....

   Z fir()const{ return l_;}
      //: first

   Z last()const{ return r_;}
      //: last

   Z operator[](Z i)const{ if (i<=1) return l_; else return r_;}
      //: []
```

```
Z gap(IvZ const& iv);
   //: gap
   // Distance between the sets *this and iv.
   // Here the understanding is that the distance from a void
   // set to any set is 0.

Z relPos(Z i)const
   //: relative position
   // gives a code for the location of i relative to *this.
{
   if (i<l_) return -1;
   else if (i>r_) return 1;
   else return 0;
}

Z relInd(Z i)const{ return i-l_;}
   //: relative index
   // gives the difference of i with respect to the left end
   // ('the origin') of *this.
   // If *this is void the result is meaningless. Nevertheless
   // for efficiency reasons we do not test voidness.

Z ri(Z i)const{ return i-l_;}
   //. relative index
   // short form of relInd

Z cyc(Z i)const
   //: cyclic
   // Z-->Z, i|-->iv.cyc(i) is a periodic function with period
   // iv.car(), which coincides with the idendity function id on
   // the set iv.
{
   return i+c_*Z(floor(double(r_-i)/c_));
}

Z con(Z i)const
   //: constant
   // Z-->Z, i|-->iv.con(i) is the identity function id on
   // iv and defined as the constant continuation of id
   // outside of iv.
{
   if (i<=l_) return l_;
   else if (i>=r_) return r_;
   else return i;
}

IvZ leftOf(Z i)const
   //: left of
   // Returns the part of *this the elements j of which
   // satisfy j < i (and the void interval else).
```

```
   {
      if (l_ >= i) return IvZ();
      if (r_ < i) return *this;
      return IvZ(l_,i-1);
   }

   IvZ rigOf(Z i)const
      //: right of
      // Returns the part of *this the elements j of which
      // satisfy  i < j (and the void interval else).
   {
      if (r_ <= i) return IvZ();
      if (l_ > i) return *this;
      return IvZ(i+1,r_);
   }

   IvZ operator|(IvZ const& iv)const;
      // minimal interval that contains the union
      // join or l.u.b. (lowest upper bound) in lattice terminology

   IvZ join(IvZ const& iv)const{ return (*this)|iv;}
      //:join

   IvZ operator&(IvZ const& iv)const;
      // section, intersection
      // meet or g.l.b (greates lower bound) in lattice terminology.

   IvZ meet(IvZ const& iv)const{ return (*this)&iv;}
      //: meet

// set operations may yield a pair of IvZs as result
   T2<IvZ> setUnn(IvZ const& iv)const;
      //: set union
      // Consider T2<IvZ> res = iv1.setUnn(iv2);
      // If the set union of iv1 and iv2 is contiguous, the corresponding
      // IvZ-object will be res.e(), and res.b() is set to be void.
      // This order is chosen to have res.b() < res.e() in all cases.
      // Notice that the void IvZ preceeds all IvZ's.
      // If the set union is not contiguous it determines two IvZ-objects
      // which will be stored in res such that res.b() < res.e().

   T2<IvZ> setSec(IvZ const& iv)const{return T2<IvZ>(IvZ(),meet(iv));}
      //: set section
      // Coincides with lattice meet, which it returns as the second
      // component of the result. The first component is set to be void.
      // (Then the components are ordered).

   T2<IvZ> setDif(IvZ const& iv)const;
      //: set difference
      // Consider T2<IvZ> res = iv1.setDif(iv2);
```

```
      // If the set difference iv1\iv2 is contiguous, the corresponding
      // IvZ-object will be res.e(), and res.b() is set to be void.
      // See setUnn for reasons.
      // If the set union is not contiguous it determines two IvZ-objects
      // which will be stored in res such that res.b() < res.e().

// further functions
   IvZ app(IvZ const& iv)const
   {
      if (c_==0) return iv;
      else return IvZ(c_+iv.c_,l_);
   }
      //: append
      // We return a IvZ which is obtained from *this (if it is not void)
      // by appending iv.car() elements at the right end. If this is void
      // we simply return iv.

   bool hasElm(Z i)const{ return l_<=i && i<=r_;}
      //: has element
      // answers the question whether point i is an element of the
      // interval *this. S<Z> has a member function of the same
      // signature.

   bool ni(Z i)const{ return hasElm(i);}
      //: ni
      // 'in' reversed, from LATEX symbol \ni for the mirror image of
      // the epsilon-like 'is element'-symbol

   bool contains(IvZ const& iv)const;
      //:: contains
      // says whether iv is a subset (true subset 'or equal') of *this
      // or not.

   bool isSubSetOf(IvZ const& iv)const{  return iv.contains(*this);}
      //: is sub-set of
      // returns true iff *this is a subset of iv

   bool isSupSetOf(IvZ const& iv)const{ return (*this).contains(iv);}
      //: is super-set of
      // returns true iff s is a subset of *this

   IvZ operator+(Z s)const{return IvZ(l_+s,r_+s);}
      //: +
      // Returns a copy of *this with both ends shifted by s.

   IvZ operator-(Z s)const{return IvZ(l_-s,r_-s);}
      //: -
      // Returns a copy of *this with both ends shifted by -s.

   IvZ& operator+=(Z s){ l_+=s; r_+=s; return *this;}
```

```
      //: +=

   IvZ& operator-=(Z s){ l_-=s; r_-=s; return *this;}
      //: -=

   IvZ operator+(IvZ const& iv)const
      //: +
      // Range of i+j for i \in *this and j \in iv.
   { return IvZ(b()+iv.b(),e()+iv.e());}

   IvZ operator-(IvZ const& iv)const
      //: -
      // Range of i-j for i \in *this and j \in iv.
   { return IvZ(b()-iv.b(),e()-iv.e());}

   IvZ& b_(Z i){Z ir=i-l_; return operator+=(ir);}
      //. set begin
      // Changes b() into i and returns a reference to the
      // modified object.
      // Recall that the names of non-constant functions end in '_'.

   IvZ& e_(Z i){Z ir=i-r_; return operator+=(ir);}
      //. set end
      // Changes e() into i and returns a reference to the
      // modified object.

   Z ranSel(Z j=0)const;
      //: random selection
      // Returns a 'randomly' selected element in *this
      // Here the argument j plays the same role as in
      //    R CpmRoot::randomR(Z j=0);
      // (see cpmnumbers.h)
      // Notice that there is also a function ran(Z) which
      // returns a IvZ, so that the present function can not
      // be named simply ran.
      // If *this is void, we return 0 and issue a warning.

   static Z testLatIdn(Z complexity=512);
      //: test lattice identities
      // Tests the complete set of lattice identities.
      // See e.g. S. Mac Lane, G. Birkhoff: Algebra MacMillan 1968
      // p. 487 Theorem 4.
 };

//////////////////////////// Implementation ////////////////////////////
// see also cpmzinterval.cpp
inline Z IvZ::inf()const
{
   if (c_==0)
      cpmwarning("Iv::inf() undefined for void instance, b() returned");
```

```
   return l_;
}
inline Z IvZ::sup()const
{
   if (c_==0)
      cpmwarning("Iv::sup() undefined for void instance, e() returned");
   return r_;
}

} // namespace

#endif
```

46 cpmzinterval.cpp

```
//? cpmzinterval.cpp
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


#include <cpmzinterval.h>
#include <cpmtypes.h>

using CpmRoot::Z;
using CpmRoot::R;
using CpmRoot::Word;
using CpmRoot::randomR;
using CpmArrays::IvZ;
using CpmArrays::T2;
using namespace CpmStd;

//////////////////////////// class IvZ ////////////////////////////////

IvZ::IvZ(R a, R b)
   {
      Z i,j;
      if (a<=b) {i=cpmrnd(cpmfloor(a)); j=cpmrnd(cpmceil(b));}
      else {j=cpmrnd(cpmfloor(a)); i=cpmrnd(cpmceil(b));}
      *this=IvZ(i,j);
   }

Z IvZ::gap(IvZ const& iv)
{
   if (isVoid()) return 0;
   if (iv.isVoid()) return 0;
   if (r_<iv.l_) return iv.l_-r_;
   else if (iv.r_<l_) return l_-iv.r_;
   else return 0;
}
```

```
bool IvZ::contains(const IvZ& iv)const
{
   if (iv.isVoid()) return true;
      // the void set is subset of every set, even of the void set
   else{ // now iv is non-void
      if (isVoid()) return false;
         // a non-void set iv is never a subset of the void set
      else return l_<=iv.l_ && iv.r_<=r_;
   }
}

IvZ IvZ::operator|(IvZ const& iv)const
{
   if (iv.isVoid()) return *this;
      // the union with the void set is the original
   else{ // now iv is non-void
      if (isVoid()) return iv;
         // a non-void set iv is never a subset of the void set
      else { // now both *this and iv are non-void
         Z i1=inf();
         Z s1=sup();
         Z i2=iv.inf();
         Z s2=iv.sup();
         Z i=CpmRootX::inf<Z>(i1,i2);
         Z s=CpmRootX::sup<Z>(s1,s2);
         return IvZ(i,s);
      }
   }
}

IvZ IvZ::operator&(IvZ const& iv)const
{
   if (isVoid()||iv.isVoid()) return IvZ();
      // the section with the void set is the void set
   else{ // now both *this and iv are non-void
      Z s1=sup();
      Z i2=iv.inf();
      if (s1<i2) return IvZ();
      else{
         Z i1=inf();
         Z s2=iv.sup();
         if (i1>s2) return IvZ();
         else{ // now there is an overlap
             Z i=CpmRootX::sup<Z>(i1,i2);
             Z s=CpmRootX::inf<Z>(s1,s2);
             return IvZ(i,s);
         }
      }
   }
}
```

```
T2<IvZ> IvZ::setUnn(IvZ const& iv)const
{
   if (iv.isVoid()) return T2<IvZ>(IvZ(),*this);
   if (isVoid()) return T2<IvZ>(IvZ(),iv);
   if (contains(iv)) return T2<IvZ>(IvZ(),*this);
   if (iv.contains(*this)) return T2<IvZ>(IvZ(),iv);
   if (meet(iv).isVoid()){
      if (*this < iv) return T2<IvZ>(*this,iv);
      else return T2<IvZ>(iv,*this); // then iv<*this
   }
   return T2<IvZ>(IvZ(),join(iv));
}

T2<IvZ> IvZ::setDif(IvZ const& iv)const
{
   if (iv.isVoid()) return T2<IvZ>(*this,IvZ());
   if (isVoid()) return T2<IvZ>(IvZ(),IvZ());
   if (contains(iv)){
      IvZ ir=rigOf(iv.e());
      IvZ il=leftOf(iv.b());
      if (!ir.isVoid())
         return T2<IvZ>(il,ir);
      else
         return T2<IvZ>(ir,il);
   }
   if (iv.contains(*this)) return T2<IvZ>(IvZ(),IvZ());
   if (meet(iv).isVoid()){ return T2<IvZ>(IvZ(),*this);}
   if (iv<*this) return T2<IvZ>(IvZ(),rigOf(iv.e()));
   else return T2<IvZ>(IvZ(),leftOf(iv.b()));
}

Z IvZ::ranSel(Z j)const
{
   if (c_==0){
      cpmwarning("IvZ::ranSel(Z): instance is void, 0 returned");
      return 0;
   }
   else if (c_==1){
      return l_;
   }
   else{
      R r=randomR(j);
      Z res=l_+cpmtoz(r*c_);
      // (Z)(r*c_) takes values 0,1,...c_-1 with equal probabilities.
      // These are c_ values, which is OK.
      if (res>r_){
         cpmwarning("IvZ::ranSel(Z) proposed a value outside of *this, r_\
 returned");
         res=r_;
```

```
      }
      return res;
   }
}

// CPM_IO

bool IvZ::prnOn(ostream& str)const
{
   cpmwt("IvZ");
   cpmp(c_);
   cpmp(l_);
   return true;
}

bool IvZ::scanFrom(istream& str)
{
   cpms(c_);
   cpms(l_);
   ini_();
   return true;
}

// CPM_ORDER

Z IvZ::com(IvZ const& iv)const
{
   if (isVoid()){
      if (iv.isVoid()) return 0;
      else return 1;
   }
   if (l_<iv.l_) return 1;
   if (l_>iv.l_) return -1;
   if (r_<iv.r_) return 1;
   if (r_>iv.r_) return -1;
   return 0;
}

// CPM_DESCRIPTORS (not exactly since not virtual)

Word IvZ::toWord()const
{
   if (c_==0) return Word("IvZ()");
   ostringstream ost;
   ost<<"IvZ("<<l_<<","<<r_<<")";
   return Word(ost);
}

// CPM_TEST_X, functions already in CPM_TEST
```

```
IvZ IvZ::test(Z complexity)const{ return IvZ(complexity);}

Z IvZ::hash()const
{ return CpmRoot::hashT<Z>(l_)+CpmRoot::hashT<Z>(r_);}

namespace{
   R f(R x){ return x*2-1;}
   // values between -1 and +1
}

IvZ IvZ::ran(Z j)const
{
   R x=R(l_), y=R(r_), c=R(c_);
   R r1,r2,r3;
   if (j==0){
      r1=randomR();
      r2=randomR();
      r3=randomR();
   }
   else{
      Z j3=j*3;
      r1=randomR(j3);
      r2=randomR(j3+1);
      r3=randomR(j3+2);
   }
   if (r1<0.1 && r2>0.9) return IvZ();
      // here one gets 1 percent void sets on average
   x*=f(r1);
   y*=f(r2);
   c*=f(r3);
   return IvZ(x+c,y+c);
}

// CPM_TEST_X , functions not in CPM_TEST

R IvZ::dis(IvZ const& iv)const{ return (*this)==iv ? 0. : 1.;}
R IvZ::abs()const{ return R(c_);}
R IvZ::absSqr()const{ R c(c_); return c*c;}

// testing lattice idendities for functions meet and join

Z IvZ::testLatIdn(Z complexity)
   {
      IvZ o0;
      IvZ o1=o0.test(complexity);
      Z n=complexity;
      // Z n=cpmtoz(cpmsqrt(R(complexity)));
      // could be useful
      Z err=0;
      for (Z j=1;j<=n;++j){
```

```
      IvZ a=o1.ran();
      IvZ b=o1.ran();
      IvZ c=o1.ran();
      //cout<<a.toWord()<<endl;
      //cout<<b.toWord()<<endl;
      //cout<<c.toWord()<<endl;
      bool bb;
      bb=a.meet(b)==b.meet(a);
      if (!bb) err++;
      bb=a.join(b)==b.join(a);
      if (!bb) err++;
      bb=a.meet(b.meet(c))==(a.meet(b)).meet(c);
      if (!bb) err++;
      bb=a.join(b.join(c))==(a.join(b)).join(c);
      if (!bb) err++;
      bb=a.meet(a)==a;
      if (!bb) err++;
      bb=a.join(a)==a;
      if (!bb) err++;
      bb=a.meet(a.join(b))==a.join(a.meet(b));
      if (!bb) err++;
      bb=a.meet(a.join(b))==a;
      if (!bb) err++;
   }

   cout<<"IvZ::testLattice(Z), complexity="<<complexity<<
      ", err="<<err<<endl;
   return err;
}
```

47 features.h

```
/* Copyright (C) 1991-2018 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   The GNU C Library is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
   Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public
   License along with the GNU C Library; if not, see
   <http://www.gnu.org/licenses/>.  */

#ifndef _FEATURES_H
#define _FEATURES_H 1

/* These are defined by the user (or the compiler)
   to specify the desired environment:

   __STRICT_ANSI__ ISO Standard C.
   _ISOC99_SOURCE Extensions to ISO C89 from ISO C99.
   _ISOC11_SOURCE Extensions to ISO C99 from ISO C11.
   __STDC_WANT_LIB_EXT2__
Extensions to ISO C99 from TR 27431-2:2010.
   __STDC_WANT_IEC_60559_BFP_EXT__
Extensions to ISO C11 from TS 18661-1:2014.
   __STDC_WANT_IEC_60559_FUNCS_EXT__
Extensions to ISO C11 from TS 18661-4:2015.
   __STDC_WANT_IEC_60559_TYPES_EXT__
Extensions to ISO C11 from TS 18661-3:2015.

   _POSIX_SOURCE IEEE Std 1003.1.
   _POSIX_C_SOURCE If ==1, like _POSIX_SOURCE; if >=2 add IEEE Std 1003.2;
if >=199309L, add IEEE Std 1003.1b-1993;
if >=199506L, add IEEE Std 1003.1c-1995;
if >=200112L, all of IEEE 1003.1-2004
if >=200809L, all of IEEE 1003.1-2008
   _XOPEN_SOURCE Includes POSIX and XPG things.  Set to 500 if
Single Unix conformance is wanted, to 600 for the
sixth revision, to 700 for the seventh revision.
   _XOPEN_SOURCE_EXTENDED XPG things and X/Open Unix extensions.
   _LARGEFILE_SOURCE Some more functions for correct standard I/O.
   _LARGEFILE64_SOURCE Additional functionality from LFS for large files.
```

```
   _FILE_OFFSET_BITS=N Select default filesystem interface.
   _ATFILE_SOURCE Additional *at interfaces.
   _GNU_SOURCE All of the above, plus GNU extensions.
   _DEFAULT_SOURCE The default set of features (taking precedence over
__STRICT_ANSI__).

   _FORTIFY_SOURCE Add security hardening to many library functions.
Set to 1 or 2; 2 performs stricter checks than 1.

   _REENTRANT, _THREAD_SAFE
Obsolete; equivalent to _POSIX_C_SOURCE=199506L.

   The '-ansi' switch to the GNU C compiler, and standards conformance
   options such as '-std=c99', define __STRICT_ANSI__.  If none of
   these are defined, or if _DEFAULT_SOURCE is defined, the default is
   to have _POSIX_SOURCE set to one and _POSIX_C_SOURCE set to
   200809L, as well as enabling miscellaneous functions from BSD and
   SVID.  If more than one of these are defined, they accumulate.  For
   example __STRICT_ANSI__, _POSIX_SOURCE and _POSIX_C_SOURCE together
   give you ISO C, 1003.1, and 1003.2, but nothing else.

   These are defined by this file and are used by the
   header files to decide what to declare or define:

   __GLIBC_USE (F) Define things from feature set F.  This is defined
to 1 or 0; the subsequent macros are either defined
or undefined, and those tests should be moved to
__GLIBC_USE.
   __USE_ISOC11 Define ISO C11 things.
   __USE_ISOC99 Define ISO C99 things.
   __USE_ISOC95 Define ISO C90 AMD1 (C95) things.
   __USE_ISOCXX11 Define ISO C++11 things.
   __USE_POSIX Define IEEE Std 1003.1 things.
   __USE_POSIX2 Define IEEE Std 1003.2 things.
   __USE_POSIX199309 Define IEEE Std 1003.1, and .1b things.
   __USE_POSIX199506 Define IEEE Std 1003.1, .1b, .1c and .1i things.
   __USE_XOPEN Define XPG things.
   __USE_XOPEN_EXTENDED Define X/Open Unix things.
   __USE_UNIX98 Define Single Unix V2 things.
   __USE_XOPEN2K        Define XPG6 things.
   __USE_XOPEN2KXSI     Define XPG6 XSI things.
   __USE_XOPEN2K8       Define XPG7 things.
   __USE_XOPEN2K8XSI    Define XPG7 XSI things.
   __USE_LARGEFILE Define correct standard I/O things.
   __USE_LARGEFILE64 Define LFS things with separate names.
   __USE_FILE_OFFSET64 Define 64bit interface as default.
   __USE_MISC Define things from 4.3BSD or System V Unix.
   __USE_ATFILE Define *at interfaces and AT_* constants for them.
   __USE_GNU Define GNU extensions.
   __USE_FORTIFY_LEVEL Additional security measures used, according to level.
```

```
   The macros '__GNU_LIBRARY__', '__GLIBC__', and '__GLIBC_MINOR__' are
   defined by this file unconditionally.  '__GNU_LIBRARY__' is provided
   only for compatibility.  All new code should use the other symbols
   to test for features.

   All macros listed above as possibly being defined by this file are
   explicitly undefined if they are not explicitly defined.
   Feature-test macros that are not defined by the user or compiler
   but are implied by the other feature-test macros defined (or by the
   lack of any definitions) are defined by the file.

   ISO C feature test macros depend on the definition of the macro
   when an affected header is included, not when the first system
   header is included, and so they are handled in
   <bits/libc-header-start.h>, which does not have a multiple include
   guard.  Feature test macros that can be handled from the first
   system header included are handled here.  */


/* Undefine everything, so we get a clean slate.  */
#undef __USE_ISOC11
#undef __USE_ISOC99
#undef __USE_ISOC95
#undef __USE_ISOCXX11
#undef __USE_POSIX
#undef __USE_POSIX2
#undef __USE_POSIX199309
#undef __USE_POSIX199506
#undef __USE_XOPEN
#undef __USE_XOPEN_EXTENDED
#undef __USE_UNIX98
#undef __USE_XOPEN2K
#undef __USE_XOPEN2KXSI
#undef __USE_XOPEN2K8
#undef __USE_XOPEN2K8XSI
#undef __USE_LARGEFILE
#undef __USE_LARGEFILE64
#undef __USE_FILE_OFFSET64
#undef __USE_MISC
#undef __USE_ATFILE
#undef __USE_GNU
#undef __USE_FORTIFY_LEVEL
#undef __KERNEL_STRICT_NAMES
#undef __GLIBC_USE_DEPRECATED_GETS

/* Suppress kernel-name space pollution unless user expressedly asks
   for it.  */
#ifndef _LOOSE_KERNEL_NAMES
# define __KERNEL_STRICT_NAMES
```

```
#endif

/* Convenience macro to test the version of gcc.
   Use like this:
   #if __GNUC_PREREQ (2,8)
   ... code requiring gcc 2.8 or later ...
   #endif
   Note: only works for GCC 2.0 and later, because __GNUC_MINOR__ was
   added in 2.0.  */
#if defined __GNUC__ && defined __GNUC_MINOR__
# define __GNUC_PREREQ(maj, min) \
((__GNUC__ << 16) + __GNUC_MINOR__ >= ((maj) << 16) + (min))
#else
# define __GNUC_PREREQ(maj, min) 0
#endif

/* Similarly for clang.  Features added to GCC after version 4.2 may
   or may not also be available in clang, and clang's definitions of
   __GNUC(_MINOR)__ are fixed at 4 and 2 respectively.  Not all such
   features can be queried via __has_extension/__has_feature.  */
#if defined __clang_major__ && defined __clang_minor__
# define __glibc_clang_prereq(maj, min) \
  ((__clang_major__ << 16) + __clang_minor__ >= ((maj) << 16) + (min))
#else
# define __glibc_clang_prereq(maj, min) 0
#endif

/* Whether to use feature set F.  */
#define __GLIBC_USE(F) __GLIBC_USE_ ## F

/* _BSD_SOURCE and _SVID_SOURCE are deprecated aliases for
   _DEFAULT_SOURCE.  If _DEFAULT_SOURCE is present we do not
   issue a warning; the expectation is that the source is being
   transitioned to use the new macro.  */
#if (defined _BSD_SOURCE || defined _SVID_SOURCE) \
    && !defined _DEFAULT_SOURCE
# warning "_BSD_SOURCE and _SVID_SOURCE are deprecated, use _DEFAULT_SOURCE"
# undef  _DEFAULT_SOURCE
# define _DEFAULT_SOURCE 1
#endif

/* If _GNU_SOURCE was defined by the user, turn on all the other features.  */
#ifdef _GNU_SOURCE
# undef  _ISOC95_SOURCE
# define _ISOC95_SOURCE 1
# undef  _ISOC99_SOURCE
# define _ISOC99_SOURCE 1
# undef  _ISOC11_SOURCE
# define _ISOC11_SOURCE 1
# undef  _POSIX_SOURCE
```

```
# define _POSIX_SOURCE 1
# undef  _POSIX_C_SOURCE
# define _POSIX_C_SOURCE 200809L
# undef  _XOPEN_SOURCE
# define _XOPEN_SOURCE 700
# undef  _XOPEN_SOURCE_EXTENDED
# define _XOPEN_SOURCE_EXTENDED 1
# undef  _LARGEFILE64_SOURCE
# define _LARGEFILE64_SOURCE 1
# undef  _DEFAULT_SOURCE
# define _DEFAULT_SOURCE 1
# undef  _ATFILE_SOURCE
# define _ATFILE_SOURCE 1
#endif

/* If nothing (other than _GNU_SOURCE and _DEFAULT_SOURCE) is defined,
   define _DEFAULT_SOURCE.  */
#if (defined _DEFAULT_SOURCE \
     || (!defined __STRICT_ANSI__ \
 && !defined _ISOC99_SOURCE \
 && !defined _POSIX_SOURCE && !defined _POSIX_C_SOURCE \
 && !defined _XOPEN_SOURCE))
# undef  _DEFAULT_SOURCE
# define _DEFAULT_SOURCE 1
#endif

/* This is to enable the ISO C11 extension.  */
#if (defined _ISOC11_SOURCE \
     || (defined __STDC_VERSION__ && __STDC_VERSION__ >= 201112L))
# define __USE_ISOC11 1
#endif

/* This is to enable the ISO C99 extension.  */
#if (defined _ISOC99_SOURCE || defined _ISOC11_SOURCE \
     || (defined __STDC_VERSION__ && __STDC_VERSION__ >= 199901L))
# define __USE_ISOC99 1
#endif

/* This is to enable the ISO C90 Amendment 1:1995 extension.  */
#if (defined _ISOC99_SOURCE || defined _ISOC11_SOURCE \
     || (defined __STDC_VERSION__ && __STDC_VERSION__ >= 199409L))
# define __USE_ISOC95 1
#endif

#ifdef __cplusplus
/* This is to enable compatibility for ISO C++17.  */
# if __cplusplus >= 201703L
#  define __USE_ISOC11 1
# endif
/* This is to enable compatibility for ISO C++11.
```

```
   Check the temporary macro for now, too.  */
# if __cplusplus >= 201103L || defined __GXX_EXPERIMENTAL_CXX0X__
#  define __USE_ISOCXX11 1
#  define __USE_ISOC99 1
# endif
#endif

/* If none of the ANSI/POSIX macros are defined, or if _DEFAULT_SOURCE
   is defined, use POSIX.1-2008 (or another version depending on
   _XOPEN_SOURCE).  */
#ifdef _DEFAULT_SOURCE
# if !defined _POSIX_SOURCE && !defined _POSIX_C_SOURCE
#  define __USE_POSIX_IMPLICITLY 1
# endif
# undef  _POSIX_SOURCE
# define _POSIX_SOURCE 1
# undef  _POSIX_C_SOURCE
# define _POSIX_C_SOURCE 200809L
#endif

#if ((!defined __STRICT_ANSI__ \
      || (defined _XOPEN_SOURCE && (_XOPEN_SOURCE - 0) >= 500)) \
     && !defined _POSIX_SOURCE && !defined _POSIX_C_SOURCE)
# define _POSIX_SOURCE 1
# if defined _XOPEN_SOURCE && (_XOPEN_SOURCE - 0) < 500
#  define _POSIX_C_SOURCE 2
# elif defined _XOPEN_SOURCE && (_XOPEN_SOURCE - 0) < 600
#  define _POSIX_C_SOURCE 199506L
# elif defined _XOPEN_SOURCE && (_XOPEN_SOURCE - 0) < 700
#  define _POSIX_C_SOURCE 200112L
# else
#  define _POSIX_C_SOURCE 200809L
# endif
# define __USE_POSIX_IMPLICITLY 1
#endif

/* Some C libraries once required _REENTRANT and/or _THREAD_SAFE to be
   defined in all multithreaded code.  GNU libc has not required this
   for many years.  We now treat them as compatibility synonyms for
   _POSIX_C_SOURCE=199506L, which is the earliest level of POSIX with
   comprehensive support for multithreaded code.  Using them never
   lowers the selected level of POSIX conformance, only raises it.  */
#if ((!defined _POSIX_C_SOURCE || (_POSIX_C_SOURCE - 0) < 199506L) \
     && (defined _REENTRANT || defined _THREAD_SAFE))
# define _POSIX_SOURCE   1
# undef  _POSIX_C_SOURCE
# define _POSIX_C_SOURCE 199506L
#endif

#if (defined _POSIX_SOURCE \
```

```
     || (defined _POSIX_C_SOURCE && _POSIX_C_SOURCE >= 1) \
     || defined _XOPEN_SOURCE)
# define __USE_POSIX 1
#endif

#if defined _POSIX_C_SOURCE && _POSIX_C_SOURCE >= 2 || defined _XOPEN_SOURCE
# define __USE_POSIX2 1
#endif

#if defined _POSIX_C_SOURCE && (_POSIX_C_SOURCE - 0) >= 199309L
# define __USE_POSIX199309 1
#endif

#if defined _POSIX_C_SOURCE && (_POSIX_C_SOURCE - 0) >= 199506L
# define __USE_POSIX199506 1
#endif

#if defined _POSIX_C_SOURCE && (_POSIX_C_SOURCE - 0) >= 200112L
# define __USE_XOPEN2K 1
# undef __USE_ISOC95
# define __USE_ISOC95 1
# undef __USE_ISOC99
# define __USE_ISOC99 1
#endif

#if defined _POSIX_C_SOURCE && (_POSIX_C_SOURCE - 0) >= 200809L
# define __USE_XOPEN2K8 1
# undef  _ATFILE_SOURCE
# define _ATFILE_SOURCE 1
#endif

#ifdef _XOPEN_SOURCE
# define __USE_XOPEN 1
# if (_XOPEN_SOURCE - 0) >= 500
#  define __USE_XOPEN_EXTENDED 1
#  define __USE_UNIX98 1
#  undef _LARGEFILE_SOURCE
#  define _LARGEFILE_SOURCE 1
#  if (_XOPEN_SOURCE - 0) >= 600
#   if (_XOPEN_SOURCE - 0) >= 700
#    define __USE_XOPEN2K8 1
#    define __USE_XOPEN2K8XSI 1
#   endif
#   define __USE_XOPEN2K 1
#   define __USE_XOPEN2KXSI 1
#   undef __USE_ISOC95
#   define __USE_ISOC95 1
#   undef __USE_ISOC99
#   define __USE_ISOC99 1
#  endif
```

```
# else
#  ifdef _XOPEN_SOURCE_EXTENDED
#   define __USE_XOPEN_EXTENDED 1
#  endif
# endif
#endif

#ifdef _LARGEFILE_SOURCE
# define __USE_LARGEFILE 1
#endif

#ifdef _LARGEFILE64_SOURCE
# define __USE_LARGEFILE64 1
#endif

#if defined _FILE_OFFSET_BITS && _FILE_OFFSET_BITS == 64
# define __USE_FILE_OFFSET64 1
#endif

#if defined _DEFAULT_SOURCE
# define __USE_MISC 1
#endif

#ifdef _ATFILE_SOURCE
# define __USE_ATFILE 1
#endif

#ifdef _GNU_SOURCE
# define __USE_GNU 1
#endif

#if defined _FORTIFY_SOURCE && _FORTIFY_SOURCE > 0 \
    && __GNUC_PREREQ (4, 1) && defined __OPTIMIZE__ && __OPTIMIZE__ > 0
# if _FORTIFY_SOURCE > 1
#  define __USE_FORTIFY_LEVEL 2
# else
#  define __USE_FORTIFY_LEVEL 1
# endif
#else
# define __USE_FORTIFY_LEVEL 0
#endif

/* The function 'gets' existed in C89, but is impossible to use
   safely.  It has been removed from ISO C11 and ISO C++14.  Note: for
   compatibility with various implementations of <cstdio>, this test
   must consider only the value of __cplusplus when compiling C++.  */
#if defined __cplusplus ? __cplusplus >= 201402L : defined __USE_ISOC11
# define __GLIBC_USE_DEPRECATED_GETS 0
#else
# define __GLIBC_USE_DEPRECATED_GETS 1
```

```
#endif

/* Get definitions of __STDC_* predefined macros, if the compiler has
   not preincluded this header automatically.  */
#include <stdc-predef.h>

/* This macro indicates that the installed library is the GNU C Library.
   For historic reasons the value now is 6 and this will stay from now
   on.  The use of this variable is deprecated.  Use __GLIBC__ and
   __GLIBC_MINOR__ now (see below) when you want to test for a specific
   GNU C library version and use the values in <gnu/lib-names.h> to get
   the sonames of the shared libraries.  */
#undef  __GNU_LIBRARY__
#define __GNU_LIBRARY__ 6

/* Major and minor version number of the GNU C library package.  Use
   these macros to test for features in specific releases.  */
#define __GLIBC__ 2
#define __GLIBC_MINOR__ 27

#define __GLIBC_PREREQ(maj, min) \
((__GLIBC__ << 16) + __GLIBC_MINOR__ >= ((maj) << 16) + (min))

/* This is here only because every header file already includes this one.  */
#ifndef __ASSEMBLER__
# ifndef _SYS_CDEFS_H
#  include <sys/cdefs.h>
# endif

/* If we don't have __REDIRECT, prototypes will be missing if
   __USE_FILE_OFFSET64 but not __USE_LARGEFILE[64]. */
# if defined __USE_FILE_OFFSET64 && !defined __REDIRECT
#  define __USE_LARGEFILE 1
#  define __USE_LARGEFILE64 1
# endif

#endif /* !ASSEMBLER */

/* Decide whether we can define 'extern inline' functions in headers.  */
#if __GNUC_PREREQ (2, 7) && defined __OPTIMIZE__ \
    && !defined __OPTIMIZE_SIZE__ && !defined __NO_INLINE__ \
    && defined __extern_inline
# define __USE_EXTERN_INLINES 1
#endif


/* This is here only because every header file already includes this one.
   Get the definitions of all the appropriate `__stub_FUNCTION' symbols.
   <gnu/stubs.h> contains `#define __stub_FUNCTION' when FUNCTION is a stub
   that will always return failure (and set errno to ENOSYS).  */
```

```
#include <gnu/stubs.h>

#endif /* features.h  */
```

48 mpfr.h

```
/* mpfr.h -- Include file for mpfr.

Copyright 1999-2018 Free Software Foundation, Inc.
Contributed by the AriC and Caramba projects, INRIA.

This file is part of the GNU MPFR Library.

The GNU MPFR Library is free software; you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation; either version 3 of the License, or (at your
option) any later version.

The GNU MPFR Library is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser General Public
License for more details.

You should have received a copy of the GNU Lesser General Public License
along with the GNU MPFR Library; see the file COPYING.LESSER.  If not, see
http://www.gnu.org/licenses/ or write to the Free Software Foundation, Inc.,
51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA. */

#ifndef __MPFR_H
#define __MPFR_H

/* Define MPFR version number */
#define MPFR_VERSION_MAJOR 4
#define MPFR_VERSION_MINOR 0
#define MPFR_VERSION_PATCHLEVEL 1
#define MPFR_VERSION_STRING "4.0.1"

/* User macros:
   MPFR_USE_FILE:        Define it to make MPFR define functions dealing
                         with FILE* (auto-detect).
   MPFR_USE_INTMAX_T:    Define it to make MPFR define functions dealing
                         with intmax_t (auto-detect).
   MPFR_USE_VA_LIST:     Define it to make MPFR define functions dealing
                         with va_list (auto-detect).
   MPFR_USE_C99_FEATURE: Define it to 1 to make MPFR support C99-feature
                         (auto-detect), to 0 to bypass the detection.
   MPFR_USE_EXTENSION:   Define it to make MPFR use GCC extension to
                         reduce warnings.
   MPFR_USE_NO_MACRO:    Define it to make MPFR remove any overriding
                         function macro.
*/

/* Macros dealing with MPFR VERSION */
```

```
#define MPFR_VERSION_NUM(a,b,c) (((a) << 16L) | ((b) << 8) | (c))
#define MPFR_VERSION \
MPFR_VERSION_NUM(MPFR_VERSION_MAJOR,MPFR_VERSION_MINOR,MPFR_VERSION_PATCHLEVEL)

#include <gmp.h>

/* Avoid some problems with macro expansion if the user defines macros
   with the same name as keywords. By convention, identifiers and macro
   names starting with mpfr_ are reserved by MPFR. */
typedef void            mpfr_void;
typedef int             mpfr_int;
typedef unsigned int    mpfr_uint;
typedef long            mpfr_long;
typedef unsigned long   mpfr_ulong;
typedef size_t          mpfr_size_t;

/* Global (possibly TLS) flags. Might also be used in an mpfr_t in the
   future (there would be room as mpfr_sign_t just needs 1 byte).
   TODO: The tests currently assume that the flags fits in an unsigned int;
   this should be cleaned up, e.g. by defining a function that outputs the
   flags as a string or by using the flags_out function (from tests/tests.c
   directly). */
typedef unsigned int    mpfr_flags_t;

/* Flags macros (in the public API) */
#define MPFR_FLAGS_UNDERFLOW 1
#define MPFR_FLAGS_OVERFLOW 2
#define MPFR_FLAGS_NAN 4
#define MPFR_FLAGS_INEXACT 8
#define MPFR_FLAGS_ERANGE 16
#define MPFR_FLAGS_DIVBY0 32
#define MPFR_FLAGS_ALL (MPFR_FLAGS_UNDERFLOW | \
                        MPFR_FLAGS_OVERFLOW  | \
                        MPFR_FLAGS_NAN       | \
                        MPFR_FLAGS_INEXACT   | \
                        MPFR_FLAGS_ERANGE    | \
                        MPFR_FLAGS_DIVBY0)

/* Definition of rounding modes (DON'T USE MPFR_RNDNA!).
   Warning! Changing the contents of this enum should be seen as an
   interface change since the old and the new types are not compatible
   (the integer type compatible with the enumerated type can even change,
   see ISO C99, 6.7.2.2#4), and in Makefile.am, AGE should be set to 0.

   MPFR_RNDU must appear just before MPFR_RNDD (see
   MPFR_IS_RNDUTEST_OR_RNDDNOTTEST in mpfr-impl.h).

   If you change the order of the rounding modes, please update the routines
   in texceptions.c which assume 0=RNDN, 1=RNDZ, 2=RNDU, 3=RNDD, 4=RNDA.
*/
```

```
typedef enum {
  MPFR_RNDN=0,  /* round to nearest, with ties to even */
  MPFR_RNDZ,    /* round toward zero */
  MPFR_RNDU,    /* round toward +Inf */
  MPFR_RNDD,    /* round toward -Inf */
  MPFR_RNDA,    /* round away from zero */
  MPFR_RNDF,    /* faithful rounding */
  MPFR_RNDNA=-1 /* round to nearest, with ties away from zero (mpfr_round) */
} mpfr_rnd_t;

/* kept for compatibility with MPFR 2.4.x and before */
#define GMP_RNDN MPFR_RNDN
#define GMP_RNDZ MPFR_RNDZ
#define GMP_RNDU MPFR_RNDU
#define GMP_RNDD MPFR_RNDD

/* Note: With the following default choices for _MPFR_PREC_FORMAT and
   _MPFR_EXP_FORMAT, mpfr_exp_t will be the same as [mp_exp_t] (at least
   up to GMP 5). */

/* Define precision: 1 (short), 2 (int) or 3 (long) (DON'T USE IT!) */
#ifndef _MPFR_PREC_FORMAT
# if __GMP_MP_SIZE_T_INT
#  define _MPFR_PREC_FORMAT 2
# else
#  define _MPFR_PREC_FORMAT 3
# endif
#endif

/* Define exponent: 1 (short), 2 (int), 3 (long) or 4 (intmax_t)
   (DON'T USE IT!) */
#ifndef _MPFR_EXP_FORMAT
# define _MPFR_EXP_FORMAT _MPFR_PREC_FORMAT
#endif

#if _MPFR_PREC_FORMAT > _MPFR_EXP_FORMAT
# error "mpfr_prec_t must not be larger than mpfr_exp_t"
#endif

/* Let's make mpfr_prec_t signed in order to avoid problems due to the
   usual arithmetic conversions when mixing mpfr_prec_t and mpfr_exp_t
   in an expression (for error analysis) if casts are forgotten. */
#if   _MPFR_PREC_FORMAT == 1
typedef short mpfr_prec_t;
typedef unsigned short mpfr_uprec_t;
#elif _MPFR_PREC_FORMAT == 2
typedef int   mpfr_prec_t;
typedef unsigned int   mpfr_uprec_t;
#elif _MPFR_PREC_FORMAT == 3
typedef long  mpfr_prec_t;
```

```
typedef unsigned long  mpfr_uprec_t;
#else
# error "Invalid MPFR Prec format"
#endif

/* Definition of precision limits without needing <limits.h> */
/* Note: The casts allows the expression to yield the wanted behavior
   for _MPFR_PREC_FORMAT == 1 (due to integer promotion rules). We
   also make sure that MPFR_PREC_MIN and MPFR_PREC_MAX have a signed
   integer type. The "- 256" allows more security, avoiding some
   integer overflows in extreme cases; ideally it should be useless. */
#define MPFR_PREC_MIN 1
#define MPFR_PREC_MAX ((mpfr_prec_t) ((((mpfr_uprec_t) -1) >> 1) - 256))

/* Definition of sign */
typedef int          mpfr_sign_t;

/* Definition of the exponent. _MPFR_EXP_FORMAT must be large enough
   so that mpfr_exp_t has at least 32 bits. */
#if   _MPFR_EXP_FORMAT == 1
typedef short mpfr_exp_t;
typedef unsigned short mpfr_uexp_t;
#elif _MPFR_EXP_FORMAT == 2
typedef int mpfr_exp_t;
typedef unsigned int mpfr_uexp_t;
#elif _MPFR_EXP_FORMAT == 3
typedef long mpfr_exp_t;
typedef unsigned long mpfr_uexp_t;
#elif _MPFR_EXP_FORMAT == 4
/* Note: in this case, intmax_t and uintmax_t must be defined before
   the inclusion of mpfr.h (we do not include <stdint.h> here because
   of some non-ISO C99 implementations that support these types). */
typedef intmax_t mpfr_exp_t;
typedef uintmax_t mpfr_uexp_t;
#else
# error "Invalid MPFR Exp format"
#endif

/* Definition of the standard exponent limits */
#define MPFR_EMAX_DEFAULT ((mpfr_exp_t) (((mpfr_ulong) 1 << 30) - 1))
#define MPFR_EMIN_DEFAULT (-(MPFR_EMAX_DEFAULT))

/* DON'T USE THIS! (For MPFR-public macros only, see below.)
   The mpfr_sgn macro uses the fact that __MPFR_EXP_NAN and __MPFR_EXP_ZERO
   are the smallest values. For a n-bit type, EXP_MAX is 2^(n-1)-1,
   EXP_ZERO is 1-2^(n-1), EXP_NAN is 2-2^(n-1), EXP_INF is 3-2^(n-1).
   This may change in the future. MPFR code should not be based on these
   representations (but if this is absolutely needed, protect the code
   with a static assertion). */
#define __MPFR_EXP_MAX ((mpfr_exp_t) (((mpfr_uexp_t) -1) >> 1))
```

```
#define __MPFR_EXP_NAN  (1 - __MPFR_EXP_MAX)
#define __MPFR_EXP_ZERO (0 - __MPFR_EXP_MAX)
#define __MPFR_EXP_INF  (2 - __MPFR_EXP_MAX)

/* Definition of the main structure */
typedef struct {
  mpfr_prec_t  _mpfr_prec;
  mpfr_sign_t  _mpfr_sign;
  mpfr_exp_t   _mpfr_exp;
  mp_limb_t   *_mpfr_d;
} __mpfr_struct;

/* Compatibility with previous types of MPFR */
#ifndef mp_rnd_t
# define mp_rnd_t  mpfr_rnd_t
#endif
#ifndef mp_prec_t
# define mp_prec_t mpfr_prec_t
#endif

/*
   The represented number is
      _sign*(_d[k-1]/B+_d[k-2]/B^2+...+_d[0]/B^k)*2^_exp
   where k=ceil(_mp_prec/GMP_NUMB_BITS) and B=2^GMP_NUMB_BITS.

   For the msb (most significant bit) normalized representation, we must have
      _d[k-1]>=B/2, unless the number is singular.

   We must also have the last k*GMP_NUMB_BITS-_prec bits set to zero.
*/

typedef __mpfr_struct mpfr_t[1];
typedef __mpfr_struct *mpfr_ptr;
typedef const __mpfr_struct *mpfr_srcptr;

/* For those who need a direct and fast access to the sign field.
   However it is not in the API, thus use it at your own risk: it might
   not be supported, or change name, in further versions!
   Unfortunately, it must be defined here (instead of MPFR's internal
   header file mpfr-impl.h) because it is used by some macros below.
*/
#define MPFR_SIGN(x) ((x)->_mpfr_sign)

/* Stack interface */
typedef enum {
  MPFR_NAN_KIND     = 0,
  MPFR_INF_KIND     = 1,
  MPFR_ZERO_KIND    = 2,
  MPFR_REGULAR_KIND = 3
} mpfr_kind_t;
```

```
/* Free cache policy */
typedef enum {
  MPFR_FREE_LOCAL_CACHE  = 1,  /* 1 << 0 */
  MPFR_FREE_GLOBAL_CACHE = 2   /* 1 << 1 */
} mpfr_free_cache_t;

/* GMP defines:
    + size_t:                Standard size_t
    + __GMP_NOTHROW          For C++: can't throw .
    + __GMP_EXTERN_INLINE    Attribute for inline function.
    + __GMP_DECLSPEC_EXPORT  compiling to go into a DLL
    + __GMP_DECLSPEC_IMPORT  compiling to go into a application
*/
/* Extra MPFR defines */
#define __MPFR_SENTINEL_ATTR
#if defined (__GNUC__)
# if __GNUC__ >= 4
#  undef __MPFR_SENTINEL_ATTR
#  define __MPFR_SENTINEL_ATTR __attribute__ ((sentinel))
# endif
#endif

/* If the user hasn't requested his/her preference
   and if the intension of support by the compiler is C99
   and if the compiler is known to support the C99 feature
   then we can auto-detect the C99 support as OK.
   __GNUC__ is used to detect GNU-C, ICC & CLANG compilers.
   Currently we need only variadic macros, and they are present
   since GCC >= 3. We don't test library version since we don't
   use any feature present in the library too (except intmax_t,
   but they use another detection).*/
#ifndef MPFR_USE_C99_FEATURE
# if defined(__STDC_VERSION__) && (__STDC_VERSION__ >= 199901L)
#  if defined (__GNUC__)
#   if __GNUC__ >= 3
#    define MPFR_USE_C99_FEATURE 1
#   endif
#  endif
# endif
# ifndef MPFR_USE_C99_FEATURE
#  define MPFR_USE_C99_FEATURE 0
# endif
#endif

/* Support for WINDOWS Dll:
   Check if we are inside a MPFR build, and if so export the functions.
   Otherwise does the same thing as GMP */
#if defined(__MPFR_WITHIN_MPFR) && __GMP_LIBGMP_DLL
# define __MPFR_DECLSPEC __GMP_DECLSPEC_EXPORT
```

```
#else
# ifndef __GMP_DECLSPEC
#  define __GMP_DECLSPEC
# endif
# define __MPFR_DECLSPEC __GMP_DECLSPEC
#endif

/* Use MPFR_DEPRECATED to mark MPFR functions, types or variables as
   deprecated. Code inspired by Apache Subversion's svn_types.h file.
   For compatibility with MSVC, MPFR_DEPRECATED must be put before
   __MPFR_DECLSPEC (not at the end of the function declaration as
   documented in the GCC manual); GCC does not seem to care.
   Moreover, in order to avoid a warning when testing such functions,
   do something like:
     +-----------------------------------------
     |#ifndef _MPFR_NO_DEPRECATED_funcname
     |MPFR_DEPRECATED
     |#endif
     |__MPFR_DECLSPEC int mpfr_funcname (...);
     +-----------------------------------------
   and in the corresponding test program:
     +-----------------------------------------
     |#define _MPFR_NO_DEPRECATED_funcname
     |#include "mpfr-test.h"
     +-----------------------------------------
*/
#if defined(__GNUC__) && \
  (__GNUC__ >= 4 || (__GNUC__ == 3 && __GNUC_MINOR__ >= 1))
# define MPFR_DEPRECATED __attribute__ ((deprecated))
#elif defined(_MSC_VER) && _MSC_VER >= 1300
# define MPFR_DEPRECATED __declspec(deprecated)
#else
# define MPFR_DEPRECATED
#endif
/* TODO: Also define MPFR_EXPERIMENTAL for experimental functions?
   See SVN_EXPERIMENTAL in Subversion 1.9+ as an example:
   __attribute__((warning("..."))) can be used with GCC 4.3.1+ but
   not __llvm__, and __declspec(deprecated("...")) can be used with
   MSC as above. */

/* Note: In order to be declared, some functions need a specific
   system header to be included *before* "mpfr.h". If the user
   forgets to include the header, the MPFR function prototype in
   the user object file is not correct. To avoid wrong results,
   we raise a linker error in that case by changing their internal
   name in the library (prefixed by __gmpfr instead of mpfr). See
   the lines of the form "#define mpfr_xxx __gmpfr_xxx" below. */

#if defined (__cplusplus)
extern "C" {
```

```
#endif

__MPFR_DECLSPEC const char * mpfr_get_version (void);
__MPFR_DECLSPEC const char * mpfr_get_patches (void);
__MPFR_DECLSPEC int mpfr_buildopt_tls_p          (void);
__MPFR_DECLSPEC int mpfr_buildopt_float128_p     (void);
__MPFR_DECLSPEC int mpfr_buildopt_decimal_p      (void);
__MPFR_DECLSPEC int mpfr_buildopt_gmpinternals_p (void);
__MPFR_DECLSPEC int mpfr_buildopt_sharedcache_p  (void);
__MPFR_DECLSPEC const char * mpfr_buildopt_tune_case (void);

__MPFR_DECLSPEC mpfr_exp_t mpfr_get_emin     (void);
__MPFR_DECLSPEC int        mpfr_set_emin     (mpfr_exp_t);
__MPFR_DECLSPEC mpfr_exp_t mpfr_get_emin_min (void);
__MPFR_DECLSPEC mpfr_exp_t mpfr_get_emin_max (void);
__MPFR_DECLSPEC mpfr_exp_t mpfr_get_emax     (void);
__MPFR_DECLSPEC int        mpfr_set_emax     (mpfr_exp_t);
__MPFR_DECLSPEC mpfr_exp_t mpfr_get_emax_min (void);
__MPFR_DECLSPEC mpfr_exp_t mpfr_get_emax_max (void);

__MPFR_DECLSPEC void mpfr_set_default_rounding_mode (mpfr_rnd_t);
__MPFR_DECLSPEC mpfr_rnd_t mpfr_get_default_rounding_mode (void);
__MPFR_DECLSPEC const char * mpfr_print_rnd_mode (mpfr_rnd_t);

__MPFR_DECLSPEC void mpfr_clear_flags (void);
__MPFR_DECLSPEC void mpfr_clear_underflow (void);
__MPFR_DECLSPEC void mpfr_clear_overflow (void);
__MPFR_DECLSPEC void mpfr_clear_divby0 (void);
__MPFR_DECLSPEC void mpfr_clear_nanflag (void);
__MPFR_DECLSPEC void mpfr_clear_inexflag (void);
__MPFR_DECLSPEC void mpfr_clear_erangeflag (void);

__MPFR_DECLSPEC void mpfr_set_underflow (void);
__MPFR_DECLSPEC void mpfr_set_overflow (void);
__MPFR_DECLSPEC void mpfr_set_divby0 (void);
__MPFR_DECLSPEC void mpfr_set_nanflag (void);
__MPFR_DECLSPEC void mpfr_set_inexflag (void);
__MPFR_DECLSPEC void mpfr_set_erangeflag (void);

__MPFR_DECLSPEC int mpfr_underflow_p (void);
__MPFR_DECLSPEC int mpfr_overflow_p (void);
__MPFR_DECLSPEC int mpfr_divby0_p (void);
__MPFR_DECLSPEC int mpfr_nanflag_p (void);
__MPFR_DECLSPEC int mpfr_inexflag_p (void);
__MPFR_DECLSPEC int mpfr_erangeflag_p (void);

__MPFR_DECLSPEC void mpfr_flags_clear (mpfr_flags_t);
__MPFR_DECLSPEC void mpfr_flags_set (mpfr_flags_t);
__MPFR_DECLSPEC mpfr_flags_t mpfr_flags_test (mpfr_flags_t);
__MPFR_DECLSPEC mpfr_flags_t mpfr_flags_save (void);
```

```
__MPFR_DECLSPEC void mpfr_flags_restore (mpfr_flags_t,
                                         mpfr_flags_t);

__MPFR_DECLSPEC int mpfr_check_range (mpfr_ptr, int, mpfr_rnd_t);

__MPFR_DECLSPEC void mpfr_init2 (mpfr_ptr, mpfr_prec_t);
__MPFR_DECLSPEC void mpfr_init (mpfr_ptr);
__MPFR_DECLSPEC void mpfr_clear (mpfr_ptr);

__MPFR_DECLSPEC void
  mpfr_inits2 (mpfr_prec_t, mpfr_ptr, ...) __MPFR_SENTINEL_ATTR;
__MPFR_DECLSPEC void
  mpfr_inits (mpfr_ptr, ...) __MPFR_SENTINEL_ATTR;
__MPFR_DECLSPEC void
  mpfr_clears (mpfr_ptr, ...) __MPFR_SENTINEL_ATTR;

__MPFR_DECLSPEC int mpfr_prec_round (mpfr_ptr, mpfr_prec_t, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_can_round (mpfr_srcptr, mpfr_exp_t, mpfr_rnd_t,
                                    mpfr_rnd_t, mpfr_prec_t);
__MPFR_DECLSPEC mpfr_prec_t mpfr_min_prec (mpfr_srcptr);

__MPFR_DECLSPEC mpfr_exp_t mpfr_get_exp (mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_set_exp (mpfr_ptr, mpfr_exp_t);
__MPFR_DECLSPEC mpfr_prec_t mpfr_get_prec (mpfr_srcptr);
__MPFR_DECLSPEC void mpfr_set_prec (mpfr_ptr, mpfr_prec_t);
__MPFR_DECLSPEC void mpfr_set_prec_raw (mpfr_ptr, mpfr_prec_t);
__MPFR_DECLSPEC void mpfr_set_default_prec (mpfr_prec_t);
__MPFR_DECLSPEC mpfr_prec_t mpfr_get_default_prec (void);

__MPFR_DECLSPEC int mpfr_set_d (mpfr_ptr, double, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_set_flt (mpfr_ptr, float, mpfr_rnd_t);
#ifdef MPFR_WANT_DECIMAL_FLOATS
/* _Decimal64 is not defined in C++,
   cf https://gcc.gnu.org/bugzilla/show_bug.cgi?id=51364 */
__MPFR_DECLSPEC int mpfr_set_decimal64 (mpfr_ptr, _Decimal64, mpfr_rnd_t);
#endif
__MPFR_DECLSPEC int mpfr_set_ld (mpfr_ptr, long double, mpfr_rnd_t);
#ifdef MPFR_WANT_FLOAT128
__MPFR_DECLSPEC int mpfr_set_float128 (mpfr_ptr, __float128, mpfr_rnd_t);
__MPFR_DECLSPEC __float128 mpfr_get_float128 (mpfr_srcptr, mpfr_rnd_t);
#endif
__MPFR_DECLSPEC int mpfr_set_z (mpfr_ptr, mpz_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_set_z_2exp (mpfr_ptr, mpz_srcptr, mpfr_exp_t,
                                     mpfr_rnd_t);
__MPFR_DECLSPEC void mpfr_set_nan (mpfr_ptr);
__MPFR_DECLSPEC void mpfr_set_inf (mpfr_ptr, int);
__MPFR_DECLSPEC void mpfr_set_zero (mpfr_ptr, int);

#ifndef MPFR_USE_MINI_GMP
  /* mini-gmp does not provide mpf_t, we disable the following functions */
```

```
__MPFR_DECLSPEC int mpfr_set_f (mpfr_ptr, mpf_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_cmp_f (mpfr_srcptr, mpf_srcptr);
__MPFR_DECLSPEC int mpfr_get_f (mpf_ptr, mpfr_srcptr, mpfr_rnd_t);
#endif
__MPFR_DECLSPEC int mpfr_set_si (mpfr_ptr, long, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_set_ui (mpfr_ptr, unsigned long, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_set_si_2exp (mpfr_ptr, long, mpfr_exp_t, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_set_ui_2exp (mpfr_ptr, unsigned long, mpfr_exp_t,
                                      mpfr_rnd_t);
#ifndef MPFR_USE_MINI_GMP
  /* mini-gmp does not provide mpq_t, we disable the following functions */
__MPFR_DECLSPEC int mpfr_set_q (mpfr_ptr, mpq_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_mul_q (mpfr_ptr, mpfr_srcptr, mpq_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_div_q (mpfr_ptr, mpfr_srcptr, mpq_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_add_q (mpfr_ptr, mpfr_srcptr, mpq_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_sub_q (mpfr_ptr, mpfr_srcptr, mpq_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_cmp_q (mpfr_srcptr, mpq_srcptr);
__MPFR_DECLSPEC void mpfr_get_q (mpq_ptr q, mpfr_srcptr f);
#endif
__MPFR_DECLSPEC int mpfr_set_str (mpfr_ptr, const char *, int, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_init_set_str (mpfr_ptr, const char *, int,
                                       mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_set4 (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t, int);
__MPFR_DECLSPEC int mpfr_abs (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_set (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_neg (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_signbit (mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_setsign (mpfr_ptr, mpfr_srcptr, int, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_copysign (mpfr_ptr, mpfr_srcptr, mpfr_srcptr,
                                   mpfr_rnd_t);

__MPFR_DECLSPEC mpfr_exp_t mpfr_get_z_2exp (mpz_ptr, mpfr_srcptr);
__MPFR_DECLSPEC float mpfr_get_flt (mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC double mpfr_get_d (mpfr_srcptr, mpfr_rnd_t);
#ifdef MPFR_WANT_DECIMAL_FLOATS
__MPFR_DECLSPEC _Decimal64 mpfr_get_decimal64 (mpfr_srcptr, mpfr_rnd_t);
#endif
__MPFR_DECLSPEC long double mpfr_get_ld (mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC double mpfr_get_d1 (mpfr_srcptr);
__MPFR_DECLSPEC double mpfr_get_d_2exp (long*, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC long double mpfr_get_ld_2exp (long*, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_frexp (mpfr_exp_t*, mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC long mpfr_get_si (mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC unsigned long mpfr_get_ui (mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC char * mpfr_get_str (char*, mpfr_exp_t*, int, size_t,
                                     mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_get_z (mpz_ptr z, mpfr_srcptr f, mpfr_rnd_t);

__MPFR_DECLSPEC void mpfr_free_str (char *);
```

```
__MPFR_DECLSPEC int mpfr_urandom (mpfr_ptr, gmp_randstate_t, mpfr_rnd_t);
#ifndef _MPFR_NO_DEPRECATED_GRANDOM /* for the test of this function */
MPFR_DEPRECATED
#endif
__MPFR_DECLSPEC int mpfr_grandom (mpfr_ptr, mpfr_ptr, gmp_randstate_t,
                                  mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_nrandom (mpfr_ptr, gmp_randstate_t, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_erandom (mpfr_ptr, gmp_randstate_t, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_urandomb (mpfr_ptr, gmp_randstate_t);

__MPFR_DECLSPEC void mpfr_nextabove (mpfr_ptr);
__MPFR_DECLSPEC void mpfr_nextbelow (mpfr_ptr);
__MPFR_DECLSPEC void mpfr_nexttoward (mpfr_ptr, mpfr_srcptr);

#ifndef MPFR_USE_MINI_GMP
__MPFR_DECLSPEC int mpfr_printf (const char*, ...);
__MPFR_DECLSPEC int mpfr_asprintf (char**, const char*, ...);
__MPFR_DECLSPEC int mpfr_sprintf (char*, const char*, ...);
__MPFR_DECLSPEC int mpfr_snprintf (char*, size_t, const char*, ...);
#endif

__MPFR_DECLSPEC int mpfr_pow (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_pow_si (mpfr_ptr, mpfr_srcptr, long, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_pow_ui (mpfr_ptr, mpfr_srcptr, unsigned long,
                                 mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_ui_pow_ui (mpfr_ptr, unsigned long, unsigned long,
                                    mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_ui_pow (mpfr_ptr, unsigned long, mpfr_srcptr,
                                 mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_pow_z (mpfr_ptr, mpfr_srcptr, mpz_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_sqrt (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_sqrt_ui (mpfr_ptr, unsigned long, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_rec_sqrt (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_add (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_sub (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_mul (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_div (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_add_ui (mpfr_ptr, mpfr_srcptr, unsigned long,
                                 mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_sub_ui (mpfr_ptr, mpfr_srcptr, unsigned long,
                                 mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_ui_sub (mpfr_ptr, unsigned long, mpfr_srcptr,
                                 mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_mul_ui (mpfr_ptr, mpfr_srcptr, unsigned long,
                                 mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_div_ui (mpfr_ptr, mpfr_srcptr, unsigned long,
                                 mpfr_rnd_t);
```

```
__MPFR_DECLSPEC int mpfr_ui_div (mpfr_ptr, unsigned long, mpfr_srcptr,
                                 mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_add_si (mpfr_ptr, mpfr_srcptr, long, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_sub_si (mpfr_ptr, mpfr_srcptr, long, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_si_sub (mpfr_ptr, long, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_mul_si (mpfr_ptr, mpfr_srcptr, long, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_div_si (mpfr_ptr, mpfr_srcptr, long, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_si_div (mpfr_ptr, long, mpfr_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_add_d (mpfr_ptr, mpfr_srcptr, double, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_sub_d (mpfr_ptr, mpfr_srcptr, double, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_d_sub (mpfr_ptr, double, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_mul_d (mpfr_ptr, mpfr_srcptr, double, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_div_d (mpfr_ptr, mpfr_srcptr, double, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_d_div (mpfr_ptr, double, mpfr_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_sqr (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_const_pi (mpfr_ptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_const_log2 (mpfr_ptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_const_euler (mpfr_ptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_const_catalan (mpfr_ptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_agm (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_log (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_log2 (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_log10 (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_log1p (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_log_ui (mpfr_ptr, unsigned long, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_exp (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_exp2 (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_exp10 (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_expm1 (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_eint (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_li2 (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_cmp  (mpfr_srcptr, mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_cmp3 (mpfr_srcptr, mpfr_srcptr, int);
__MPFR_DECLSPEC int mpfr_cmp_d (mpfr_srcptr, double);
__MPFR_DECLSPEC int mpfr_cmp_ld (mpfr_srcptr, long double);
__MPFR_DECLSPEC int mpfr_cmpabs (mpfr_srcptr, mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_cmp_ui (mpfr_srcptr, unsigned long);
__MPFR_DECLSPEC int mpfr_cmp_si (mpfr_srcptr, long);
__MPFR_DECLSPEC int mpfr_cmp_ui_2exp (mpfr_srcptr, unsigned long, mpfr_exp_t);
__MPFR_DECLSPEC int mpfr_cmp_si_2exp (mpfr_srcptr, long, mpfr_exp_t);
__MPFR_DECLSPEC void mpfr_reldiff (mpfr_ptr, mpfr_srcptr, mpfr_srcptr,
                                   mpfr_rnd_t);
```

```
__MPFR_DECLSPEC int mpfr_eq (mpfr_srcptr, mpfr_srcptr, unsigned long);
__MPFR_DECLSPEC int mpfr_sgn (mpfr_srcptr);

__MPFR_DECLSPEC int mpfr_mul_2exp (mpfr_ptr, mpfr_srcptr, unsigned long,
                                   mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_div_2exp (mpfr_ptr, mpfr_srcptr, unsigned long,
                                   mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_mul_2ui (mpfr_ptr, mpfr_srcptr, unsigned long,
                                  mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_div_2ui (mpfr_ptr, mpfr_srcptr, unsigned long,
                                  mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_mul_2si (mpfr_ptr, mpfr_srcptr, long, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_div_2si (mpfr_ptr, mpfr_srcptr, long, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_rint (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_roundeven (mpfr_ptr, mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_round (mpfr_ptr, mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_trunc (mpfr_ptr, mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_ceil (mpfr_ptr, mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_floor (mpfr_ptr, mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_rint_roundeven (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_rint_round (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_rint_trunc (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_rint_ceil (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_rint_floor (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_frac (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_modf (mpfr_ptr, mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_remquo (mpfr_ptr, long*, mpfr_srcptr, mpfr_srcptr,
                                 mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_remainder (mpfr_ptr, mpfr_srcptr, mpfr_srcptr,
                                    mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_fmod (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_fmodquo (mpfr_ptr, long*, mpfr_srcptr, mpfr_srcptr,
                                  mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_fits_ulong_p (mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_fits_slong_p (mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_fits_uint_p (mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_fits_sint_p (mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_fits_ushort_p (mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_fits_sshort_p (mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_fits_uintmax_p (mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_fits_intmax_p (mpfr_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC void mpfr_extract (mpz_ptr, mpfr_srcptr, unsigned int);
__MPFR_DECLSPEC void mpfr_swap (mpfr_ptr, mpfr_ptr);
__MPFR_DECLSPEC void mpfr_dump (mpfr_srcptr);

__MPFR_DECLSPEC int mpfr_nan_p (mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_inf_p (mpfr_srcptr);
```

```
__MPFR_DECLSPEC int mpfr_number_p (mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_integer_p (mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_zero_p (mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_regular_p (mpfr_srcptr);

__MPFR_DECLSPEC int mpfr_greater_p (mpfr_srcptr, mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_greaterequal_p (mpfr_srcptr, mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_less_p (mpfr_srcptr, mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_lessequal_p (mpfr_srcptr, mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_lessgreater_p (mpfr_srcptr, mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_equal_p (mpfr_srcptr, mpfr_srcptr);
__MPFR_DECLSPEC int mpfr_unordered_p (mpfr_srcptr, mpfr_srcptr);

__MPFR_DECLSPEC int mpfr_atanh (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_acosh (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_asinh (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_cosh (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_sinh (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_tanh (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_sinh_cosh (mpfr_ptr, mpfr_ptr, mpfr_srcptr,
                                    mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_sech (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_csch (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_coth (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_acos (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_asin (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_atan (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_sin (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_sin_cos (mpfr_ptr, mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_cos (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_tan (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_atan2 (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_sec (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_csc (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_cot (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_hypot (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_erf (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_erfc (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_cbrt (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
#ifndef _MPFR_NO_DEPRECATED_ROOT /* for the test of this function */
MPFR_DEPRECATED
#endif
__MPFR_DECLSPEC int mpfr_root (mpfr_ptr, mpfr_srcptr, unsigned long,
                               mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_rootn_ui (mpfr_ptr, mpfr_srcptr, unsigned long,
                                   mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_gamma (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
```

```
__MPFR_DECLSPEC int mpfr_gamma_inc (mpfr_ptr, mpfr_srcptr, mpfr_srcptr,
                                    mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_beta (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_lngamma (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_lgamma (mpfr_ptr, int *, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_digamma (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_zeta (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_zeta_ui (mpfr_ptr, unsigned long, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_fac_ui (mpfr_ptr, unsigned long, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_j0 (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_j1 (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_jn (mpfr_ptr, long, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_y0 (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_y1 (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_yn (mpfr_ptr, long, mpfr_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_ai (mpfr_ptr, mpfr_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_min (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_max (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_dim (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_mul_z (mpfr_ptr, mpfr_srcptr, mpz_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_div_z (mpfr_ptr, mpfr_srcptr, mpz_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_add_z (mpfr_ptr, mpfr_srcptr, mpz_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_sub_z (mpfr_ptr, mpfr_srcptr, mpz_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_z_sub (mpfr_ptr, mpz_srcptr, mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_cmp_z (mpfr_srcptr, mpz_srcptr);

__MPFR_DECLSPEC int mpfr_fma (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_srcptr,
                              mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_fms (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_srcptr,
                              mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_fmma (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_srcptr,
                               mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_fmms (mpfr_ptr, mpfr_srcptr, mpfr_srcptr, mpfr_srcptr,
                               mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_sum (mpfr_ptr, const mpfr_ptr *, unsigned long,
                              mpfr_rnd_t);

__MPFR_DECLSPEC void mpfr_free_cache (void);
__MPFR_DECLSPEC void mpfr_free_cache2 (mpfr_free_cache_t);
__MPFR_DECLSPEC void mpfr_free_pool (void);
__MPFR_DECLSPEC int mpfr_mp_memory_cleanup (void);

__MPFR_DECLSPEC int mpfr_subnormalize (mpfr_ptr, int, mpfr_rnd_t);

__MPFR_DECLSPEC int mpfr_strtofr (mpfr_ptr, const char *, char **, int,
                                  mpfr_rnd_t);
```

```
__MPFR_DECLSPEC void mpfr_round_nearest_away_begin (mpfr_t);
__MPFR_DECLSPEC int mpfr_round_nearest_away_end (mpfr_t, int);

__MPFR_DECLSPEC size_t mpfr_custom_get_size (mpfr_prec_t);
__MPFR_DECLSPEC void mpfr_custom_init (void *, mpfr_prec_t);
__MPFR_DECLSPEC void * mpfr_custom_get_significand (mpfr_srcptr);
__MPFR_DECLSPEC mpfr_exp_t mpfr_custom_get_exp (mpfr_srcptr);
__MPFR_DECLSPEC void mpfr_custom_move (mpfr_ptr, void *);
__MPFR_DECLSPEC void mpfr_custom_init_set (mpfr_ptr, int, mpfr_exp_t,
                                           mpfr_prec_t, void *);
__MPFR_DECLSPEC int    mpfr_custom_get_kind (mpfr_srcptr);

#if defined (__cplusplus)
}
#endif

/* Define MPFR_USE_EXTENSION to avoid "gcc -pedantic" warnings. */
#ifndef MPFR_EXTENSION
# if defined(MPFR_USE_EXTENSION)
#  define MPFR_EXTENSION __extension__
# else
#  define MPFR_EXTENSION
# endif
#endif

/* Warning! This macro doesn't work with K&R C (e.g., compare the "gcc -E"
   output with and without -traditional) and shouldn't be used internally.
   For public use only, but see the MPFR manual. */
#define MPFR_DECL_INIT(_x, _p)                                          \
  MPFR_EXTENSION mp_limb_t __gmpfr_local_tab_##_x[((_p)-1)/GMP_NUMB_BITS+1]; \
  MPFR_EXTENSION mpfr_t _x = {{(_p),1,__MPFR_EXP_NAN,__gmpfr_local_tab_##_x}}

#if MPFR_USE_C99_FEATURE
/* C99 & C11 version: functions with multiple inputs supported */
#define mpfr_round_nearest_away(func, rop, ...)                         \
  (mpfr_round_nearest_away_begin(rop),                                  \
   mpfr_round_nearest_away_end((rop), func((rop), __VA_ARGS__, MPFR_RNDN)))
#else
/* C89 version: function with one input supported */
#define mpfr_round_nearest_away(func, rop, op)                          \
  (mpfr_round_nearest_away_begin(rop),                                  \
   mpfr_round_nearest_away_end((rop), func((rop), (op), MPFR_RNDN)))
#endif

/* Fast access macros to replace function interface.
   If the USER don't want to use the macro interface, let him make happy
   even if it produces faster and smaller code. */
#ifndef MPFR_USE_NO_MACRO

/* Inlining theses functions is both faster and smaller */
```

```
#define mpfr_nan_p(_x)      ((_x)->_mpfr_exp == __MPFR_EXP_NAN)
#define mpfr_inf_p(_x)      ((_x)->_mpfr_exp == __MPFR_EXP_INF)
#define mpfr_zero_p(_x)     ((_x)->_mpfr_exp == __MPFR_EXP_ZERO)
#define mpfr_regular_p(_x)  ((_x)->_mpfr_exp >  __MPFR_EXP_INF)
#define mpfr_sgn(_x)                                              \
  ((_x)->_mpfr_exp < __MPFR_EXP_INF ?                             \
   (mpfr_nan_p (_x) ? mpfr_set_erangeflag () : (mpfr_void) 0), 0 : \
   MPFR_SIGN (_x))

/* Prevent them from using as lvalues */
#define MPFR_VALUE_OF(x)  (0 ? (x) : (x))
#define mpfr_get_prec(_x) MPFR_VALUE_OF((_x)->_mpfr_prec)
#define mpfr_get_exp(_x)  MPFR_VALUE_OF((_x)->_mpfr_exp)
/* Note 1: If need be, the MPFR_VALUE_OF can be used for other expressions
   (of any type). Thanks to Wojtek Lerch and Tim Rentsch for the idea.
   Note 2: Defining mpfr_get_exp() as a macro has the effect to disable
   the check that the argument is a pure FP number (done in the function);
   this increases the risk of undetected error and makes debugging more
   complex. Is it really worth in practice? (Potential FIXME) */

#define mpfr_round(a,b) mpfr_rint((a), (b), MPFR_RNDNA)
#define mpfr_trunc(a,b) mpfr_rint((a), (b), MPFR_RNDZ)
#define mpfr_ceil(a,b)  mpfr_rint((a), (b), MPFR_RNDU)
#define mpfr_floor(a,b) mpfr_rint((a), (b), MPFR_RNDD)

#define mpfr_cmp_ui(b,i) mpfr_cmp_ui_2exp((b),(i),0)
#define mpfr_cmp_si(b,i) mpfr_cmp_si_2exp((b),(i),0)
#define mpfr_set(a,b,r)  mpfr_set4(a,b,r,MPFR_SIGN(b))
#define mpfr_abs(a,b,r)  mpfr_set4(a,b,r,1)
#define mpfr_copysign(a,b,c,r) mpfr_set4(a,b,r,MPFR_SIGN(c))
#define mpfr_setsign(a,b,s,r) mpfr_set4(a,b,r,(s) ? -1 : 1)
#define mpfr_signbit(x)  (MPFR_SIGN(x) < 0)
#define mpfr_cmp(b, c)   mpfr_cmp3(b, c, 1)
#define mpfr_mul_2exp(y,x,n,r) mpfr_mul_2ui((y),(x),(n),(r))
#define mpfr_div_2exp(y,x,n,r) mpfr_div_2ui((y),(x),(n),(r))


/* When using GCC, optimize certain common comparisons and affectations.
   + Remove some Intel C/C++ (ICC) versions since they now define __GNUC__
     but produce a huge number of warnings if you use this code.
     VL: I couldn't reproduce a single warning when enabling these macros
     with icc 10.1 20080212 on Itanium. But with this version, the obsolete
     __ICC macro isn't defined (__INTEL_COMPILER is, though), so that these
     macros are enabled anyway. Checking with other ICC versions is needed.
     For now, !defined(__ICC) seems to be the right test. Possibly detect
     whether warnings are produced or not with a configure test.
   + Remove C++ too, since it complains too much. */
/* Added casts to improve robustness in case of undefined behavior and
   compiler extensions based on UB (in particular -fwrapv). MPFR doesn't
   use such extensions, but these macros will be used by 3rd-party code,
```

```
   where such extensions may be required.
   Moreover casts to unsigned long have been added to avoid warnings in
   programs that use MPFR and are compiled with -Wconversion; such casts
   are OK since if X is a constant expression, then (unsigned long) X is
   also a constant expression, so that the optimizations still work. The
   warnings are probably related to the following two bugs:
     https://gcc.gnu.org/bugzilla/show_bug.cgi?id=4210
     https://gcc.gnu.org/bugzilla/show_bug.cgi?id=38470 (possibly a variant)
   and the casts could be removed once these bugs are fixed.
   Casts shouldn't be used on the generic calls (to the ..._2exp functions),
   where implicit conversions are performed. Indeed, having at least one
   implicit conversion in the macro allows the compiler to emit diagnostics
   when normally expected, for instance in the following call:
     mpfr_set_ui (x, "foo", MPFR_RNDN);
   If this is not possible (for future macros), one of the tricks described
   on http://groups.google.com/group/comp.std.c/msg/e92abd24bf9eaf7b could
   be used. */
#if defined (__GNUC__) && !defined(__ICC) && !defined(__cplusplus)
#if (__GNUC__ >= 2)

#undef mpfr_cmp_ui
/* We use the fact that mpfr_sgn on NaN sets the erange flag and returns 0.
   But warning! mpfr_sgn is specified as a macro in the API, thus the macro
   mustn't be used if side effects are possible, like here. */
#define mpfr_cmp_ui(_f,_u)                                      \
  (__builtin_constant_p (_u) && (mpfr_ulong) (_u) == 0 ?        \
   (mpfr_sgn) (_f) :                                            \
   mpfr_cmp_ui_2exp ((_f), (_u), 0))

#undef mpfr_cmp_si
#define mpfr_cmp_si(_f,_s)                                      \
  (__builtin_constant_p (_s) && (mpfr_long) (_s) >= 0 ?         \
   mpfr_cmp_ui ((_f), (mpfr_ulong) (mpfr_long) (_s)) :          \
   mpfr_cmp_si_2exp ((_f), (_s), 0))

#if __GNUC__ > 2 || __GNUC_MINOR__ >= 95
#undef mpfr_set_ui
#define mpfr_set_ui(_f,_u,_r)                                   \
  (__builtin_constant_p (_u) && (mpfr_ulong) (_u) == 0 ?        \
   __extension__ ({                                             \
       mpfr_ptr _p = (_f);                                      \
       _p->_mpfr_sign = 1;                                      \
       _p->_mpfr_exp = __MPFR_EXP_ZERO;                         \
       (mpfr_void) (_r); 0; }) :                                \
   mpfr_set_ui_2exp ((_f), (_u), 0, (_r)))
#endif

#undef mpfr_set_si
#define mpfr_set_si(_f,_s,_r)                                   \
  (__builtin_constant_p (_s) && (mpfr_long) (_s) >= 0 ?         \
```

```
   mpfr_set_ui ((_f), (mpfr_ulong) (mpfr_long) (_s), (_r)) :    \
   mpfr_set_si_2exp ((_f), (_s), 0, (_r)))

#if __GNUC__ > 3 || (__GNUC__ == 3 && __GNUC_MINOR__ >= 4)
/* If the source is a constant number that is a power of 2,
   optimize the call */
#undef mpfr_mul_ui
#define mpfr_mul_ui(_f, _g, _u,_r)                              \
  (__builtin_constant_p (_u) && (mpfr_ulong) (_u) >= 1 &&       \
   ((mpfr_ulong) (_u) & ((mpfr_ulong) (_u) - 1)) == 0 ?         \
   mpfr_mul_2si((_f), (_g), __builtin_ctzl (_u), (_r)) :        \
   mpfr_mul_ui ((_f), (_g), (_u), (_r)))
#undef mpfr_div_ui
#define mpfr_div_ui(_f, _g, _u,_r)                              \
  (__builtin_constant_p (_u) && (mpfr_ulong) (_u) >= 1 &&       \
   ((mpfr_ulong) (_u) & ((mpfr_ulong) (_u) - 1)) == 0 ?         \
   mpfr_mul_2si((_f), (_g), - __builtin_ctzl (_u), (_r)) :      \
   mpfr_div_ui ((_f), (_g), (_u), (_r)))
#endif

/* If the source is a constant number that is non-negative,
   optimize the call */
#undef mpfr_mul_si
#define mpfr_mul_si(_f, _g, _s,_r)                                      \
  (__builtin_constant_p (_s) && (mpfr_long) (_s) >= 0 ?                 \
   mpfr_mul_ui ((_f), (_g), (mpfr_ulong) (mpfr_long) (_s), (_r)) :      \
   mpfr_mul_si ((_f), (_g), (_s), (_r)))
#undef mpfr_div_si
#define mpfr_div_si(_f, _g, _s,_r)                                      \
  (__builtin_constant_p (_s) && (mpfr_long) (_s) >= 0 ?                 \
   mpfr_div_ui ((_f), (_g), (mpfr_ulong) (mpfr_long) (_s), (_r)) :      \
   mpfr_div_si ((_f), (_g), (_s), (_r)))

#endif
#endif

/* Macro version of mpfr_stack interface for fast access */
#define mpfr_custom_get_size(p) ((mpfr_size_t)                          \
       (((p)+GMP_NUMB_BITS-1)/GMP_NUMB_BITS*sizeof (mp_limb_t)))
#define mpfr_custom_init(m,p) do {} while (0)
#define mpfr_custom_get_significand(x) ((mpfr_void*)((x)->_mpfr_d))
#define mpfr_custom_get_exp(x) ((x)->_mpfr_exp)
#define mpfr_custom_move(x,m) do { ((x)->_mpfr_d = (mp_limb_t*)(m)); } while (0)
#define mpfr_custom_init_set(x,k,e,p,m) do {                    \
  mpfr_ptr _x = (x);                                            \
  mpfr_exp_t _e;                                                \
  mpfr_kind_t _t;                                               \
  mpfr_int _s, _k;                                              \
  _k = (k);                                                     \
  if (_k >= 0)  {                                               \
```

```
    _t = (mpfr_kind_t) _k;                                      \
    _s = 1;                                                     \
  } else {                                                      \
    _t = (mpfr_kind_t) - _k;                                    \
    _s = -1;                                                    \
  }                                                             \
  _e = _t == MPFR_REGULAR_KIND ? (e) :                          \
    _t == MPFR_NAN_KIND ? __MPFR_EXP_NAN :                      \
    _t == MPFR_INF_KIND ? __MPFR_EXP_INF : __MPFR_EXP_ZERO;     \
  _x->_mpfr_prec = (p);                                         \
  _x->_mpfr_sign = _s;                                          \
  _x->_mpfr_exp  = _e;                                          \
  _x->_mpfr_d    = (mp_limb_t*) (m);                            \
 } while (0)
#define mpfr_custom_get_kind(x)                                         \
  ( (x)->_mpfr_exp >  __MPFR_EXP_INF ?                                  \
    (mpfr_int) MPFR_REGULAR_KIND * MPFR_SIGN (x)                        \
  : (x)->_mpfr_exp == __MPFR_EXP_INF ?                                  \
    (mpfr_int) MPFR_INF_KIND * MPFR_SIGN (x)                            \
  : (x)->_mpfr_exp == __MPFR_EXP_NAN ? (mpfr_int) MPFR_NAN_KIND         \
  : (mpfr_int) MPFR_ZERO_KIND * MPFR_SIGN (x) )


#endif /* MPFR_USE_NO_MACRO */

/* Theses are defined to be macros */
#define mpfr_init_set_si(x, i, rnd) \
 ( mpfr_init(x), mpfr_set_si((x), (i), (rnd)) )
#define mpfr_init_set_ui(x, i, rnd) \
 ( mpfr_init(x), mpfr_set_ui((x), (i), (rnd)) )
#define mpfr_init_set_d(x, d, rnd) \
 ( mpfr_init(x), mpfr_set_d((x), (d), (rnd)) )
#define mpfr_init_set_ld(x, d, rnd) \
 ( mpfr_init(x), mpfr_set_ld((x), (d), (rnd)) )
#define mpfr_init_set_z(x, i, rnd) \
 ( mpfr_init(x), mpfr_set_z((x), (i), (rnd)) )
#ifndef MPFR_USE_MINI_GMP
#define mpfr_init_set_q(x, i, rnd) \
 ( mpfr_init(x), mpfr_set_q((x), (i), (rnd)) )
#define mpfr_init_set_f(x, y, rnd) \
 ( mpfr_init(x), mpfr_set_f((x), (y), (rnd)) )
#endif
#define mpfr_init_set(x, y, rnd) \
 ( mpfr_init(x), mpfr_set((x), (y), (rnd)) )

/* Compatibility layer -- obsolete functions and macros */
/* Note: it is not possible to output warnings, unless one defines
 * a deprecated variable and uses it, e.g.
 *   MPFR_DEPRECATED extern int mpfr_deprecated_feature;
 *   #define MPFR_EMIN_MIN ((void)mpfr_deprecated_feature,mpfr_get_emin_min())
```

```
 * (the cast to void avoids a warning because the left-hand operand
 * has no effect).
 */
#define mpfr_cmp_abs mpfr_cmpabs
#define mpfr_round_prec(x,r,p) mpfr_prec_round(x,p,r)
#define __gmp_default_rounding_mode (mpfr_get_default_rounding_mode())
#define __mpfr_emin (mpfr_get_emin())
#define __mpfr_emax (mpfr_get_emax())
#define __mpfr_default_fp_bit_precision (mpfr_get_default_fp_bit_precision())
#define MPFR_EMIN_MIN mpfr_get_emin_min()
#define MPFR_EMIN_MAX mpfr_get_emin_max()
#define MPFR_EMAX_MIN mpfr_get_emax_min()
#define MPFR_EMAX_MAX mpfr_get_emax_max()
#define mpfr_version (mpfr_get_version())
#ifndef mpz_set_fr
# define mpz_set_fr mpfr_get_z
#endif
#define mpfr_get_z_exp mpfr_get_z_2exp
#define mpfr_custom_get_mantissa mpfr_custom_get_significand

#endif /* __MPFR_H */


/* Check if <stdint.h> / <inttypes.h> is included or if the user
   explicitly wants intmax_t. Automatical detection is done by
   checking:
     - INTMAX_C and UINTMAX_C, but not if the compiler is a C++ one
       (as suggested by Patrick Pelissier) because the test does not
       work well in this case. See:
         https://sympa.inria.fr/sympa/arc/mpfr/2010-02/msg00025.html
       We do not check INTMAX_MAX and UINTMAX_MAX because under Solaris,
       these macros are always defined by <limits.h> (i.e. even when
       <stdint.h> and <inttypes.h> are not included).
     - _STDINT_H (defined by the glibc), _STDINT_H_ (defined under
       Mac OS X) and _STDINT (defined under MS Visual Studio), but
       this test may not work with all implementations.
       Portable software should not rely on these tests.
*/
#if (defined (INTMAX_C) && defined (UINTMAX_C) && !defined(__cplusplus)) || \
  defined (MPFR_USE_INTMAX_T) || \
  defined (_STDINT_H) || defined (_STDINT_H_) || defined (_STDINT) || \
  defined (_SYS_STDINT_H_) /* needed for FreeBSD */
# ifndef _MPFR_H_HAVE_INTMAX_T
# define _MPFR_H_HAVE_INTMAX_T 1

#if defined (__cplusplus)
extern "C" {
#endif

#define mpfr_set_sj __gmpfr_set_sj
```

```
#define mpfr_set_sj_2exp __gmpfr_set_sj_2exp
#define mpfr_set_uj __gmpfr_set_uj
#define mpfr_set_uj_2exp __gmpfr_set_uj_2exp
#define mpfr_get_sj __gmpfr_mpfr_get_sj
#define mpfr_get_uj __gmpfr_mpfr_get_uj
__MPFR_DECLSPEC int mpfr_set_sj (mpfr_t, intmax_t, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_set_sj_2exp (mpfr_t, intmax_t, intmax_t, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_set_uj (mpfr_t, uintmax_t, mpfr_rnd_t);
__MPFR_DECLSPEC int mpfr_set_uj_2exp (mpfr_t, uintmax_t, intmax_t, mpfr_rnd_t);
__MPFR_DECLSPEC intmax_t mpfr_get_sj (mpfr_srcptr, mpfr_rnd_t);
__MPFR_DECLSPEC uintmax_t mpfr_get_uj (mpfr_srcptr, mpfr_rnd_t);

#if defined (__cplusplus)
}
#endif

# endif /* _MPFR_H_HAVE_INTMAX_T */
#endif


/* Check if <stdio.h> has been included or if the user wants FILE */
#if defined (_GMP_H_HAVE_FILE) || defined (MPFR_USE_FILE)
# ifndef _MPFR_H_HAVE_FILE
# define _MPFR_H_HAVE_FILE 1

#if defined (__cplusplus)
extern "C" {
#endif

#define mpfr_inp_str __gmpfr_inp_str
#define mpfr_out_str __gmpfr_out_str
__MPFR_DECLSPEC size_t mpfr_inp_str (mpfr_ptr, FILE*, int, mpfr_rnd_t);
__MPFR_DECLSPEC size_t mpfr_out_str (FILE*, int, size_t, mpfr_srcptr,
                                     mpfr_rnd_t);
#ifndef MPFR_USE_MINI_GMP
#define mpfr_fprintf __gmpfr_fprintf
__MPFR_DECLSPEC int mpfr_fprintf (FILE*, const char*, ...);
#endif
#define mpfr_fpif_export __gmpfr_fpif_export
#define mpfr_fpif_import __gmpfr_fpif_import
__MPFR_DECLSPEC int mpfr_fpif_export (FILE*, mpfr_ptr);
__MPFR_DECLSPEC int mpfr_fpif_import (mpfr_ptr, FILE*);

#if defined (__cplusplus)
}
#endif

# endif /* _MPFR_H_HAVE_FILE */
#endif
```

```
/* check if <stdarg.h> has been included or if the user wants va_list */
#if defined (_GMP_H_HAVE_VA_LIST) || defined (MPFR_USE_VA_LIST)
# ifndef _MPFR_H_HAVE_VA_LIST
# define _MPFR_H_HAVE_VA_LIST 1

#if defined (__cplusplus)
extern "C" {
#endif

#define mpfr_vprintf __gmpfr_vprintf
#define mpfr_vasprintf __gmpfr_vasprintf
#define mpfr_vsprintf __gmpfr_vsprintf
#define mpfr_vsnprintf __gmpfr_vsnprintf
__MPFR_DECLSPEC int mpfr_vprintf (const char*, va_list);
__MPFR_DECLSPEC int mpfr_vasprintf (char**, const char*, va_list);
__MPFR_DECLSPEC int mpfr_vsprintf (char*, const char*, va_list);
__MPFR_DECLSPEC int mpfr_vsnprintf (char*, size_t, const char*, va_list);

#if defined (__cplusplus)
}
#endif

# endif /* _MPFR_H_HAVE_VA_LIST */
#endif


/* check if <stdarg.h> has been included and if FILE is available
   (see above) */
#if defined (_MPFR_H_HAVE_VA_LIST) && defined (_MPFR_H_HAVE_FILE)
# ifndef _MPFR_H_HAVE_VA_LIST_FILE
# define _MPFR_H_HAVE_VA_LIST_FILE 1

#if defined (__cplusplus)
extern "C" {
#endif

#define mpfr_vfprintf __gmpfr_vfprintf
__MPFR_DECLSPEC int mpfr_vfprintf (FILE*, const char*, va_list);

#if defined (__cplusplus)
}
#endif

# endif /* _MPFR_H_HAVE_VA_LIST_FILE */
#endif
```

49 stdio.h

```
/* Define ISO C stdio on top of C++ iostreams.
   Copyright (C) 1991-2018 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   The GNU C Library is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
   Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public
   License along with the GNU C Library; if not, see
   <http://www.gnu.org/licenses/>.  */

/*
 * ISO C99 Standard: 7.19 Input/output <stdio.h>
 */

#ifndef _STDIO_H
#define _STDIO_H 1

#define __GLIBC_INTERNAL_STARTING_HEADER_IMPLEMENTATION
#include <bits/libc-header-start.h>

__BEGIN_DECLS

#define __need_size_t
#define __need_NULL
#include <stddef.h>

#include <bits/types.h>
#include <bits/types/__FILE.h>
#include <bits/types/FILE.h>

#define _STDIO_USES_IOSTREAM

#include <bits/libio.h>

#if defined __USE_XOPEN || defined __USE_XOPEN2K8
# ifdef __GNUC__
#  ifndef _VA_LIST_DEFINED
typedef _G_va_list va_list;
#   define _VA_LIST_DEFINED
```

```
#  endif
# else
#  include <stdarg.h>
# endif
#endif

#if defined __USE_UNIX98 || defined __USE_XOPEN2K
# ifndef __off_t_defined
# ifndef __USE_FILE_OFFSET64
typedef __off_t off_t;
# else
typedef __off64_t off_t;
# endif
# define __off_t_defined
# endif
# if defined __USE_LARGEFILE64 && !defined __off64_t_defined
typedef __off64_t off64_t;
# define __off64_t_defined
# endif
#endif

#ifdef __USE_XOPEN2K8
# ifndef __ssize_t_defined
typedef __ssize_t ssize_t;
# define __ssize_t_defined
# endif
#endif

/* The type of the second argument to ‘fgetpos’ and ‘fsetpos’.  */
#ifndef __USE_FILE_OFFSET64
typedef _G_fpos_t fpos_t;
#else
typedef _G_fpos64_t fpos_t;
#endif
#ifdef __USE_LARGEFILE64
typedef _G_fpos64_t fpos64_t;
#endif

/* The possibilities for the third argument to ‘setvbuf’.  */
#define _IOFBF 0 /* Fully buffered.  */
#define _IOLBF 1 /* Line buffered.  */
#define _IONBF 2 /* No buffering.  */


/* Default buffer size.  */
#ifndef BUFSIZ
# define BUFSIZ _IO_BUFSIZ
#endif
```

```
/* End of file character.
   Some things throughout the library rely on this being -1.  */
#ifndef EOF
# define EOF (-1)
#endif


/* The possibilities for the third argument to 'fseek'.
   These values should not be changed.  */
#define SEEK_SET 0 /* Seek from beginning of file.  */
#define SEEK_CUR 1 /* Seek from current position.  */
#define SEEK_END 2 /* Seek from end of file.  */
#ifdef __USE_GNU
# define SEEK_DATA 3 /* Seek to next data.  */
# define SEEK_HOLE 4 /* Seek to next hole.  */
#endif


#if defined __USE_MISC || defined __USE_XOPEN
/* Default path prefix for 'tempnam' and 'tmpnam'.  */
# define P_tmpdir "/tmp"
#endif


/* Get the values:
   L_tmpnam How long an array of chars must be to be passed to 'tmpnam'.
   TMP_MAX The minimum number of unique filenames generated by tmpnam
(and tempnam when it uses tmpnam's name space),
or tempnam (the two are separate).
   L_ctermid How long an array to pass to 'ctermid'.
   L_cuserid How long an array to pass to 'cuserid'.
   FOPEN_MAX Minimum number of files that can be open at once.
   FILENAME_MAX Maximum length of a filename.  */
#include <bits/stdio_lim.h>


/* Standard streams.  */
extern struct _IO_FILE *stdin; /* Standard input stream.  */
extern struct _IO_FILE *stdout; /* Standard output stream.  */
extern struct _IO_FILE *stderr; /* Standard error output stream.  */
/* C89/C99 say they're macros.  Make them happy.  */
#define stdin stdin
#define stdout stdout
#define stderr stderr

/* Remove file FILENAME.  */
extern int remove (const char *__filename) __THROW;
/* Rename file OLD to NEW.  */
extern int rename (const char *__old, const char *__new) __THROW;
```

```
#ifdef __USE_ATFILE
/* Rename file OLD relative to OLDFD to NEW relative to NEWFD.  */
extern int renameat (int __oldfd, const char *__old, int __newfd,
     const char *__new) __THROW;
#endif

/* Create a temporary file and open it read/write.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
#ifndef __USE_FILE_OFFSET64
extern FILE *tmpfile (void) __wur;
#else
# ifdef __REDIRECT
extern FILE *__REDIRECT (tmpfile, (void), tmpfile64) __wur;
# else
#  define tmpfile tmpfile64
# endif
#endif

#ifdef __USE_LARGEFILE64
extern FILE *tmpfile64 (void) __wur;
#endif

/* Generate a temporary filename.  */
extern char *tmpnam (char *__s) __THROW __wur;

#ifdef __USE_MISC
/* This is the reentrant variant of `tmpnam'.  The only difference is
   that it does not allow S to be NULL.  */
extern char *tmpnam_r (char *__s) __THROW __wur;
#endif


#if defined __USE_MISC || defined __USE_XOPEN
/* Generate a unique temporary filename using up to five characters of PFX
   if it is not NULL.  The directory to put this file in is searched for
   as follows: First the environment variable "TMPDIR" is checked.
   If it contains the name of a writable directory, that directory is used.
   If not and if DIR is not NULL, that value is checked.  If that fails,
   P_tmpdir is tried and finally "/tmp".  The storage for the filename
   is allocated by `malloc'.  */
extern char *tempnam (const char *__dir, const char *__pfx)
     __THROW __attribute_malloc__ __wur;
#endif


/* Close STREAM.

   This function is a possible cancellation point and therefore not
```

```
   marked with __THROW.  */
extern int fclose (FILE *__stream);
/* Flush STREAM, or all streams if STREAM is NULL.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fflush (FILE *__stream);

#ifdef __USE_MISC
/* Faster versions when locking is not required.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern int fflush_unlocked (FILE *__stream);
#endif

#ifdef __USE_GNU
/* Close all streams.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern int fcloseall (void);
#endif


#ifndef __USE_FILE_OFFSET64
/* Open a file and create a new stream for it.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern FILE *fopen (const char *__restrict __filename,
    const char *__restrict __modes) __wur;
/* Open a file, replacing an existing stream with it.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern FILE *freopen (const char *__restrict __filename,
      const char *__restrict __modes,
      FILE *__restrict __stream) __wur;
#else
# ifdef __REDIRECT
extern FILE *__REDIRECT (fopen, (const char *__restrict __filename,
 const char *__restrict __modes), fopen64)
  __wur;
extern FILE *__REDIRECT (freopen, (const char *__restrict __filename,
   const char *__restrict __modes,
```

```
   FILE *__restrict __stream), freopen64)
  __wur;
# else
#  define fopen fopen64
#  define freopen freopen64
# endif
#endif
#ifdef __USE_LARGEFILE64
extern FILE *fopen64 (const char *__restrict __filename,
      const char *__restrict __modes) __wur;
extern FILE *freopen64 (const char *__restrict __filename,
const char *__restrict __modes,
FILE *__restrict __stream) __wur;
#endif

#ifdef __USE_POSIX
/* Create a new stream that refers to an existing system file descriptor.  */
extern FILE *fdopen (int __fd, const char *__modes) __THROW __wur;
#endif

#ifdef __USE_GNU
/* Create a new stream that refers to the given magic cookie,
   and uses the given functions for input and output.  */
extern FILE *fopencookie (void *__restrict __magic_cookie,
  const char *__restrict __modes,
  _IO_cookie_io_functions_t __io_funcs) __THROW __wur;
#endif

#if defined __USE_XOPEN2K8 || __GLIBC_USE (LIB_EXT2)
/* Create a new stream that refers to a memory buffer.  */
extern FILE *fmemopen (void *__s, size_t __len, const char *__modes)
  __THROW __wur;

/* Open a stream that writes into a malloc'd buffer that is expanded as
   necessary.  *BUFLOC and *SIZELOC are updated with the buffer's location
   and the number of characters written on fflush or fclose.  */
extern FILE *open_memstream (char **__bufloc, size_t *__sizeloc) __THROW __wur;
#endif


/* If BUF is NULL, make STREAM unbuffered.
   Else make it use buffer BUF, of size BUFSIZ.  */
extern void setbuf (FILE *__restrict __stream, char *__restrict __buf) __THROW;
/* Make STREAM use buffering mode MODE.
   If BUF is not NULL, use N bytes of it for buffering;
   else allocate an internal buffer N bytes long.  */
extern int setvbuf (FILE *__restrict __stream, char *__restrict __buf,
    int __modes, size_t __n) __THROW;

#ifdef __USE_MISC
```

```
/* If BUF is NULL, make STREAM unbuffered.
   Else make it use SIZE bytes of BUF for buffering.  */
extern void setbuffer (FILE *__restrict __stream, char *__restrict __buf,
       size_t __size) __THROW;

/* Make STREAM line-buffered.  */
extern void setlinebuf (FILE *__stream) __THROW;
#endif


/* Write formatted output to STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fprintf (FILE *__restrict __stream,
    const char *__restrict __format, ...);
/* Write formatted output to stdout.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int printf (const char *__restrict __format, ...);
/* Write formatted output to S.  */
extern int sprintf (char *__restrict __s,
    const char *__restrict __format, ...) __THROWNL;

/* Write formatted output to S from argument list ARG.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int vfprintf (FILE *__restrict __s, const char *__restrict __format,
     _G_va_list __arg);
/* Write formatted output to stdout from argument list ARG.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int vprintf (const char *__restrict __format, _G_va_list __arg);
/* Write formatted output to S from argument list ARG.  */
extern int vsprintf (char *__restrict __s, const char *__restrict __format,
     _G_va_list __arg) __THROWNL;

#if defined __USE_ISOC99 || defined __USE_UNIX98
/* Maximum chars of output to write in MAXLEN.  */
extern int snprintf (char *__restrict __s, size_t __maxlen,
     const char *__restrict __format, ...)
     __THROWNL __attribute__ ((__format__ (__printf__, 3, 4)));

extern int vsnprintf (char *__restrict __s, size_t __maxlen,
      const char *__restrict __format, _G_va_list __arg)
     __THROWNL __attribute__ ((__format__ (__printf__, 3, 0)));
#endif
```

```
#if __GLIBC_USE (LIB_EXT2)
/* Write formatted output to a string dynamically allocated with ‘malloc’.
   Store the address of the string in *PTR.  */
extern int vasprintf (char **__restrict __ptr, const char *__restrict __f,
      _G_va_list __arg)
     __THROWNL __attribute__ ((__format__ (__printf__, 2, 0))) __wur;
extern int __asprintf (char **__restrict __ptr,
       const char *__restrict __fmt, ...)
     __THROWNL __attribute__ ((__format__ (__printf__, 2, 3))) __wur;
extern int asprintf (char **__restrict __ptr,
     const char *__restrict __fmt, ...)
     __THROWNL __attribute__ ((__format__ (__printf__, 2, 3))) __wur;
#endif

#ifdef __USE_XOPEN2K8
/* Write formatted output to a file descriptor.  */
extern int vdprintf (int __fd, const char *__restrict __fmt,
     _G_va_list __arg)
     __attribute__ ((__format__ (__printf__, 2, 0)));
extern int dprintf (int __fd, const char *__restrict __fmt, ...)
     __attribute__ ((__format__ (__printf__, 2, 3)));
#endif


/* Read formatted input from STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fscanf (FILE *__restrict __stream,
   const char *__restrict __format, ...) __wur;
/* Read formatted input from stdin.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int scanf (const char *__restrict __format, ...) __wur;
/* Read formatted input from S.  */
extern int sscanf (const char *__restrict __s,
   const char *__restrict __format, ...) __THROW;

#if defined __USE_ISOC99 && !defined __USE_GNU \
    && (!defined __LDBL_COMPAT || !defined __REDIRECT) \
    && (defined __STRICT_ANSI__ || defined __USE_XOPEN2K)
# ifdef __REDIRECT
/* For strict ISO C99 or POSIX compliance disallow %as, %aS and %a[
   GNU extension which conflicts with valid %a followed by letter
   s, S or [.  */
extern int __REDIRECT (fscanf, (FILE *__restrict __stream,
const char *__restrict __format, ...),
       __isoc99_fscanf) __wur;
```

```
extern int __REDIRECT (scanf, (const char *__restrict __format, ...),
       __isoc99_scanf) __wur;
extern int __REDIRECT_NTH (sscanf, (const char *__restrict __s,
    const char *__restrict __format, ...),
   __isoc99_sscanf);
# else
extern int __isoc99_fscanf (FILE *__restrict __stream,
    const char *__restrict __format, ...) __wur;
extern int __isoc99_scanf (const char *__restrict __format, ...) __wur;
extern int __isoc99_sscanf (const char *__restrict __s,
    const char *__restrict __format, ...) __THROW;
#  define fscanf __isoc99_fscanf
#  define scanf __isoc99_scanf
#  define sscanf __isoc99_sscanf
# endif
#endif

#ifdef __USE_ISOC99
/* Read formatted input from S into argument list ARG.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int vfscanf (FILE *__restrict __s, const char *__restrict __format,
    _G_va_list __arg)
     __attribute__ ((__format__ (__scanf__, 2, 0))) __wur;

/* Read formatted input from stdin into argument list ARG.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int vscanf (const char *__restrict __format, _G_va_list __arg)
     __attribute__ ((__format__ (__scanf__, 1, 0))) __wur;

/* Read formatted input from S into argument list ARG.  */
extern int vsscanf (const char *__restrict __s,
    const char *__restrict __format, _G_va_list __arg)
     __THROW __attribute__ ((__format__ (__scanf__, 2, 0)));

# if !defined __USE_GNU \
     && (!defined __LDBL_COMPAT || !defined __REDIRECT) \
     && (defined __STRICT_ANSI__ || defined __USE_XOPEN2K)
#  ifdef __REDIRECT
/* For strict ISO C99 or POSIX compliance disallow %as, %aS and %a[
   GNU extension which conflicts with valid %a followed by letter
   s, S or [.  */
extern int __REDIRECT (vfscanf,
       (FILE *__restrict __s,
const char *__restrict __format, _G_va_list __arg),
       __isoc99_vfscanf)
     __attribute__ ((__format__ (__scanf__, 2, 0))) __wur;
```

```
extern int __REDIRECT (vscanf, (const char *__restrict __format,
_G_va_list __arg), __isoc99_vscanf)
     __attribute__ ((__format__ (__scanf__, 1, 0))) __wur;
extern int __REDIRECT_NTH (vsscanf,
   (const char *__restrict __s,
    const char *__restrict __format,
    _G_va_list __arg), __isoc99_vsscanf)
     __attribute__ ((__format__ (__scanf__, 2, 0)));
#  else
extern int __isoc99_vfscanf (FILE *__restrict __s,
     const char *__restrict __format,
     _G_va_list __arg) __wur;
extern int __isoc99_vscanf (const char *__restrict __format,
    _G_va_list __arg) __wur;
extern int __isoc99_vsscanf (const char *__restrict __s,
     const char *__restrict __format,
     _G_va_list __arg) __THROW;
#   define vfscanf __isoc99_vfscanf
#   define vscanf __isoc99_vscanf
#   define vsscanf __isoc99_vsscanf
#  endif
# endif
#endif /* Use ISO C9x.  */


/* Read a character from STREAM.

   These functions are possible cancellation points and therefore not
   marked with __THROW.  */
extern int fgetc (FILE *__stream);
extern int getc (FILE *__stream);

/* Read a character from stdin.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int getchar (void);

/* The C standard explicitly says this is a macro, so we always do the
   optimization for it.  */
#define getc(_fp) _IO_getc (_fp)

#ifdef __USE_POSIX199506
/* These are defined in POSIX.1:1996.

   These functions are possible cancellation points and therefore not
   marked with __THROW.  */
extern int getc_unlocked (FILE *__stream);
extern int getchar_unlocked (void);
#endif /* Use POSIX.  */
```

```
#ifdef __USE_MISC
/* Faster version when locking is not necessary.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern int fgetc_unlocked (FILE *__stream);
#endif /* Use MISC.  */


/* Write a character to STREAM.

   These functions are possible cancellation points and therefore not
   marked with __THROW.

   These functions is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fputc (int __c, FILE *__stream);
extern int putc (int __c, FILE *__stream);

/* Write a character to stdout.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int putchar (int __c);

/* The C standard explicitly says this can be a macro,
   so we always do the optimization for it.  */
#define putc(_ch, _fp) _IO_putc (_ch, _fp)

#ifdef __USE_MISC
/* Faster version when locking is not necessary.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern int fputc_unlocked (int __c, FILE *__stream);
#endif /* Use MISC.  */

#ifdef __USE_POSIX199506
/* These are defined in POSIX.1:1996.

   These functions are possible cancellation points and therefore not
   marked with __THROW.  */
extern int putc_unlocked (int __c, FILE *__stream);
extern int putchar_unlocked (int __c);
#endif /* Use POSIX.  */
```

```
#if defined __USE_MISC \
    || (defined __USE_XOPEN && !defined __USE_XOPEN2K)
/* Get a word (int) from STREAM.  */
extern int getw (FILE *__stream);

/* Write a word (int) to STREAM.  */
extern int putw (int __w, FILE *__stream);
#endif


/* Get a newline-terminated string of finite length from STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern char *fgets (char *__restrict __s, int __n, FILE *__restrict __stream)
     __wur;

#if __GLIBC_USE (DEPRECATED_GETS)
/* Get a newline-terminated string from stdin, removing the newline.

   This function is impossible to use safely.  It has been officially
   removed from ISO C11 and ISO C++14, and we have also removed it
   from the _GNU_SOURCE feature list.  It remains available when
   explicitly using an old ISO C, Unix, or POSIX standard.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern char *gets (char *__s) __wur __attribute_deprecated__;
#endif

#ifdef __USE_GNU
/* This function does the same as ‘fgets’ but does not lock the stream.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern char *fgets_unlocked (char *__restrict __s, int __n,
     FILE *__restrict __stream) __wur;
#endif


#if defined __USE_XOPEN2K8 || __GLIBC_USE (LIB_EXT2)
/* Read up to (and including) a DELIMITER from STREAM into *LINEPTR
   (and null-terminate it). *LINEPTR is a pointer returned from malloc (or
   NULL), pointing to *N characters of space.  It is realloc'd as
   necessary.  Returns the number of characters read (not including the
   null terminator), or -1 on error or EOF.
```

```
   These functions are not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation they are cancellation points and
   therefore not marked with __THROW.  */
extern _IO_ssize_t __getdelim (char **__restrict __lineptr,
       size_t *__restrict __n, int __delimiter,
       FILE *__restrict __stream) __wur;
extern _IO_ssize_t getdelim (char **__restrict __lineptr,
     size_t *__restrict __n, int __delimiter,
     FILE *__restrict __stream) __wur;

/* Like 'getdelim', but reads up to a newline.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern _IO_ssize_t getline (char **__restrict __lineptr,
    size_t *__restrict __n,
    FILE *__restrict __stream) __wur;
#endif


/* Write a string to STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fputs (const char *__restrict __s, FILE *__restrict __stream);

/* Write a string, followed by a newline, to stdout.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int puts (const char *__s);


/* Push a character back onto the input buffer of STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int ungetc (int __c, FILE *__stream);


/* Read chunks of generic data from STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern size_t fread (void *__restrict __ptr, size_t __size,
     size_t __n, FILE *__restrict __stream) __wur;
```

```
/* Write chunks of generic data to STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern size_t fwrite (const void *__restrict __ptr, size_t __size,
      size_t __n, FILE *__restrict __s);

#ifdef __USE_GNU
/* This function does the same as ‘fputs’ but does not lock the stream.

   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern int fputs_unlocked (const char *__restrict __s,
   FILE *__restrict __stream);
#endif

#ifdef __USE_MISC
/* Faster versions when locking is not necessary.

   These functions are not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation they are cancellation points and
   therefore not marked with __THROW.  */
extern size_t fread_unlocked (void *__restrict __ptr, size_t __size,
      size_t __n, FILE *__restrict __stream) __wur;
extern size_t fwrite_unlocked (const void *__restrict __ptr, size_t __size,
       size_t __n, FILE *__restrict __stream);
#endif


/* Seek to a certain position on STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fseek (FILE *__stream, long int __off, int __whence);
/* Return the current position of STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern long int ftell (FILE *__stream) __wur;
/* Rewind to the beginning of STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern void rewind (FILE *__stream);

/* The Single Unix Specification, Version 2, specifies an alternative,
   more adequate interface for the two functions above which deal with
```

```
   file offset.  'long int' is not the right type.  These definitions
   are originally defined in the Large File Support API.  */

#if defined __USE_LARGEFILE || defined __USE_XOPEN2K
# ifndef __USE_FILE_OFFSET64
/* Seek to a certain position on STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fseeko (FILE *__stream, __off_t __off, int __whence);
/* Return the current position of STREAM.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern __off_t ftello (FILE *__stream) __wur;
# else
#  ifdef __REDIRECT
extern int __REDIRECT (fseeko,
       (FILE *__stream, __off64_t __off, int __whence),
       fseeko64);
extern __off64_t __REDIRECT (ftello, (FILE *__stream), ftello64);
#  else
#   define fseeko fseeko64
#   define ftello ftello64
#  endif
# endif
#endif

#ifndef __USE_FILE_OFFSET64
/* Get STREAM's position.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fgetpos (FILE *__restrict __stream, fpos_t *__restrict __pos);
/* Set STREAM's position.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int fsetpos (FILE *__stream, const fpos_t *__pos);
#else
# ifdef __REDIRECT
extern int __REDIRECT (fgetpos, (FILE *__restrict __stream,
 fpos_t *__restrict __pos), fgetpos64);
extern int __REDIRECT (fsetpos,
       (FILE *__stream, const fpos_t *__pos), fsetpos64);
# else
#  define fgetpos fgetpos64
#  define fsetpos fsetpos64
# endif
#endif
```

```
#ifdef __USE_LARGEFILE64
extern int fseeko64 (FILE *__stream, __off64_t __off, int __whence);
extern __off64_t ftello64 (FILE *__stream) __wur;
extern int fgetpos64 (FILE *__restrict __stream, fpos64_t *__restrict __pos);
extern int fsetpos64 (FILE *__stream, const fpos64_t *__pos);
#endif

/* Clear the error and EOF indicators for STREAM.  */
extern void clearerr (FILE *__stream) __THROW;
/* Return the EOF indicator for STREAM.  */
extern int feof (FILE *__stream) __THROW __wur;
/* Return the error indicator for STREAM.  */
extern int ferror (FILE *__stream) __THROW __wur;

#ifdef __USE_MISC
/* Faster versions when locking is not required.  */
extern void clearerr_unlocked (FILE *__stream) __THROW;
extern int feof_unlocked (FILE *__stream) __THROW __wur;
extern int ferror_unlocked (FILE *__stream) __THROW __wur;
#endif


/* Print a message describing the meaning of the value of errno.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern void perror (const char *__s);

/* Provide the declarations for ‘sys_errlist’ and ‘sys_nerr’ if they
   are available on this system.  Even if available, these variables
   should not be used directly.  The ‘strerror’ function provides
   all the necessary functionality.  */
#include <bits/sys_errlist.h>


#ifdef __USE_POSIX
/* Return the system file descriptor for STREAM.  */
extern int fileno (FILE *__stream) __THROW __wur;
#endif /* Use POSIX.  */

#ifdef __USE_MISC
/* Faster version when locking is not required.  */
extern int fileno_unlocked (FILE *__stream) __THROW __wur;
#endif


#ifdef __USE_POSIX2
/* Create a new stream connected to a pipe running the given command.
```

```
   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern FILE *popen (const char *__command, const char *__modes) __wur;

/* Close a stream opened by popen and return the status of its child.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int pclose (FILE *__stream);
#endif


#ifdef __USE_POSIX
/* Return the name of the controlling terminal.  */
extern char *ctermid (char *__s) __THROW;
#endif /* Use POSIX.  */


#if (defined __USE_XOPEN && !defined __USE_XOPEN2K) || defined __USE_GNU
/* Return the name of the current user.  */
extern char *cuserid (char *__s);
#endif /* Use X/Open, but not issue 6.  */


#ifdef __USE_GNU
struct obstack; /* See <obstack.h>.  */

/* Write formatted output to an obstack.  */
extern int obstack_printf (struct obstack *__restrict __obstack,
   const char *__restrict __format, ...)
     __THROWNL __attribute__ ((__format__ (__printf__, 2, 3)));
extern int obstack_vprintf (struct obstack *__restrict __obstack,
    const char *__restrict __format,
    _G_va_list __args)
     __THROWNL __attribute__ ((__format__ (__printf__, 2, 0)));
#endif /* Use GNU.  */


#ifdef __USE_POSIX199506
/* These are defined in POSIX.1:1996.  */

/* Acquire ownership of STREAM.  */
extern void flockfile (FILE *__stream) __THROW;

/* Try to acquire ownership of STREAM but do not block if it is not
   possible.  */
extern int ftrylockfile (FILE *__stream) __THROW __wur;

/* Relinquish the ownership granted for STREAM.  */
extern void funlockfile (FILE *__stream) __THROW;
```

```
#endif /* POSIX */

#if defined __USE_XOPEN && !defined __USE_XOPEN2K && !defined __USE_GNU
/*  X/Open Issues 1-5 required getopt to be declared in this
   header.  It was removed in Issue 6.  GNU follows Issue 6.  */
# include <bits/getopt_posix.h>
#endif

/* If we are compiling with optimizing read this file.  It contains
   several optimizing inline functions and macros.  */
#ifdef __USE_EXTERN_INLINES
# include <bits/stdio.h>
#endif
#if __USE_FORTIFY_LEVEL > 0 && defined __fortify_function
# include <bits/stdio2.h>
#endif
#ifdef __LDBL_COMPAT
# include <bits/stdio-ldbl.h>
#endif

__END_DECLS

#endif /* <stdio.h> included.  */
```

50 stdlib.h

```
/* Copyright (C) 1991-2018 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   The GNU C Library is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
   Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public
   License along with the GNU C Library; if not, see
   <http://www.gnu.org/licenses/>.  */

/*
 * ISO C99 Standard: 7.20 General utilities <stdlib.h>
 */

#ifndef _STDLIB_H

#define __GLIBC_INTERNAL_STARTING_HEADER_IMPLEMENTATION
#include <bits/libc-header-start.h>

/* Get size_t, wchar_t and NULL from <stddef.h>.  */
#define __need_size_t
#define __need_wchar_t
#define __need_NULL
#include <stddef.h>

__BEGIN_DECLS

#define _STDLIB_H 1

#if (defined __USE_XOPEN || defined __USE_XOPEN2K8) && !defined _SYS_WAIT_H
/* XPG requires a few symbols from <sys/wait.h> being defined.  */
# include <bits/waitflags.h>
# include <bits/waitstatus.h>

/* Define the macros <sys/wait.h> also would define this way.  */
# define WEXITSTATUS(status) __WEXITSTATUS (status)
# define WTERMSIG(status) __WTERMSIG (status)
# define WSTOPSIG(status) __WSTOPSIG (status)
# define WIFEXITED(status) __WIFEXITED (status)
# define WIFSIGNALED(status) __WIFSIGNALED (status)
```

```
# define WIFSTOPPED(status) __WIFSTOPPED (status)
# ifdef __WIFCONTINUED
#  define WIFCONTINUED(status) __WIFCONTINUED (status)
# endif
#endif /* X/Open or XPG7 and <sys/wait.h> not included.  */

/* _FloatN API tests for enablement.  */
#include <bits/floatn.h>

/* Returned by 'div'.  */
typedef struct
  {
    int quot; /* Quotient.  */
    int rem; /* Remainder.  */
  } div_t;

/* Returned by 'ldiv'.  */
#ifndef __ldiv_t_defined
typedef struct
  {
    long int quot; /* Quotient.  */
    long int rem; /* Remainder.  */
  } ldiv_t;
# define __ldiv_t_defined 1
#endif

#if defined __USE_ISOC99 && !defined __lldiv_t_defined
/* Returned by 'lldiv'.  */
__extension__ typedef struct
  {
    long long int quot; /* Quotient.  */
    long long int rem; /* Remainder.  */
  } lldiv_t;
# define __lldiv_t_defined 1
#endif


/* The largest number rand will return (same as INT_MAX).  */
#define RAND_MAX 2147483647


/* We define these the same for all machines.
   Changes from this to the outside world should be done in '_exit'.  */
#define EXIT_FAILURE 1 /* Failing exit status.  */
#define EXIT_SUCCESS 0 /* Successful exit status.  */


/* Maximum length of a multibyte character in the current locale.  */
#define MB_CUR_MAX (__ctype_get_mb_cur_max ())
extern size_t __ctype_get_mb_cur_max (void) __THROW __wur;
```

```
/* Convert a string to a floating-point number.  */
extern double atof (const char *__nptr)
     __THROW __attribute_pure__ __nonnull ((1)) __wur;
/* Convert a string to an integer.  */
extern int atoi (const char *__nptr)
     __THROW __attribute_pure__ __nonnull ((1)) __wur;
/* Convert a string to a long integer.  */
extern long int atol (const char *__nptr)
     __THROW __attribute_pure__ __nonnull ((1)) __wur;

#ifdef __USE_ISOC99
/* Convert a string to a long long integer.  */
__extension__ extern long long int atoll (const char *__nptr)
     __THROW __attribute_pure__ __nonnull ((1)) __wur;
#endif

/* Convert a string to a floating-point number.  */
extern double strtod (const char *__restrict __nptr,
      char **__restrict __endptr)
     __THROW __nonnull ((1));

#ifdef __USE_ISOC99
/* Likewise for ‘float’ and ‘long double’ sizes of floating-point numbers.  */
extern float strtof (const char *__restrict __nptr,
     char **__restrict __endptr) __THROW __nonnull ((1));

extern long double strtold (const char *__restrict __nptr,
    char **__restrict __endptr)
     __THROW __nonnull ((1));
#endif

/* Likewise for '_FloatN' and '_FloatNx'.  */

#if __HAVE_FLOAT16 && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern _Float16 strtof16 (const char *__restrict __nptr,
  char **__restrict __endptr)
     __THROW __nonnull ((1));
#endif

#if __HAVE_FLOAT32 && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern _Float32 strtof32 (const char *__restrict __nptr,
  char **__restrict __endptr)
     __THROW __nonnull ((1));
#endif

#if __HAVE_FLOAT64 && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern _Float64 strtof64 (const char *__restrict __nptr,
  char **__restrict __endptr)
```

```
     __THROW __nonnull ((1));
#endif

#if __HAVE_FLOAT128 && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern _Float128 strtof128 (const char *__restrict __nptr,
    char **__restrict __endptr)
     __THROW __nonnull ((1));
#endif

#if __HAVE_FLOAT32X && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern _Float32x strtof32x (const char *__restrict __nptr,
    char **__restrict __endptr)
     __THROW __nonnull ((1));
#endif

#if __HAVE_FLOAT64X && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern _Float64x strtof64x (const char *__restrict __nptr,
    char **__restrict __endptr)
     __THROW __nonnull ((1));
#endif

#if __HAVE_FLOAT128X && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern _Float128x strtof128x (const char *__restrict __nptr,
      char **__restrict __endptr)
     __THROW __nonnull ((1));
#endif

/* Convert a string to a long integer.  */
extern long int strtol (const char *__restrict __nptr,
char **__restrict __endptr, int __base)
     __THROW __nonnull ((1));
/* Convert a string to an unsigned long integer.  */
extern unsigned long int strtoul (const char *__restrict __nptr,
  char **__restrict __endptr, int __base)
     __THROW __nonnull ((1));

#ifdef __USE_MISC
/* Convert a string to a quadword integer.  */
__extension__
extern long long int strtoq (const char *__restrict __nptr,
     char **__restrict __endptr, int __base)
     __THROW __nonnull ((1));
/* Convert a string to an unsigned quadword integer.  */
__extension__
extern unsigned long long int strtouq (const char *__restrict __nptr,
       char **__restrict __endptr, int __base)
     __THROW __nonnull ((1));
#endif /* Use misc.  */

#ifdef __USE_ISOC99
```

```
/* Convert a string to a quadword integer.  */
__extension__
extern long long int strtoll (const char *__restrict __nptr,
      char **__restrict __endptr, int __base)
     __THROW __nonnull ((1));
/* Convert a string to an unsigned quadword integer.  */
__extension__
extern unsigned long long int strtoull (const char *__restrict __nptr,
char **__restrict __endptr, int __base)
     __THROW __nonnull ((1));
#endif /* ISO C99 or use MISC.  */

/* Convert a floating-point number to a string.  */
#if __GLIBC_USE (IEC_60559_BFP_EXT)
extern int strfromd (char *__dest, size_t __size, const char *__format,
     double __f)
     __THROW __nonnull ((3));

extern int strfromf (char *__dest, size_t __size, const char *__format,
     float __f)
     __THROW __nonnull ((3));

extern int strfroml (char *__dest, size_t __size, const char *__format,
     long double __f)
     __THROW __nonnull ((3));
#endif

#if __HAVE_FLOAT16 && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern int strfromf16 (char *__dest, size_t __size, const char * __format,
       _Float16 __f)
     __THROW __nonnull ((3));
#endif

#if __HAVE_FLOAT32 && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern int strfromf32 (char *__dest, size_t __size, const char * __format,
       _Float32 __f)
     __THROW __nonnull ((3));
#endif

#if __HAVE_FLOAT64 && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern int strfromf64 (char *__dest, size_t __size, const char * __format,
       _Float64 __f)
     __THROW __nonnull ((3));
#endif

#if __HAVE_FLOAT128 && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern int strfromf128 (char *__dest, size_t __size, const char * __format,
_Float128 __f)
     __THROW __nonnull ((3));
#endif
```

```
#if __HAVE_FLOAT32X && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern int strfromf32x (char *__dest, size_t __size, const char * __format,
_Float32x __f)
     __THROW __nonnull ((3));
#endif

#if __HAVE_FLOAT64X && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern int strfromf64x (char *__dest, size_t __size, const char * __format,
_Float64x __f)
     __THROW __nonnull ((3));
#endif

#if __HAVE_FLOAT128X && __GLIBC_USE (IEC_60559_TYPES_EXT)
extern int strfromf128x (char *__dest, size_t __size, const char * __format,
 _Float128x __f)
     __THROW __nonnull ((3));
#endif


#ifdef __USE_GNU
/* Parallel versions of the functions above which take the locale to
   use as an additional parameter.  These are GNU extensions inspired
   by the POSIX.1-2008 extended locale API.  */
# include <bits/types/locale_t.h>

extern long int strtol_l (const char *__restrict __nptr,
  char **__restrict __endptr, int __base,
  locale_t __loc) __THROW __nonnull ((1, 4));

extern unsigned long int strtoul_l (const char *__restrict __nptr,
    char **__restrict __endptr,
    int __base, locale_t __loc)
     __THROW __nonnull ((1, 4));

__extension__
extern long long int strtoll_l (const char *__restrict __nptr,
char **__restrict __endptr, int __base,
locale_t __loc)
     __THROW __nonnull ((1, 4));

__extension__
extern unsigned long long int strtoull_l (const char *__restrict __nptr,
  char **__restrict __endptr,
  int __base, locale_t __loc)
     __THROW __nonnull ((1, 4));

extern double strtod_l (const char *__restrict __nptr,
char **__restrict __endptr, locale_t __loc)
     __THROW __nonnull ((1, 3));
```

```
extern float strtof_l (const char *__restrict __nptr,
       char **__restrict __endptr, locale_t __loc)
     __THROW __nonnull ((1, 3));

extern long double strtold_l (const char *__restrict __nptr,
      char **__restrict __endptr,
      locale_t __loc)
     __THROW __nonnull ((1, 3));

# if __HAVE_FLOAT16
extern _Float16 strtof16_l (const char *__restrict __nptr,
    char **__restrict __endptr,
    locale_t __loc)
     __THROW __nonnull ((1, 3));
# endif

# if __HAVE_FLOAT32
extern _Float32 strtof32_l (const char *__restrict __nptr,
    char **__restrict __endptr,
    locale_t __loc)
     __THROW __nonnull ((1, 3));
# endif

# if __HAVE_FLOAT64
extern _Float64 strtof64_l (const char *__restrict __nptr,
    char **__restrict __endptr,
    locale_t __loc)
     __THROW __nonnull ((1, 3));
# endif

# if __HAVE_FLOAT128
extern _Float128 strtof128_l (const char *__restrict __nptr,
      char **__restrict __endptr,
      locale_t __loc)
     __THROW __nonnull ((1, 3));
# endif

# if __HAVE_FLOAT32X
extern _Float32x strtof32x_l (const char *__restrict __nptr,
      char **__restrict __endptr,
      locale_t __loc)
     __THROW __nonnull ((1, 3));
# endif

# if __HAVE_FLOAT64X
extern _Float64x strtof64x_l (const char *__restrict __nptr,
      char **__restrict __endptr,
      locale_t __loc)
     __THROW __nonnull ((1, 3));
```

```
# endif

# if __HAVE_FLOAT128X
extern _Float128x strtof128x_l (const char *__restrict __nptr,
char **__restrict __endptr,
locale_t __loc)
     __THROW __nonnull ((1, 3));
# endif
#endif /* GNU */


#ifdef __USE_EXTERN_INLINES
__extern_inline int
__NTH (atoi (const char *__nptr))
{
  return (int) strtol (__nptr, (char **) NULL, 10);
}
__extern_inline long int
__NTH (atol (const char *__nptr))
{
  return strtol (__nptr, (char **) NULL, 10);
}

# ifdef __USE_ISOC99
__extension__ __extern_inline long long int
__NTH (atoll (const char *__nptr))
{
  return strtoll (__nptr, (char **) NULL, 10);
}
# endif
#endif /* Optimizing and Inlining.  */


#if defined __USE_MISC || defined __USE_XOPEN_EXTENDED
/* Convert N to base 64 using the digits "./0-9A-Za-z", least-significant
   digit first.  Returns a pointer to static storage overwritten by the
   next call.  */
extern char *l64a (long int __n) __THROW __wur;

/* Read a number from a string S in base 64 as above.  */
extern long int a64l (const char *__s)
     __THROW __attribute_pure__ __nonnull ((1)) __wur;

#endif /* Use misc || extended X/Open.  */

#if defined __USE_MISC || defined __USE_XOPEN_EXTENDED
# include <sys/types.h>/* we need int32_t... */

/* These are the functions that actually do things.  The `random', `srandom',
   `initstate' and `setstate' functions are those from BSD Unices.
```

```
   The ‘rand’ and ‘srand’ functions are required by the ANSI standard.
   We provide both interfaces to the same random number generator.  */
/* Return a random long integer between 0 and RAND_MAX inclusive.  */
extern long int random (void) __THROW;

/* Seed the random number generator with the given number.  */
extern void srandom (unsigned int __seed) __THROW;

/* Initialize the random number generator to use state buffer STATEBUF,
   of length STATELEN, and seed it with SEED.  Optimal lengths are 8, 16,
   32, 64, 128 and 256, the bigger the better; values less than 8 will
   cause an error and values greater than 256 will be rounded down.  */
extern char *initstate (unsigned int __seed, char *__statebuf,
size_t __statelen) __THROW __nonnull ((2));

/* Switch the random number generator to state buffer STATEBUF,
   which should have been previously initialized by ‘initstate’.  */
extern char *setstate (char *__statebuf) __THROW __nonnull ((1));


# ifdef __USE_MISC
/* Reentrant versions of the ‘random’ family of functions.
   These functions all use the following data structure to contain
   state, rather than global state variables.  */

struct random_data
  {
    int32_t *fptr; /* Front pointer.  */
    int32_t *rptr; /* Rear pointer.  */
    int32_t *state; /* Array of state values.  */
    int rand_type; /* Type of random number generator.  */
    int rand_deg; /* Degree of random number generator.  */
    int rand_sep; /* Distance between front and rear.  */
    int32_t *end_ptr; /* Pointer behind state table.  */
  };

extern int random_r (struct random_data *__restrict __buf,
     int32_t *__restrict __result) __THROW __nonnull ((1, 2));

extern int srandom_r (unsigned int __seed, struct random_data *__buf)
     __THROW __nonnull ((2));

extern int initstate_r (unsigned int __seed, char *__restrict __statebuf,
size_t __statelen,
struct random_data *__restrict __buf)
     __THROW __nonnull ((2, 4));

extern int setstate_r (char *__restrict __statebuf,
       struct random_data *__restrict __buf)
     __THROW __nonnull ((1, 2));
```

```
# endif /* Use misc.  */
#endif /* Use extended X/Open || misc. */


/* Return a random integer between 0 and RAND_MAX inclusive.  */
extern int rand (void) __THROW;
/* Seed the random number generator with the given number.  */
extern void srand (unsigned int __seed) __THROW;

#ifdef __USE_POSIX199506
/* Reentrant interface according to POSIX.1.  */
extern int rand_r (unsigned int *__seed) __THROW;
#endif


#if defined __USE_MISC || defined __USE_XOPEN
/* System V style 48-bit random number generator functions.  */

/* Return non-negative, double-precision floating-point value in [0.0,1.0).  */
extern double drand48 (void) __THROW;
extern double erand48 (unsigned short int __xsubi[3]) __THROW __nonnull ((1));

/* Return non-negative, long integer in [0,2^31).  */
extern long int lrand48 (void) __THROW;
extern long int nrand48 (unsigned short int __xsubi[3])
     __THROW __nonnull ((1));

/* Return signed, long integers in [-2^31,2^31).  */
extern long int mrand48 (void) __THROW;
extern long int jrand48 (unsigned short int __xsubi[3])
     __THROW __nonnull ((1));

/* Seed random number generator.  */
extern void srand48 (long int __seedval) __THROW;
extern unsigned short int *seed48 (unsigned short int __seed16v[3])
     __THROW __nonnull ((1));
extern void lcong48 (unsigned short int __param[7]) __THROW __nonnull ((1));

# ifdef __USE_MISC
/* Data structure for communication with thread safe versions.  This
   type is to be regarded as opaque.  It's only exported because users
   have to allocate objects of this type.  */
struct drand48_data
  {
    unsigned short int __x[3]; /* Current state.  */
    unsigned short int __old_x[3]; /* Old state.  */
    unsigned short int __c; /* Additive const. in congruential formula.  */
    unsigned short int __init; /* Flag for initializing.  */
    __extension__ unsigned long long int __a; /* Factor in congruential
   formula.  */
```

```
  };

/* Return non-negative, double-precision floating-point value in [0.0,1.0).  */
extern int drand48_r (struct drand48_data *__restrict __buffer,
      double *__restrict __result) __THROW __nonnull ((1, 2));
extern int erand48_r (unsigned short int __xsubi[3],
      struct drand48_data *__restrict __buffer,
      double *__restrict __result) __THROW __nonnull ((1, 2));

/* Return non-negative, long integer in [0,2^31).  */
extern int lrand48_r (struct drand48_data *__restrict __buffer,
      long int *__restrict __result)
     __THROW __nonnull ((1, 2));
extern int nrand48_r (unsigned short int __xsubi[3],
      struct drand48_data *__restrict __buffer,
      long int *__restrict __result)
     __THROW __nonnull ((1, 2));

/* Return signed, long integers in [-2^31,2^31).  */
extern int mrand48_r (struct drand48_data *__restrict __buffer,
      long int *__restrict __result)
     __THROW __nonnull ((1, 2));
extern int jrand48_r (unsigned short int __xsubi[3],
      struct drand48_data *__restrict __buffer,
      long int *__restrict __result)
     __THROW __nonnull ((1, 2));

/* Seed random number generator.  */
extern int srand48_r (long int __seedval, struct drand48_data *__buffer)
     __THROW __nonnull ((2));

extern int seed48_r (unsigned short int __seed16v[3],
     struct drand48_data *__buffer) __THROW __nonnull ((1, 2));

extern int lcong48_r (unsigned short int __param[7],
      struct drand48_data *__buffer)
     __THROW __nonnull ((1, 2));
# endif /* Use misc.  */
#endif /* Use misc or X/Open.  */

/* Allocate SIZE bytes of memory.  */
extern void *malloc (size_t __size) __THROW __attribute_malloc__ __wur;
/* Allocate NMEMB elements of SIZE bytes each, all initialized to 0.  */
extern void *calloc (size_t __nmemb, size_t __size)
     __THROW __attribute_malloc__ __wur;

/* Re-allocate the previously allocated block
   in PTR, making the new block SIZE bytes long.  */
/* __attribute_malloc__ is not used, because if realloc returns
   the same pointer that was passed to it, aliasing needs to be allowed
```

```
   between objects pointed by the old and new pointers.  */
extern void *realloc (void *__ptr, size_t __size)
     __THROW __attribute_warn_unused_result__;

#ifdef __USE_GNU
/* Re-allocate the previously allocated block in PTR, making the new
   block large enough for NMEMB elements of SIZE bytes each.  */
/* __attribute_malloc__ is not used, because if reallocarray returns
   the same pointer that was passed to it, aliasing needs to be allowed
   between objects pointed by the old and new pointers.  */
extern void *reallocarray (void *__ptr, size_t __nmemb, size_t __size)
     __THROW __attribute_warn_unused_result__;
#endif

/* Free a block allocated by `malloc', `realloc' or `calloc'.  */
extern void free (void *__ptr) __THROW;

#ifdef __USE_MISC
# include <alloca.h>
#endif /* Use misc.  */

#if (defined __USE_XOPEN_EXTENDED && !defined __USE_XOPEN2K) \
    || defined __USE_MISC
/* Allocate SIZE bytes on a page boundary.  The storage cannot be freed.  */
extern void *valloc (size_t __size) __THROW __attribute_malloc__ __wur;
#endif

#ifdef __USE_XOPEN2K
/* Allocate memory of SIZE bytes with an alignment of ALIGNMENT.  */
extern int posix_memalign (void **__memptr, size_t __alignment, size_t __size)
     __THROW __nonnull ((1)) __wur;
#endif

#ifdef __USE_ISOC11
/* ISO C variant of aligned allocation.  */
extern void *aligned_alloc (size_t __alignment, size_t __size)
     __THROW __attribute_malloc__ __attribute_alloc_size__ ((2)) __wur;
#endif

/* Abort execution and generate a core-dump.  */
extern void abort (void) __THROW __attribute__ ((__noreturn__));


/* Register a function to be called when `exit' is called.  */
extern int atexit (void (*__func) (void)) __THROW __nonnull ((1));

#if defined __USE_ISOC11 || defined __USE_ISOCXX11
/* Register a function to be called when `quick_exit' is called.  */
# ifdef __cplusplus
extern "C++" int at_quick_exit (void (*__func) (void))
```

```
     __THROW __asm ("at_quick_exit") __nonnull ((1));
# else
extern int at_quick_exit (void (*__func) (void)) __THROW __nonnull ((1));
# endif
#endif

#ifdef __USE_MISC
/* Register a function to be called with the status
   given to `exit' and the given argument.  */
extern int on_exit (void (*__func) (int __status, void *__arg), void *__arg)
     __THROW __nonnull ((1));
#endif

/* Call all functions registered with `atexit' and `on_exit',
   in the reverse of the order in which they were registered,
   perform stdio cleanup, and terminate program execution with STATUS.  */
extern void exit (int __status) __THROW __attribute__ ((__noreturn__));

#if defined __USE_ISOC11 || defined __USE_ISOCXX11
/* Call all functions registered with `at_quick_exit' in the reverse
   of the order in which they were registered and terminate program
   execution with STATUS.  */
extern void quick_exit (int __status) __THROW __attribute__ ((__noreturn__));
#endif

#ifdef __USE_ISOC99
/* Terminate the program with STATUS without calling any of the
   functions registered with `atexit' or `on_exit'.  */
extern void _Exit (int __status) __THROW __attribute__ ((__noreturn__));
#endif


/* Return the value of envariable NAME, or NULL if it doesn't exist.  */
extern char *getenv (const char *__name) __THROW __nonnull ((1)) __wur;

#ifdef __USE_GNU
/* This function is similar to the above but returns NULL if the
   programs is running with SUID or SGID enabled.  */
extern char *secure_getenv (const char *__name)
     __THROW __nonnull ((1)) __wur;
#endif

#if defined __USE_MISC || defined __USE_XOPEN
/* The SVID says this is in <stdio.h>, but this seems a better place. */
/* Put STRING, which is of the form "NAME=VALUE", in the environment.
   If there is no `=', remove NAME from the environment.  */
extern int putenv (char *__string) __THROW __nonnull ((1));
#endif

#ifdef __USE_XOPEN2K
```

```
/* Set NAME to VALUE in the environment.
   If REPLACE is nonzero, overwrite an existing value.  */
extern int setenv (const char *__name, const char *__value, int __replace)
     __THROW __nonnull ((2));

/* Remove the variable NAME from the environment.  */
extern int unsetenv (const char *__name) __THROW __nonnull ((1));
#endif

#ifdef __USE_MISC
/* The 'clearenv' was planned to be added to POSIX.1 but probably
   never made it.  Nevertheless the POSIX.9 standard (POSIX bindings
   for Fortran 77) requires this function.  */
extern int clearenv (void) __THROW;
#endif


#if defined __USE_MISC \
    || (defined __USE_XOPEN_EXTENDED && !defined __USE_XOPEN2K8)
/* Generate a unique temporary file name from TEMPLATE.
   The last six characters of TEMPLATE must be "XXXXXX";
   they are replaced with a string that makes the file name unique.
   Always returns TEMPLATE, it's either a temporary file name or a null
   string if it cannot get a unique file name.  */
extern char *mktemp (char *__template) __THROW __nonnull ((1));
#endif

#if defined __USE_XOPEN_EXTENDED || defined __USE_XOPEN2K8
/* Generate a unique temporary file name from TEMPLATE.
   The last six characters of TEMPLATE must be "XXXXXX";
   they are replaced with a string that makes the filename unique.
   Returns a file descriptor open on the file for reading and writing,
   or -1 if it cannot create a uniquely-named file.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
# ifndef __USE_FILE_OFFSET64
extern int mkstemp (char *__template) __nonnull ((1)) __wur;
# else
#  ifdef __REDIRECT
extern int __REDIRECT (mkstemp, (char *__template), mkstemp64)
     __nonnull ((1)) __wur;
#  else
#   define mkstemp mkstemp64
#  endif
# endif
# ifdef __USE_LARGEFILE64
extern int mkstemp64 (char *__template) __nonnull ((1)) __wur;
# endif
#endif
```

```
#ifdef __USE_MISC
/* Similar to mkstemp, but the template can have a suffix after the
   XXXXXX.  The length of the suffix is specified in the second
   parameter.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
# ifndef __USE_FILE_OFFSET64
extern int mkstemps (char *__template, int __suffixlen) __nonnull ((1)) __wur;
# else
#  ifdef __REDIRECT
extern int __REDIRECT (mkstemps, (char *__template, int __suffixlen),
       mkstemps64) __nonnull ((1)) __wur;
#  else
#   define mkstemps mkstemps64
#  endif
# endif
# ifdef __USE_LARGEFILE64
extern int mkstemps64 (char *__template, int __suffixlen)
     __nonnull ((1)) __wur;
# endif
#endif

#ifdef __USE_XOPEN2K8
/* Create a unique temporary directory from TEMPLATE.
   The last six characters of TEMPLATE must be "XXXXXX";
   they are replaced with a string that makes the directory name unique.
   Returns TEMPLATE, or a null pointer if it cannot get a unique name.
   The directory is created mode 700.  */
extern char *mkdtemp (char *__template) __THROW __nonnull ((1)) __wur;
#endif

#ifdef __USE_GNU
/* Generate a unique temporary file name from TEMPLATE similar to
   mkstemp.  But allow the caller to pass additional flags which are
   used in the open call to create the file..

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
# ifndef __USE_FILE_OFFSET64
extern int mkostemp (char *__template, int __flags) __nonnull ((1)) __wur;
# else
#  ifdef __REDIRECT
extern int __REDIRECT (mkostemp, (char *__template, int __flags), mkostemp64)
     __nonnull ((1)) __wur;
#  else
#   define mkostemp mkostemp64
#  endif
# endif
```

```
# ifdef __USE_LARGEFILE64
extern int mkostemp64 (char *__template, int __flags) __nonnull ((1)) __wur;
# endif

/* Similar to mkostemp, but the template can have a suffix after the
   XXXXXX.  The length of the suffix is specified in the second
   parameter.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
# ifndef __USE_FILE_OFFSET64
extern int mkostemps (char *__template, int __suffixlen, int __flags)
     __nonnull ((1)) __wur;
# else
#  ifdef __REDIRECT
extern int __REDIRECT (mkostemps, (char *__template, int __suffixlen,
   int __flags), mkostemps64)
     __nonnull ((1)) __wur;
#  else
#   define mkostemps mkostemps64
#  endif
# endif
# ifdef __USE_LARGEFILE64
extern int mkostemps64 (char *__template, int __suffixlen, int __flags)
     __nonnull ((1)) __wur;
# endif
#endif


/* Execute the given line as a shell command.

   This function is a cancellation point and therefore not marked with
   __THROW.  */
extern int system (const char *__command) __wur;


#ifdef __USE_GNU
/* Return a malloc'd string containing the canonical absolute name of the
   existing named file.  */
extern char *canonicalize_file_name (const char *__name)
     __THROW __nonnull ((1)) __wur;
#endif

#if defined __USE_MISC || defined __USE_XOPEN_EXTENDED
/* Return the canonical absolute name of file NAME.  If RESOLVED is
   null, the result is malloc'd; otherwise, if the canonical name is
   PATH_MAX chars or more, returns null with `errno' set to
   ENAMETOOLONG; if the name fits in fewer than PATH_MAX chars,
   returns the name in RESOLVED.  */
extern char *realpath (const char *__restrict __name,
```

```
       char *__restrict __resolved) __THROW __wur;
#endif


/* Shorthand for type of comparison functions.  */
#ifndef __COMPAR_FN_T
# define __COMPAR_FN_T
typedef int (*__compar_fn_t) (const void *, const void *);

# ifdef __USE_GNU
typedef __compar_fn_t comparison_fn_t;
# endif
#endif
#ifdef __USE_GNU
typedef int (*__compar_d_fn_t) (const void *, const void *, void *);
#endif

/* Do a binary search for KEY in BASE, which consists of NMEMB elements
   of SIZE bytes each, using COMPAR to perform the comparisons.  */
extern void *bsearch (const void *__key, const void *__base,
      size_t __nmemb, size_t __size, __compar_fn_t __compar)
     __nonnull ((1, 2, 5)) __wur;

#ifdef __USE_EXTERN_INLINES
# include <bits/stdlib-bsearch.h>
#endif

/* Sort NMEMB elements of BASE, of SIZE bytes each,
   using COMPAR to perform the comparisons.  */
extern void qsort (void *__base, size_t __nmemb, size_t __size,
   __compar_fn_t __compar) __nonnull ((1, 4));
#ifdef __USE_GNU
extern void qsort_r (void *__base, size_t __nmemb, size_t __size,
     __compar_d_fn_t __compar, void *__arg)
  __nonnull ((1, 4));
#endif


/* Return the absolute value of X.  */
extern int abs (int __x) __THROW __attribute__ ((__const__)) __wur;
extern long int labs (long int __x) __THROW __attribute__ ((__const__)) __wur;

#ifdef __USE_ISOC99
__extension__ extern long long int llabs (long long int __x)
     __THROW __attribute__ ((__const__)) __wur;
#endif


/* Return the `div_t', `ldiv_t' or `lldiv_t' representation
   of the value of NUMER over DENOM. */
```

```
/* GCC may have built-ins for these someday.  */
extern div_t div (int __numer, int __denom)
     __THROW __attribute__ ((__const__)) __wur;
extern ldiv_t ldiv (long int __numer, long int __denom)
     __THROW __attribute__ ((__const__)) __wur;

#ifdef __USE_ISOC99
__extension__ extern lldiv_t lldiv (long long int __numer,
    long long int __denom)
     __THROW __attribute__ ((__const__)) __wur;
#endif


#if (defined __USE_XOPEN_EXTENDED && !defined __USE_XOPEN2K8) \
    || defined __USE_MISC
/* Convert floating point numbers to strings.  The returned values are
   valid only until another call to the same function.  */

/* Convert VALUE to a string with NDIGIT digits and return a pointer to
   this.  Set *DECPT with the position of the decimal character and *SIGN
   with the sign of the number.  */
extern char *ecvt (double __value, int __ndigit, int *__restrict __decpt,
   int *__restrict __sign) __THROW __nonnull ((3, 4)) __wur;

/* Convert VALUE to a string rounded to NDIGIT decimal digits.  Set *DECPT
   with the position of the decimal character and *SIGN with the sign of
   the number.  */
extern char *fcvt (double __value, int __ndigit, int *__restrict __decpt,
   int *__restrict __sign) __THROW __nonnull ((3, 4)) __wur;

/* If possible convert VALUE to a string with NDIGIT significant digits.
   Otherwise use exponential representation.  The resulting string will
   be written to BUF.  */
extern char *gcvt (double __value, int __ndigit, char *__buf)
     __THROW __nonnull ((3)) __wur;
#endif

#ifdef __USE_MISC
/* Long double versions of above functions.  */
extern char *qecvt (long double __value, int __ndigit,
    int *__restrict __decpt, int *__restrict __sign)
     __THROW __nonnull ((3, 4)) __wur;
extern char *qfcvt (long double __value, int __ndigit,
    int *__restrict __decpt, int *__restrict __sign)
     __THROW __nonnull ((3, 4)) __wur;
extern char *qgcvt (long double __value, int __ndigit, char *__buf)
     __THROW __nonnull ((3)) __wur;


/* Reentrant version of the functions above which provide their own
```

```
   buffers.  */
extern int ecvt_r (double __value, int __ndigit, int *__restrict __decpt,
   int *__restrict __sign, char *__restrict __buf,
   size_t __len) __THROW __nonnull ((3, 4, 5));
extern int fcvt_r (double __value, int __ndigit, int *__restrict __decpt,
   int *__restrict __sign, char *__restrict __buf,
   size_t __len) __THROW __nonnull ((3, 4, 5));

extern int qecvt_r (long double __value, int __ndigit,
    int *__restrict __decpt, int *__restrict __sign,
    char *__restrict __buf, size_t __len)
     __THROW __nonnull ((3, 4, 5));
extern int qfcvt_r (long double __value, int __ndigit,
    int *__restrict __decpt, int *__restrict __sign,
    char *__restrict __buf, size_t __len)
     __THROW __nonnull ((3, 4, 5));
#endif /* misc */


/* Return the length of the multibyte character
   in S, which is no longer than N.  */
extern int mblen (const char *__s, size_t __n) __THROW;
/* Return the length of the given multibyte character,
   putting its `wchar_t' representation in *PWC.  */
extern int mbtowc (wchar_t *__restrict __pwc,
   const char *__restrict __s, size_t __n) __THROW;
/* Put the multibyte character represented
   by WCHAR in S, returning its length.  */
extern int wctomb (char *__s, wchar_t __wchar) __THROW;


/* Convert a multibyte string to a wide char string.  */
extern size_t mbstowcs (wchar_t *__restrict  __pwcs,
const char *__restrict __s, size_t __n) __THROW;
/* Convert a wide char string to multibyte string.  */
extern size_t wcstombs (char *__restrict __s,
const wchar_t *__restrict __pwcs, size_t __n)
     __THROW;


#ifdef __USE_MISC
/* Determine whether the string value of RESPONSE matches the affirmation
   or negative response expression as specified by the LC_MESSAGES category
   in the program's current locale.  Returns 1 if affirmative, 0 if
   negative, and -1 if not matching.  */
extern int rpmatch (const char *__response) __THROW __nonnull ((1)) __wur;
#endif


#if defined __USE_XOPEN_EXTENDED || defined __USE_XOPEN2K8
```

```
/* Parse comma separated suboption from *OPTIONP and match against
   strings in TOKENS.  If found return index and set *VALUEP to
   optional value introduced by an equal sign.  If the suboption is
   not part of TOKENS return in *VALUEP beginning of unknown
   suboption.  On exit *OPTIONP is set to the beginning of the next
   token or at the terminating NUL character.  */
extern int getsubopt (char **__restrict __optionp,
      char *const *__restrict __tokens,
      char **__restrict __valuep)
     __THROW __nonnull ((1, 2, 3)) __wur;
#endif


#ifdef __USE_XOPEN
/* Setup DES tables according KEY.  */
extern void setkey (const char *__key) __THROW __nonnull ((1));
#endif


/* X/Open pseudo terminal handling.  */

#ifdef __USE_XOPEN2KXSI
/* Return a master pseudo-terminal handle.  */
extern int posix_openpt (int __oflag) __wur;
#endif

#ifdef __USE_XOPEN_EXTENDED
/* The next four functions all take a master pseudo-tty fd and
   perform an operation on the associated slave:  */

/* Chown the slave to the calling user.  */
extern int grantpt (int __fd) __THROW;

/* Release an internal lock so the slave can be opened.
   Call after grantpt().  */
extern int unlockpt (int __fd) __THROW;

/* Return the pathname of the pseudo terminal slave associated with
   the master FD is open on, or NULL on errors.
   The returned storage is good until the next call to this function.  */
extern char *ptsname (int __fd) __THROW __wur;
#endif

#ifdef __USE_GNU
/* Store at most BUFLEN characters of the pathname of the slave pseudo
   terminal associated with the master FD is open on in BUF.
   Return 0 on success, otherwise an error number.  */
extern int ptsname_r (int __fd, char *__buf, size_t __buflen)
     __THROW __nonnull ((2));
```

```
/* Open a master pseudo terminal and return its file descriptor.  */
extern int getpt (void);
#endif

#ifdef __USE_MISC
/* Put the 1 minute, 5 minute and 15 minute load averages into the first
   NELEM elements of LOADAVG.  Return the number written (never more than
   three, but may be less than NELEM), or -1 if an error occurred.  */
extern int getloadavg (double __loadavg[], int __nelem)
     __THROW __nonnull ((1));
#endif

#if defined __USE_XOPEN_EXTENDED && !defined __USE_XOPEN2K
/* Return the index into the active-logins file (utmp) for
   the controlling terminal.  */
extern int ttyslot (void) __THROW;
#endif

#include <bits/stdlib-float.h>

/* Define some macros helping to catch buffer overflows.  */
#if __USE_FORTIFY_LEVEL > 0 && defined __fortify_function
# include <bits/stdlib.h>
#endif
#ifdef __LDBL_COMPAT
# include <bits/stdlib-ldbl.h>
#endif

__END_DECLS

#endif /* stdlib.h  */
```

51 time.h

```
/* Copyright (C) 1991-2018 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   The GNU C Library is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
   Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public
   License along with the GNU C Library; if not, see
   <http://www.gnu.org/licenses/>.  */

/*
 * ISO C99 Standard: 7.23 Date and time <time.h>
 */

#ifndef _TIME_H
#define _TIME_H 1

#include <features.h>

#define __need_size_t
#define __need_NULL
#include <stddef.h>

/* This defines CLOCKS_PER_SEC, which is the number of processor clock
   ticks per second, and possibly a number of other constants.   */
#include <bits/time.h>

/* Many of the typedefs and structs whose official home is this header
   may also need to be defined by other headers.  */
#include <bits/types/clock_t.h>
#include <bits/types/time_t.h>
#include <bits/types/struct_tm.h>

#if defined __USE_POSIX199309 || defined __USE_ISOC11
# include <bits/types/struct_timespec.h>
#endif

#ifdef __USE_POSIX199309
# include <bits/types/clockid_t.h>
# include <bits/types/timer_t.h>
```

```
# include <bits/types/struct_itimerspec.h>
struct sigevent;
#endif

#ifdef __USE_XOPEN2K
# ifndef __pid_t_defined
typedef __pid_t pid_t;
#  define __pid_t_defined
# endif
#endif

#ifdef __USE_XOPEN2K8
# include <bits/types/locale_t.h>
#endif

#ifdef __USE_ISOC11
/* Time base values for timespec_get.  */
# define TIME_UTC 1
#endif

__BEGIN_DECLS

/* Time used by the program so far (user time + system time).
   The result / CLOCKS_PER_SEC is program time in seconds.  */
extern clock_t clock (void) __THROW;

/* Return the current time and put it in *TIMER if TIMER is not NULL.  */
extern time_t time (time_t *__timer) __THROW;

/* Return the difference between TIME1 and TIME0.  */
extern double difftime (time_t __time1, time_t __time0)
     __THROW __attribute__ ((__const__));

/* Return the `time_t' representation of TP and normalize TP.  */
extern time_t mktime (struct tm *__tp) __THROW;


/* Format TP into S according to FORMAT.
   Write no more than MAXSIZE characters and return the number
   of characters written, or 0 if it would exceed MAXSIZE.  */
extern size_t strftime (char *__restrict __s, size_t __maxsize,
const char *__restrict __format,
const struct tm *__restrict __tp) __THROW;

#ifdef __USE_XOPEN
/* Parse S according to FORMAT and store binary time information in TP.
   The return value is a pointer to the first unparsed character in S.  */
extern char *strptime (const char *__restrict __s,
       const char *__restrict __fmt, struct tm *__tp)
     __THROW;
```

```
#endif

#ifdef __USE_XOPEN2K8
/* Similar to the two functions above but take the information from
   the provided locale and not the global locale.  */

extern size_t strftime_l (char *__restrict __s, size_t __maxsize,
  const char *__restrict __format,
  const struct tm *__restrict __tp,
  locale_t __loc) __THROW;
#endif

#ifdef __USE_GNU
extern char *strptime_l (const char *__restrict __s,
 const char *__restrict __fmt, struct tm *__tp,
 locale_t __loc) __THROW;
#endif


/* Return the 'struct tm' representation of *TIMER
   in Universal Coordinated Time (aka Greenwich Mean Time).  */
extern struct tm *gmtime (const time_t *__timer) __THROW;

/* Return the 'struct tm' representation
   of *TIMER in the local timezone.  */
extern struct tm *localtime (const time_t *__timer) __THROW;

#ifdef __USE_POSIX
/* Return the 'struct tm' representation of *TIMER in UTC,
   using *TP to store the result.  */
extern struct tm *gmtime_r (const time_t *__restrict __timer,
    struct tm *__restrict __tp) __THROW;

/* Return the 'struct tm' representation of *TIMER in local time,
   using *TP to store the result.  */
extern struct tm *localtime_r (const time_t *__restrict __timer,
       struct tm *__restrict __tp) __THROW;
#endif /* POSIX */

/* Return a string of the form "Day Mon dd hh:mm:ss yyyy\n"
   that is the representation of TP in this format.  */
extern char *asctime (const struct tm *__tp) __THROW;

/* Equivalent to 'asctime (localtime (timer))'.  */
extern char *ctime (const time_t *__timer) __THROW;

#ifdef __USE_POSIX
/* Reentrant versions of the above functions.  */

/* Return in BUF a string of the form "Day Mon dd hh:mm:ss yyyy\n"
```

```
   that is the representation of TP in this format.  */
extern char *asctime_r (const struct tm *__restrict __tp,
char *__restrict __buf) __THROW;

/* Equivalent to ‘asctime_r (localtime_r (timer, *TMP*), buf)’.  */
extern char *ctime_r (const time_t *__restrict __timer,
      char *__restrict __buf) __THROW;
#endif /* POSIX */


/* Defined in localtime.c.  */
extern char *__tzname[2]; /* Current timezone names.  */
extern int __daylight; /* If daylight-saving time is ever in use.  */
extern long int __timezone; /* Seconds west of UTC.  */


#ifdef __USE_POSIX
/* Same as above.  */
extern char *tzname[2];

/* Set time conversion information from the TZ environment variable.
   If TZ is not defined, a locale-dependent default is used.  */
extern void tzset (void) __THROW;
#endif

#if defined __USE_MISC || defined __USE_XOPEN
extern int daylight;
extern long int timezone;
#endif

#ifdef __USE_MISC
/* Set the system time to *WHEN.
   This call is restricted to the superuser.  */
extern int stime (const time_t *__when) __THROW;
#endif


/* Nonzero if YEAR is a leap year (every 4 years,
   except every 100th isn't, and every 400th is).  */
#define __isleap(year) \
  ((year) % 4 == 0 && ((year) % 100 != 0 || (year) % 400 == 0))


#ifdef __USE_MISC
/* Miscellaneous functions many Unices inherited from the public domain
   localtime package.  These are included only for compatibility.  */

/* Like ‘mktime’, but for TP represents Universal Time, not local time.  */
extern time_t timegm (struct tm *__tp) __THROW;
```

```
/* Another name for ‘mktime’.  */
extern time_t timelocal (struct tm *__tp) __THROW;

/* Return the number of days in YEAR.  */
extern int dysize (int __year) __THROW  __attribute__ ((__const__));
#endif


#ifdef __USE_POSIX199309
/* Pause execution for a number of nanoseconds.

   This function is a cancellation point and therefore not marked with
   __THROW.  */
extern int nanosleep (const struct timespec *__requested_time,
      struct timespec *__remaining);


/* Get resolution of clock CLOCK_ID.  */
extern int clock_getres (clockid_t __clock_id, struct timespec *__res) __THROW;

/* Get current value of clock CLOCK_ID and store it in TP.  */
extern int clock_gettime (clockid_t __clock_id, struct timespec *__tp) __THROW;

/* Set clock CLOCK_ID to value TP.  */
extern int clock_settime (clockid_t __clock_id, const struct timespec *__tp)
     __THROW;

# ifdef __USE_XOPEN2K
/* High-resolution sleep with the specified clock.

   This function is a cancellation point and therefore not marked with
   __THROW.  */
extern int clock_nanosleep (clockid_t __clock_id, int __flags,
    const struct timespec *__req,
    struct timespec *__rem);

/* Return clock ID for CPU-time clock.  */
extern int clock_getcpuclockid (pid_t __pid, clockid_t *__clock_id) __THROW;
# endif


/* Create new per-process timer using CLOCK_ID.  */
extern int timer_create (clockid_t __clock_id,
 struct sigevent *__restrict __evp,
 timer_t *__restrict __timerid) __THROW;

/* Delete timer TIMERID.  */
extern int timer_delete (timer_t __timerid) __THROW;

/* Set timer TIMERID to VALUE, returning old value in OVALUE.  */
```

```
extern int timer_settime (timer_t __timerid, int __flags,
  const struct itimerspec *__restrict __value,
  struct itimerspec *__restrict __ovalue) __THROW;

/* Get current value of timer TIMERID and store it in VALUE.  */
extern int timer_gettime (timer_t __timerid, struct itimerspec *__value)
     __THROW;

/* Get expiration overrun for timer TIMERID.  */
extern int timer_getoverrun (timer_t __timerid) __THROW;
#endif


#ifdef __USE_ISOC11
/* Set TS to calendar time based in time base BASE.  */
extern int timespec_get (struct timespec *__ts, int __base)
     __THROW __nonnull ((1));
#endif


#ifdef __USE_XOPEN_EXTENDED
/* Set to one of the following values to indicate an error.
     1  the DATEMSK environment variable is null or undefined,
     2  the template file cannot be opened for reading,
     3  failed to get file status information,
     4  the template file is not a regular file,
     5  an error is encountered while reading the template file,
     6  memory allication failed (not enough memory available),
     7  there is no line in the template that matches the input,
     8  invalid input specification Example: February 31 or a time is
specified that can not be represented in a time_t (representing
the time in seconds since 00:00:00 UTC, January 1, 1970) */
extern int getdate_err;

/* Parse the given string as a date specification and return a value
   representing the value.  The templates from the file identified by
   the environment variable DATEMSK are used.  In case of an error
   'getdate_err' is set.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern struct tm *getdate (const char *__string);
#endif

#ifdef __USE_GNU
/* Since 'getdate' is not reentrant because of the use of 'getdate_err'
   and the static buffer to return the result in, we provide a thread-safe
   variant.  The functionality is the same.  The result is returned in
   the buffer pointed to by RESBUFP and in case of an error the return
   value is != 0 with the same values as given above for 'getdate_err'.
```

```
   This function is not part of POSIX and therefore no official
   cancellation point.  But due to similarity with an POSIX interface
   or due to the implementation it is a cancellation point and
   therefore not marked with __THROW.  */
extern int getdate_r (const char *__restrict __string,
      struct tm *__restrict __resbufp);
#endif

__END_DECLS

#endif /* time.h.  */
```

52 tut1.cpp

```
//? tut1.cpp
//? C+- by Ulrich Mutze. Status of work 2020-07-01.
//? Copyright (c) 2020 Ulrich Mutze
//? contact: see contact-info at www.ulrichmutze.de
//?
//? This program is free software: you can redistribute it and/or
//? modify it under the terms of the GNU General Public License as
//? published by the Free Software Foundation, either version 3 of
//? the License, or (at your option) any later version.
//?
//? This program is distributed in the hope that it will be useful,
//? but WITHOUT ANY WARRANTY; without even the implied warranty of
//? MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//? GNU General Public License <http://www.gnu.org/licenses/> for
//? more details.


/*************************************************************************
   Comparing std::vector and CpmArays::V with respect to speed
   of copy-construction and assignment. Notice that this drastic
   advantage of CpmArays::V over std::vector still exists in C++11, where
   moving instead of copying is said to be done where appropriate.
*************************************************************************/
#include <cpmbas.h>
/*************************************************************************
1. C+- filenames start with 'cpm' and contain neither underscores nor
   blanks. Therefore, even projects employing files from many sources are
   not likely to run into problems with non-unique file names.

This directive includes all C+- header files needed here;
'bas' stands for 'basics'. Notice that we have not written
#include "cpmbas.h"
and thus have assumed that our C++ project - let it be named tut1 -
has a list of include directories defined.
Who wants to use C+- for more than a single project, should hold
C+- files in directories separate from his project files.
The general purpose C+- files to be used in the project tut1 are on my
computer in xxx/cpm/cpm0/include and xxx/cpm/cpm0/source.
The files that are specific for the tut1 application are in
yyy/tut1/include and yyy/tut1/source.
In yyy/tut1/include there are the project specific headers
among which there need to be two C+- related configuration files:
cpmdefinitions.h and cpmsystemdependencies.h. Customizing these files
allows us to control the behavior of the C+- classes in our project, as
will be explained later.
In yyy/tut1/source the present project-defining main source tut1.cpp is to
be placed.
```

```
Now we are in a position to define the file content of project
tut1: Its include directories have to be set as
yyy/tut1/include and xxx/cpm/cpm0/include
and the files to be compiled (translation units) have to be chosen as
yyy/tut1/source/tut1.cpp and xxx/cpm/cpm0/source/cpmbas.cpp.

1.1 Details (skip on first reading ?):
The file cpmbas.cpp is actually a collection of all other files
in xxx/cpm/cpm0/source. Its content is

#include "cpmc.cpp"
#include "cpmangle.cpp"
#include "cpmv.cpp"
#include "cpmgreg.cpp"
#include "cpmsystem.cpp"
#include "cpmtypes.cpp"
#include "cpmuc.cpp"
#include "cpmzinterval.cpp"
#include "cpmnumbers.cpp"
#include "cpmmpi.cpp"
#include "cpmword.cpp"
#include "cpmviewport.cpp"
End of 1.1
**************************************************************************/
using namespace CpmRoot;
/**************************************************************************
2. C+- namespace names start with 'Cpm' followed by an identifier
   which starts with a capital letters.

The namespace CpmRoot is rather small, so that the present using directive
should be applicable also in projects in which classes from various
sources are being used.

3. The C+- names for integer, real, and complex numbers are
   CpmRoot::Z, CpmRoot::R, Cpmoot::C. CpmRoot::Word wraps std::string and
   adds functionality to it.

The main effect of this using directive is that we may use these names
simply as Z, R, C, Word.

3.1 Details (skip on first reading ?):
There are the less ubiquitous types L, and N in CpmRoot:
L for unsigned characters ('L' for 'letter', after 'integer promotion'
the values range from 0 to 255), N for 'natural numbers', i.e. unsigned
integers. Types L,Z, and N are typedefs for unsigned char, int,
unsigned int. If we add in file cpmdefinition.h the macro
#define CPM_LONG, we change Z to long int, and N to unsigned long int.
If in cpmdefinitions.h the macro CPM_MP is defined, R has the meaning of
mpreal as defined by the wrapper library MPFRC++ to the multiple precision
library mpfr. See comments to CPM_MP in file cpmdefinitionswrc.h. If
```

```
CPM_MP is not defined R is also a typedef, long double if CPM_LONG and
double else. The types bool and string from standard C++ are useful as
they stand. Nevertheless they will be wrapped into classes - CpmRootX::B
in cpmtypes.h and class CpmRoot::Word in file cpmword.h.
End of 3.1

There is a namespace CpmRootX for the 'less essential essentials':

4. There are classes CpmRootX::B, CpmRootX::Z1, CpmRootX::R1,  for
   Boolean values, integer, and real numbers, which may replace the
   non-class types bool, Z, and R in situations where class func-
   tionality, such as automatic initialization, is indispensable.
   Note 2017-02-18: With C++11 automatic initialization can be achieved
   also for the built-in types. For instance a data member of a class X
         bool b {false};
   can be handeled as if we had declared it as
         B b;

4.1 Details (skip on first reading ?):
Namespace CpmRoot in mainly layed out in file cpmnumbers.h
and CpmRootX in cpmtypes.h. The vigilant reader may argue that we
have not available the declarations of CpmRoot, since we have no
   #include <cpmnumbers.h>
seen so far. Actually, it is there since file cpmbas.h is the following
collection of include directives
#ifndef CPM_BAS_H_
#define CPM_BAS_H_
   #include <cpmtypes.h>
   #include <cpmfr.h>
   #include <cpmvr.h>
   #include <cpmsr.h>
   #include <cpmm.h>
   #include <cpmp.h>
   #include <cpmc.h>
   #include <cpmangle.h>
   #include <cpmgreg.h>
#endif
Also here we don't see
   #include <cpmnumbers.h>
but inspecting cpmtypes.h, we find the commented directive
   #include <cpmsystem.h> // includes cpmwords.h and
      // thus also cpmnumbers.h
which shows the steps which finally include cpmnumbers.h. Acually the
directory cpm/cpm0/include contains 32 header files. This set of files
(as any set of C++ header files) is a directed graph in a natural manner:
There is an edge leading from file1 to file2 iff file2 contains the
directive  #include <file1> .
The files that can be reached from file1 along a path of that graph depend
on file1 and every translation unit which includes such a file1-dependent
header has to be re-compiled upon some change in file1. So, any system
```

```
capable of generating proper make files has to do this dependency
analysis and thus has to work with this  d e p e n d e n c y  g r a p h.
I made a tool consisting of a Ruby program for file analysis
and a C+- program for graph analysis and representation which
creates a  pictorial representation of this dependency graph of an
arbitrary collecton of C/C++ header files. This tool turned out to be very
useful.
It is an obvious logical requirement that the dependency graph is free of
cycles. It is a non-trivial task to organize the header dependencies in a
way that each translation unit gets the declarations which it needs to
know by inclusion of only a few header files. For instance, cpmword.cpp
and cpmtypes.cpp need only cpmtypes.h; cpmc.cpp needs cpmc.h and
cpmtypes.h. The present header file hierarchy is the result of many
simplifying re-organizations. However, many attempts of a simplification
failed since they entailed unacceptable complications elsewhere. I'm not
aware of tools which would automatize relevant parts of this organization
process.
End of 4.1
**************************************************************************/
using namespace CpmArrays;
/**************************************************************************
5. The general-purpose array in C+- is the template class CpmArrays::V
   Most constructors set the first valid index of a non-void V is 1 and
   not 0 as for std::vector. There is, however, a member function which
   shifts all indexes and so allows indexing to start with 0. This allows
   identical code to be used for these two types of arrays.

5.1 Details on C+- namespaces (skip on first reading ?):
The C+- class system introduces many namespaces. Their belonging
togeher under the Cpm banner is not expressed by making them
sub-namespaces of a single Cpm-namespace. Instead, their names all
start with 'Cpm' and continue with a capital letter.
If the name contains a digit, this is 2 or 3 and refers to the space
dimension under consideration. So the namespace CpmDim2 contains the
2D ('flatland') analogs of the geometric classes defined in
namespace CpmDim3.
The namespaces which are declared in the present scope by including
cpmbas.h are
CpmRoot, CpmRootX, CpmSystem, CpmArrays, CpmFunctions, CpmGeo,
CpmGraphics, CpmMPI, CpmStd, CpmTests, CpmTime.
End of 5.1
**************************************************************************/

void info();
Z tutorial1(Z,Z);
   // Declaration of two functions, the definition of which will
   // determine the functionality of the program.

int main(int argc, char* argv[])
   // Traditional main function with types of arguments and return value
```

```
   // according to C/C++. Not all compilers accept using Z instead of
   // int here.
{
 //  Z m=200, n=20000; // default values to be used if no input from the
      //  command line can be found
   Z m=1000, n=1000;
      // 5000, 5000 works fast for CPM_R but not for CPM_RLONG

   V<Word> arg=comLine(argc,argv); // CpmRoot::Word is similar to
      // std::string. CpmArrays::V is the C+- array type. There are two
      // 'schools' about indexing arrays. The 'C school' lets valid
      // indexes start with 0 and the 'Fortran school' which lets valid
      // valid indexes start with 1. Till September 2010 C+- had a
      // 'C school'-array Vl (l for 'lean') and a 'Fortran school'-array
      // V , where the implementation of V was such that any instance of
      // V had a data member of type Vl. The type Vl was used mostly
      // internally in situation where efficiency was considered to be
      // of utmost importance. Finally it became clear that the decision
      // to have these two types of arrays was a mistake. It creates a
      // permanent uncertainty for the programmer whether he should use
      // the default type or should strive for utmost efficiency by
      // using the lean version. Now we have only a single array type V
      // for which the valid indexes may form any contiguous set of
      // integers. This index range is set to start with 1 in most
      // of the available constructors but can be very conveniently reset
      // to any other start index such as 0.
      // CpmArrays::comLine is a function which transforms the
      // traditional argument of main into the more functional data type
      // of C+-. The function name 'comLine' results from
      // the  d e s c r i p t i v e  f u l l  n a m e (DFN)
      // 'command line' by a simple deterministic rule: The rule is
      // to leave unchanged all words with up to four letters and shorten
      // all longer words to 3 by taking the first two letters as they are
      // and take the first of the following consonants as the third
      // letter. Thus 'ooops' gets converted to 'oop'. If there is no
      // such consonant, the vowel at place 3 has to be taken: 'hooiii'
      // gets converted to 'hoo'. In a chain of words, the contracted
      // components get concateneted in a style which is evident from
      // the example that 'descriptive full name' goes to
      // 'desFullName'. In most cases the names come out quite nice, but
      // there are exceptions (not to the rules!): I cosider it ugly to
      // see 'field' abbreviated to 'fil' (notice that 'file', as having
      // only four letters, remains 'file')
      // The simplicity of the rule allows us to use the the function name
      // fluently whenever we remember the full name correctly. Of course,
      // the descriptive full name of a function has to be stated in the
      // declaration of this function. In our case the declaration is in
      // file cpmv.h:
      // inline V<Word> comLine(int argc, char* argv[])
      //    //: command line
```

```
      // The colon hints at the fact that here a descriptive full name
      // is being communicated.
   if (arg.valInd(2)){ // valInd's DFN is 'valid index'
      // C+- arrays judge the validity of an index directly without
      // a need to ask for 'size' and making a comparison.
      Word w2=arg[2];
      if (w2=="?"){ // Thus a question mark as the second entry of the
            // command line triggers a call to function info.
         info();
         return 0;
      }
      else{
         m=w2.toZ(); // Converting Word to Z, for getting a control
            // parameter. There are also conversions toR(), toBool(),
            // and toStr()
         if (arg.valInd(3)) n=arg[3].toZ();
      }
   }
   return tutorial1(m,n);
  }

using namespace std;

C cz(Z i) // an ad-hoc function Z --> C
// CpmRoot::C is the class of complex numbers
{
   R ir=i;
   return C(ir/(1+ir*ir),ir*cpmpi); // there is a CpmRoot::Pi = 3.14159...
      // calling constructor C(R,R)
}

// Now we define the performance measuring functions perf_v and perf_V.
// These functions have to execute identical code for two different
// vector templates: std::vector in perf_v and Cpmrrays::V in perf_V.
// Since we want to iterate these templates, it would not be easy to
// achieve this code doublication by defining a template function.
// Such a template would need at least two template arguments and would
// enforce unnatural expressions.

// This function is designed as to maximize the benefit from reference
// counting and to report the time spent on copy-construction and
// assignment (not of the tedious verification part of the function).

#define Vec std::vector
V<R> perf_v(Z m, Z n)
{
   Z mL=1;
/*
   This line, together with the following
   Word loc(...);
```

```
   CPM_MA
   CPM_MZ
   consitute a convenient idiom for signaling
   entry to and exit from a function block to the log file
   cpmcerr.txt. This writing to the log file takes place only if
   mL is larger or equal to the static data member
   CpmSystem::Message::verbose. This bulky quantity has a
   macro alias 'cpmverbose' (i.e. we have, in file cpmsystem.h
       #define cpmverbose      CpmSystem::Message::verbose
   ) and it can be set by a statement like
       cpmverbose=10;
   Its initial value is 2.
   Soon we will encounter macros cpmtime, cpmwait, cpmdebug.
   See the remarks on the macro namespace in function perf_V.
*/
   Word loc("perf_v(Z,Z)"); // messages use this as name of the function
   CPM_MA // defined in cpmmacros.h, assumes that 'mL' and 'loc' are
      // defined
   R t1=cpmtime(); /* time of function call in seconds from some
       system-defined 'point-zero in time'.
       cpmtime is a short name for function CpmSystem::time defined
       in file cpmsystem.h as follows:
           #define cpmtime         CpmSystem::time
   */
   Z i,j;
   Vec<Vec<C>> vi, vf;
      // are potentially large objects. This is the strong point of
      // C+- arrays and a weak one of std::vector and of std::valarray
    {
      Vec<C> v1(m);
      for (i=0;i<m;++i) v1[i]=cz(i);
         // The preferred form of this statement in C+- is
         // for (i=v1.b();i<=v1.e();++i) v1[i]=cz(i);
         // Written in this form, it is alo correct if v1 is of type V<C>,
         // the C+- array for which indexing starts with 1 instead of 0.

      Vec< Vec<C> > w(n,v1);
         // If we use valarray for Vec, one has to write w(v1,n) instead.
         // This is a inconvenience of the C++ standard library which is
         // due to their multi-source origin.
   //   Vec< Vec<C> > temp1=Vec< Vec<C> >(std::move(w));
   // move (copy) constructor; not working !!!
      Vec< Vec<C> > temp1=w; // copy constructor
      Vec< Vec<C> > temp2=temp1; // copy constructor
      Vec< Vec<C> > temp3=temp2; // copy constructor
      temp2=Vec< Vec<C> >(0); // assignment
      vi=w; // assignment
      vf=temp3; // assignment
   }
   R t2=cpmtime();
```

```
   R t12=t2-t1; // time needed for copy and assignment
   R err=0,sum_v=0;
   for (i=0;i<n;++i){
      for (j=0;j<m;++j){ // sic!
         err+=(vi[i][j]-vf[i][j]).abs();
         sum_v+=(vf[i][j]).abs();
      }
   } // making sure that copying and assignment worked correctly
   R t3=cpmtime();
   R t23=t3-t2;
   V<R> res{t12,t23,err,sum_v};
/*
   Convenient  l i s t  c o n s t r u c t o r, requires C++11
*/
  CPM_MZ
  return res;
}
#undef Vec
#define Vec V
V<R> perf_V(Z m, Z n)
{
   Z mL=1;
   Word loc("perf_V(Z,Z)");
      // Here goes code which is identical to the one of the
      // previous function perf_v. In actual C+- code this
      // is never implemented by copying some piece of code
      // but always by defining the macro CPM_SC (SC for 'shared
      // code' (and undefining it after usage). C+- claims a
      //  m a c r o  n a m e s p a c e :  all macros beginning
      // with the four characters 'CPM_' or the three characters
      // 'cpm'. The upper case macros stand in most cases for a few
      // lines of code and should be recognized by the user as not
      // being a simple identifier. They are thus written without a
      // semi-colon at the end in code since the semi-colon is part
      // of the macro's definition.
      // The lower-case macros, by contrast, are in most cases
      // #define-implemented abbreviations of quantities for which
      // a normal name is available (e.g. CpmSystem::Message::verbose
      // for cpmverbose). Whether these are variables or functions, they
      // are used as normal identifiers in code and the user needs not to
      // be aware of their macro nature. Hence, these macros are thus
      // written in code with a semi-colon at the end. (The semi-colon is
      // not part of the macro's definition.)
      //
   CPM_MA  // macro not followed by a semi-colon
   R t1=cpmtime(); // macro followed by a semi-colon
   Z i,j;
   Vec< Vec<C> > vi, vf;
   {
      Vec<C> v1(m);
```

```
      v1.b_(0); // now indexes start with 0 as in  std::vector
      for (i=0;i<m;++i) v1.cui(i)=cz(i);
            // here we use the 'component (with) unchecked index' cui instead
         // of []. In C+- normal indexing is by default range-checked (by
         // editing the file cpmdefinitions.h this can be changed). Since
         // std::vector[] is not range-checked, it is fair to free also C+-
         // from the burden of range checking.
      Vec< Vec<C> > w(n,v1);
      w.b_(0); // now indexes start with 0 as in  std::vector
      Vec< Vec<C> > temp1=w;
      Vec< Vec<C> > temp2=temp1;
      Vec< Vec<C> > temp3=temp2;
      temp2=Vec< Vec<C> >(0);
      vi=w;
      vf=temp3;
   }
   R t2=cpmtime();
   R t12=t2-t1;
   R err=0,sum_V=0;
   for (i=0;i<n;++i){
      for (j=0;j<m;++j){
         err+=(vi.cui(i).cui(j)-vf.cui(i).cui(j)).abs();
         sum_V+=(vf.cui(i).cui(j)).abs();
      }
   }
   R t3=cpmtime();
   R t23=t3-t2;
   CPM_MZ
   return V<R>{t12,t23,err,sum_V};
}
#undef Vec

Z tutorial1(Z m, Z n)
{
   Z mL=1;
   Word loc("tutorial1(Z,Z)");

   CPM_MA
   V<R> res_V=perf_V(m,n), res_v=perf_v(m,n);
   R t12_V=res_V[1], t12_v=res_v[1];
   R t23_V=res_V[2], t23_v=res_v[2];
   R t_V=t12_V+t23_V, t_v=t12_v+t23_v;
   R err_V=res_V[3], err_v=res_v[3];
   R sum_V=res_V[4], sum_v=res_v[4];
   R fac12=t12_v*cpminv(t12_V);
   R fac=t_v*cpminv(t_V);
   R diff=sum_v-sum_V;
      // diff should be 0 up to numerical noise
   cout<<"m="<<m<<", n="<<n<<endl;
```

```
   cout<<"t12_v="<<t12_v<<", t12_V="<<t12_V<<endl;
   cout<<"t23_v="<<t23_v<<", t23_V="<<t23_V<<endl;
   cout<<"fac12="<<fac12<<endl;
   cout<<" fac="<<fac<<endl;
   cout<<" sum_v="<<sum_v<<", sum_V="<<sum_V<<",  diff= "<<diff<<endl;
      // output of the result to the console
   cpmdebug(m);
   cpmdebug(n);
   cpmdebug(t12_v);
   cpmdebug(t12_V);
   cpmdebug(t23_v);
   cpmdebug(t23_V);
   cpmdebug(fac12);
   cpmdebug(fac);
   cpmdebug(err_v);
   cpmdebug(err_V);
   cpmdebug(sum_v);
   cpmdebug(sum_V);
   cpmdebug(diff);
      // convenient documentation of the main result on
      // the auto-generated log file cpmcerr.txt.
      // Defined in file cpmtypes.h as
   // #define cpmdebug(X) cpmcerr<<endl<<"C+- debug: "<<#X "="<< X <<endl
      // Here are the corresponding lines of this file:

//2020-02-20 with g++-9 -std=c++2a

//C+- debug: m=5000
//
//C+- debug: n=5000
//
//C+- debug: t12_v=81.9568
//
//C+- debug: t12_V=0.001214
//
//C+- debug: t23_v=1.99872
//
//C+- debug: t23_V=3.99376
//
//C+- debug: fac12=67509.7
//
//C+- debug: fac=21.0153
//
//C+- debug: err_v=0
//
//C+- debug: err_V=0
//
//C+- debug: sum_v=1.9631e+11
//
//C+- debug: sum_V=1.9631e+11
```

```
//
//C+- debug: diff=0

//2020-06-27 with g++-9 -std=c++2a
// CPM_USECOUNT, CPM_RANGE_CHECK both not defined. Copied from console output:

// m=5000
// n=5000
// t12_v=2.18456
// t12_V=2.34625
// t23_v=0.848009
// t23_V=1.15306
// fac12=0.931085
// fac=0.866618
// sum_v=1.9631e+11
// sum_V=1.9631e+11
// diff=0

// In this case the loop involving index access was about 2 times faster
// for std::vector than it was for CpmArrays::V. But the part involving
// multple copy constructions was done 67509 faster with CpmArrays::V than
// with std::vector. The main insight is that the move functionality turns
// out to have no significant effect even with the new g++-9 compiler.

//
//counter = 7
// Message: tutorial1(Z,Z) done
//  42.4427 s after program start
// Thus, as expected, the copy and assignment is much faster (in our
// case times 1212 for C+- vectors as it is for STL vectors.

   CPM_MZ
   return 0;
}

void info(){
   cout<<endl<<"takes zero, one, or two integer argument";
   cout<<endl<<"compares the performance or std::vector and CpmArays::V";
   cout<<endl<<"in copy construction and assignment operations"<<endl;
}
// end of tut1.cpp
```

53 survey.txt

```
2020-07-01   output by Ruby class UM2::CpmSurvey

         C+- higher-level names:
 Class/struct content of namespaces, together with
 the header files in which these are declared

Directories under consideration are:
 /home/mutze/e/cpm/tut1/source_publishing
        /home/mutze/e/cpm/tut1/source_publishing

 Number of files considered: 51
 Number of program lines considered: 25595
 Number of classes and structs found: 176

List of namespaces found:
   CpmArrays  introduced in file  cpmsr.h
   CpmFunctions  introduced in file  cpmfo.h
   CpmGeo  introduced in file  cpmangle.h
   CpmGraphics  introduced in file  cpmviewport.h
   CpmMPI  introduced in file  cpmmpi.h
   CpmRoot  introduced in file  cpmtypes.h
   CpmRootX  introduced in file  cpmtypes.h
   CpmStd  introduced in file  cpmnumbers.h
   CpmSystem  introduced in file  cpmsystem.h
   CpmTime  introduced in file  cpmgreg.h
 (not always the file is given that in the
 dependency tree is next to the root)

Classes and structs by namespace:

 Namespace CpmArrays:
   IvZ   class in file   cpmzinterval.h
      'intervals of integers', i.e. contiguous finite subsets of Z

   M   class in file   cpmm.h
      map, associative array, dictionary, hash

   P   class in file   cpmuc.h
      'P' as in 'pointer', constant smart pointer.

   Po   class in file   cpmp.h
      Adding order operations to Pp<T>

   Pp   class in file   cpmp.h
      polymorphic smart pointers

   S   class in file   cpms.h
```

```
   sets of homotypic elements

Sr   class in file   cpmsr.h
   S with a rich interface

T2   class in file   cpmx.h
   homotypic pairs

T3   class in file   cpmx.h
   homotypic triplets

T4   class in file   cpmx.h
   homotypic quartets

UseCount   class in file   cpmuc.h
   support for reference counting and 'copy on write'

V   class in file   cpmv.h
   vector template, indexing starts with 1 by default.

VV   class in file   cpmv.h
   matrices

VVV   class in file   cpmv.h
   tensors of rank 3

VVVV   class in file   cpmv.h
   tensors of rank 4

VVVVa   class in file   cpmva.h
   version of VVVVo with arithmetic operations

VVVVo   class in file   cpmvo.h
   version of VVVV with order-related operations

VVVa   class in file   cpmva.h
   version of VVVo with arithmetic operations

VVVo   class in file   cpmvo.h
   version of VVV with order-related operations

VVa   class in file   cpmva.h
   version of VVo with arithmetic operations

VVo   class in file   cpmvo.h
   version of VV with order-related operations

Va   class in file   cpmva.h
   version of Vo with arithmetic operations
```

```
  Vo   class in file   cpmvo.h
     version of V with order-related operations

  Vp   class in file   cpmp.h
     polymorphic V template

  Vr   class in file   cpmvr.h
     version of Va with a rich interface

  X2   class in file   cpmx.h
     heterotypic pairs

  X3   class in file   cpmx.h
     heterotypic triplets

  X4   class in file   cpmx.h
     heterotypic 4-tuples

  X5   class in file   cpmx.h
     heterotypic 5-tuples

  X6   class in file   cpmx.h
     heterotypic 6-tuples

  X7   class in file   cpmx.h
     heterotypic 7-tuples

  X8   class in file   cpmx.h
     heterotypic 8-tuples


Namespace CpmFunctions:
  Bind1   class in file   cpmf.h
     binding the first parameter

  Bind2   class in file   cpmf.h
     binding the second parameter

  ConvertDomain   class in file   cpmf.h
     converting the function domain

  ConvertRange   class in file   cpmf.h
     converting the function range

  F   class in file   cpmfl.h
     functions as a class

  F1   class in file   cpmfl.h
     functions with one parameter
```

```
F1_1   class in file   cpmfl.h
   as F1, but the first parameter is the function argument

F2   class in file   cpmfl.h
   functions with two parameters

F2_1   class in file   cpmfl.h
   as F2, but the first parameter is the function argument

F2_2   class in file   cpmfl.h
   as F2, but the second parameter is the function argument

F3   class in file   cpmfl.h
   functions with three parameters

F3_1   class in file   cpmfl.h
   as F3, but the first parameter is the function argument

F3_2   class in file   cpmfl.h
   as F3, but the second parameter is the function argument

F3_3   class in file   cpmfl.h
   as F3, but the third parameter is the function argument

F4   class in file   cpmfl.h
   functions with four parameters

F4_1   class in file   cpmfl.h
   as F4, but the first parameter is the function argument

F4_2   class in file   cpmfl.h
   as F4, but the second parameter is the function argument

F4_3   class in file   cpmfl.h
   as F4, but the third parameter is the function argument

F4_4   class in file   cpmfl.h
   as F4, but the fourth parameter is the function argument

F5   class in file   cpmf.h
   functions with five parameters

F6   class in file   cpmf.h
   functions with six parameters

F_2   class in file   cpmf.h
   functions of two variables

Fa   class in file   cpmfa.h
   version of Fo with arithmetics operations
```

```
  FncObj   class in file   cpmfl.h
     function objects

  Fo   class in file   cpmfo.h
     version of F with order-related operations

  Fr   class in file   cpmfr.h
     version of Fa with rich interface


Namespace CpmGeo:
  Angle   class in file   cpmangle.h
     angles


Namespace CpmGraphics:
  ColRef   class in file   cpmviewport.h
     lean 24-bit color-values

  Font   class in file   cpmviewport.h
     only Helvetica 12 needed from this font system of GLUT

  Rec   class in file   cpmviewport.h
     pixel rectangle which always fits Viewport::win()

  Viewport   class in file   cpmviewport.h
     Lean interface to the system's graphical capabilities.

  rgb   class in file   cpmviewport.h
     Z-valued red green blue

  xy   class in file   cpmviewport.h
     graphical points

  xyxy   class in file   cpmviewport.h
     graphical rectangles


Namespace CpmMPI:
  Com   class in file   cpmmpi.h
     class version of MPI's communicator concept for parallel computing

  Com   class in file   cpmmpi.h
     trivial implementation of Com


Namespace CpmRoot:
  Abs   class in file   cpmnumbers.h
     absolute value
```

```
Abs<L>   class in file   cpmnumbers.h
   absolute value

Abs<N>   class in file   cpmnumbers.h
   absolute value

Abs<R>   class in file   cpmnumbers.h
   absolute value

Abs<Z>   class in file   cpmnumbers.h
   absolute value

Abs<bool>   class in file   cpmnumbers.h
   absolute value

Abs<string>   class in file   cpmnumbers.h
   here string is interpreted as an array of L's

AbsSqr   class in file   cpmnumbers.h
   absolute (value) squared

AbsSqr<L>   class in file   cpmnumbers.h
   absolute (value) squared

AbsSqr<N>   class in file   cpmnumbers.h
   absolute (value) squared

AbsSqr<R>   class in file   cpmnumbers.h
   absolute (value) squared

AbsSqr<Z>   class in file   cpmnumbers.h
   absolute (value) squared

AbsSqr<bool>   class in file   cpmnumbers.h
   absolute (value) squared

AbsSqr<string>   class in file   cpmnumbers.h
   here string is interpreted as an array of L's

C   class in file   cpmc.h
   complex numbers

Comp   class in file   cpmnumbers.h
   compare

Comp<L>   class in file   cpmnumbers.h
Comp<N>   class in file   cpmnumbers.h
Comp<R>   class in file   cpmnumbers.h
Comp<Z>   class in file   cpmnumbers.h
```

```
Comp<bool>   class in file   cpmnumbers.h
Comp<string>   class in file   cpmnumbers.h
Conj   class in file   cpmnumbers.h
   conjugation

Conj<L>   class in file   cpmnumbers.h
Conj<N>   class in file   cpmnumbers.h
Conj<R>   class in file   cpmnumbers.h
Conj<Z>   class in file   cpmnumbers.h
Conj<bool>   class in file   cpmnumbers.h
Conj<string>   class in file   cpmnumbers.h
Conv   class in file   cpmnumbers.h
   conversion

Dis   class in file   cpmnumbers.h
Dis<L>   class in file   cpmnumbers.h
Dis<N>   class in file   cpmnumbers.h
Dis<R>   class in file   cpmnumbers.h
Dis<Z>   class in file   cpmnumbers.h
Dis<bool>   class in file   cpmnumbers.h
Dis<string>   class in file   cpmnumbers.h
Hash   class in file   cpmnumbers.h
   hash value

Hash<L>   class in file   cpmnumbers.h
Hash<N>   class in file   cpmnumbers.h
Hash<R>   class in file   cpmnumbers.h
Hash<Z>   class in file   cpmnumbers.h
Hash<bool>   class in file   cpmnumbers.h
Hash<string>   class in file   cpmnumbers.h
IO   class in file   cpmnumbers.h
   input output class template

IO<L>   class in file   cpmnumbers.h
   specialization of IO

IO<N>   class in file   cpmnumbers.h
   specialization of IO

IO<R>   class in file   cpmnumbers.h
   specialization of IO

IO<Z>   class in file   cpmnumbers.h
   specialization of IO

IO<bool>   class in file   cpmnumbers.h
   specialization of IO

IO<string>   class in file   cpmnumbers.h
   specialization of IO
```

```
Inv   class in file   cpmnumbers.h
   inverse

Inv<L>   class in file   cpmnumbers.h
   inverse, defined in an arbitrary manner

Inv<N>   class in file   cpmnumbers.h
   inverse

Inv<R>   class in file   cpmnumbers.h
   inverse

Inv<Z>   class in file   cpmnumbers.h
   inverse

Inv<bool>   class in file   cpmnumbers.h
   inverse

Inv<string>   class in file   cpmnumbers.h
   reverse string

Name   class in file   cpmword.h
   tool class template for defining the nameOf function

Name<CpmRootX::fpVoidToVoid>   class in file   cpmtypes.h
   also this type needs a name

Name<CpmRootX::fpWordToVoid>   class in file   cpmtypes.h
   also this type needs a name

Name<L>   class in file   cpmword.h
   L: letter

Name<N>   class in file   cpmword.h
   N: natural numbers

Name<R>   class in file   cpmword.h
   multiple precision real numbers

Name<Z>   class in file   cpmword.h
   Z: integers

Name<bool>   class in file   cpmword.h
   specialization

Name<char>   class in file   cpmword.h
   specialization

Name<string>   class in file   cpmword.h
```

```
     specialization

  Neutrals   class in file   cpmnumbers.h
     neutral elements

  Neutrals<L>   class in file   cpmnumbers.h
  Neutrals<N>   class in file   cpmnumbers.h
  Neutrals<R>   class in file   cpmnumbers.h
  Neutrals<Z>   class in file   cpmnumbers.h
  Neutrals<bool>   class in file   cpmnumbers.h
  Neutrals<string>   class in file   cpmnumbers.h
  Ran   class in file   cpmnumbers.h
     random value

  Ran<L>   class in file   cpmnumbers.h
  Ran<N>   class in file   cpmnumbers.h
  Ran<R>   class in file   cpmnumbers.h
  Ran<Z>   class in file   cpmnumbers.h
  Ran<bool>   class in file   cpmnumbers.h
  Ran<string>   class in file   cpmnumbers.h
  Root   class in file   cpmword.h
     Provides the basic functions for dealing with CpmRoot's

  Test   class in file   cpmnumbers.h
     test value

  Test<L>   class in file   cpmnumbers.h
  Test<N>   class in file   cpmnumbers.h
  Test<R>   class in file   cpmnumbers.h
  Test<Z>   class in file   cpmnumbers.h
  Test<bool>   class in file   cpmnumbers.h
  Test<string>   class in file   cpmnumbers.h
  ToWord   class in file   cpmword.h
  ToWord<L>   class in file   cpmword.h
  ToWord<N>   class in file   cpmword.h
  ToWord<R>   class in file   cpmword.h
  ToWord<Z>   class in file   cpmword.h
  ToWord<bool>   class in file   cpmword.h
  ToWord<string>   class in file   cpmword.h
  Word   class in file   cpmword.h
     wrapping charcter strings, rich functionaliy


Namespace CpmRootX:
  B   class in file   cpmtypes.h
     boolean values as a class

  R1   class in file   cpmtypes.h
     real numbers as a class
```

```
  Z1   class in file   cpmtypes.h
     integer numbers as a class


Namespace CpmSystem:
  Error   class in file   cpmsystem.h
     used if C+- classes throw errors

  Exception   class in file   cpmsystem.h
     simple debugging tool

  IFileStream   class in file   cpmsystem.h
     input file stream

  Message   class in file   cpmsystem.h
     provides basic output (unidirectional communication)

  OFileStream   class in file   cpmsystem.h
     output file stream

  Timer   class in file   cpmsystem.h
     implements e.g. getSecondsLeft()


Namespace CpmTime:
  Greg   class in file   cpmgreg.h
     time, date, and Gregorian Calendar

  TimeStyle   class in file   cpmgreg.h
     how to interprete time with respect to the globe
```

54 headerdependencies.txt

```
2020-07-01    output of Ruby class UM1::HeaderFileHierarchy
The directories under consideration are:
 /home/mutze/e/cpm
        tut1/source_publishing
The 33 header files under consideration are in alphabetic order:
    cpmangle.h
    cpmbas.h
    cpmbasicinterfaces.h
    cpmbasictypes.h
    cpmc.h
    cpmcompdef.h
    cpmdefinitions.h
    cpmf.h
    cpmfa.h
    cpmfl.h
    cpmfo.h
    cpmfr.h
    cpmgreg.h
    cpminterfaces.h
    cpmm.h
    cpmmacros.h
    cpmmpi.h
    cpmnumbers.h
    cpmp.h
    cpms.h
    cpmsr.h
    cpmsystem.h
    cpmsystemdependencies.h
    cpmtypes.h
    cpmuc.h
    cpmv.h
    cpmva.h
    cpmviewport.h
    cpmvo.h
    cpmvr.h
    cpmword.h
    cpmx.h
    cpmzinterval.h

 Which header files are included in a header file ?


cpmangle.h includes:
  cpmc.h  cpmtypes.h  cpmv.h
cpmbas.h includes:
  cpmangle.h  cpmc.h  cpmfr.h  cpmgreg.h
  cpmm.h  cpmp.h  cpmsr.h  cpmtypes.h
```

```
  cpmvr.h
cpmbasicinterfaces.h includes:
  cpmbasictypes.h
cpmbasictypes.h includes:
  cpmdefinitions.h
cpmc.h includes:
  cpmword.h
cpmcompdef.h includes:
  none of the files under consideration
cpmdefinitions.h includes:
  cpmcompdef.h
cpmf.h includes:
  cpmv.h
cpmfa.h includes:
  cpmfo.h  cpmtypes.h
cpmfl.h includes:
  cpmuc.h  cpmword.h
cpmfo.h includes:
  cpmf.h
cpmfr.h includes:
  cpmfa.h  cpmvo.h
cpmgreg.h includes:
  cpmangle.h
cpminterfaces.h includes:
  cpmbasicinterfaces.h  cpmmpi.h  cpmnumbers.h
cpmm.h includes:
  cpms.h
cpmmacros.h includes:
  none of the files under consideration
cpmmpi.h includes:
  cpmdefinitions.h
cpmnumbers.h includes:
  cpmbasictypes.h
cpmp.h includes:
  cpmv.h
cpms.h includes:
  cpmtypes.h  cpmvo.h
cpmsr.h includes:
  cpms.h
cpmsystem.h includes:
  cpmword.h
cpmsystemdependencies.h includes:
  none of the files under consideration
cpmtypes.h includes:
  cpmmacros.h  cpmsystem.h  cpmx.h
cpmuc.h includes:
  cpmbasicinterfaces.h
cpmv.h includes:
  cpmfl.h  cpmmacros.h  cpmzinterval.h
cpmva.h includes:
```

```
  cpmtypes.h  cpmvo.h
cpmviewport.h includes:
  cpmv.h
cpmvo.h includes:
  cpmv.h
cpmvr.h includes:
  cpmva.h
cpmword.h includes:
  cpminterfaces.h
cpmx.h includes:
  cpmword.h
cpmzinterval.h includes:
  cpmsystem.h  cpmx.h

 In which header files the header files are included ?


cpmangle.h is included in:
  cpmgreg.h  cpmbas.h
cpmbas.h is included in:
  no header file
cpmbasicinterfaces.h is included in:
  cpminterfaces.h  cpmuc.h
cpmbasictypes.h is included in:
  cpmbasicinterfaces.h  cpmnumbers.h
cpmc.h is included in:
  cpmbas.h  cpmangle.h
cpmcompdef.h is included in:
  cpmdefinitions.h
cpmdefinitions.h is included in:
  cpmmpi.h  cpmbasictypes.h
cpmf.h is included in:
  cpmfo.h
cpmfa.h is included in:
  cpmfr.h
cpmfl.h is included in:
  cpmv.h
cpmfo.h is included in:
  cpmfa.h
cpmfr.h is included in:
  cpmbas.h
cpmgreg.h is included in:
  cpmbas.h
cpminterfaces.h is included in:
  cpmword.h
cpmm.h is included in:
  cpmbas.h
cpmmacros.h is included in:
  cpmtypes.h  cpmv.h
cpmmpi.h is included in:
```

```
  cpminterfaces.h
cpmnumbers.h is included in:
  cpminterfaces.h
cpmp.h is included in:
  cpmbas.h
cpms.h is included in:
  cpmsr.h  cpmm.h
cpmsr.h is included in:
  cpmbas.h
cpmsystem.h is included in:
  cpmtypes.h  cpmzinterval.h
cpmsystemdependencies.h is included in:
  no header file
cpmtypes.h is included in:
  cpms.h  cpmbas.h  cpmangle.h  cpmfa.h
  cpmva.h
cpmuc.h is included in:
  cpmfl.h
cpmv.h is included in:
  cpmvo.h  cpmp.h  cpmangle.h  cpmf.h
  cpmviewport.h
cpmva.h is included in:
  cpmvr.h
cpmviewport.h is included in:
  no header file
cpmvo.h is included in:
  cpms.h  cpmfr.h  cpmva.h
cpmvr.h is included in:
  cpmbas.h
cpmword.h is included in:
  cpmsystem.h  cpmfl.h  cpmx.h  cpmc.h
cpmx.h is included in:
  cpmtypes.h  cpmzinterval.h
cpmzinterval.h is included in:
  cpmv.h
```