

Header file: #include "cpx.hh"

Libraries: libcpx.a (static)
libcpx.so (dynamic)

linux-2.6.32-504.12.2.el6.x86_64
gcc-4.4.7 20120313

Member **variables** (public):

- ▶ `x` = scalar int, double, long double **real part**
▶ `y` = scalar int, double, long double **imaginary part**

Constructor (public)

- ▶ `cpx<scalar> a` initialisation to zero
 - ▶ `cpx<scalar> a(1, 0)` initialisation to values given
 - ▶ `cpx<scalar> a = b` initialisation to b (cpx, or real)
 - ▶ `cpx<scalar> a(b)` initialisation to b (cpx, or real)

Assign (public)

- `cpx<scalar> a = b` initialisation to b (cpx, or real)

Cast operators

- ▶ `int(z)` convert cpx → int
 - ▶ `double(z)` convert cpx → double
 - ▶ `long double(z)` convert cpx → long double

Negative (friend)

- `cpx<scalar> a = -b`initialisation to b (const&, or &&)

Conjugate (friend)

- `cpx<scalar> a = ~b`initialisation to b^* (`const&`, or `&&`)

Algebra (friends)

- ▶ `cpx<scalar> a *= b` b is (`const&`, or `&&`), (`cpx`, or `real`)
- ▶ `cpx<scalar> a /= b` b is (`const&`, or `&&`), (`cpx`, or `real`)

Functions (friends)

- ▶ `fabs(z)` complex norm
- ▶ `phi(z)` complex phase [0 ... $2\pi i$]
- ▶ `phi2(z)` complex phase [- π ... + π]
- ▶ `sqrt(z)` branch+ square root
- ▶ `exp(z)` exponentiation
- ▶ `log(z)` logarithm

Print (friend)

- ▶ `cout << z << endl;` print $z = a + ib$

Usage examples:

- ▶ `return cpx<scalar> (0, z)` return temp argument from function call
- ▶ `cpx<scalar> z = e^(cpx<scalar> (0, 1)*pi)` $z = e^{i\pi}$

Note

- ▶ protect the `^` operator with `(...)` as its precedence is low
`cout << e^(j*pi) << endl;` should read `cout << (e^(j*pi)) ...`

Description

The **`cpx class`** is a very slim complex number class (2 member variables, constructors, cast operators – of which the last could actually be dropped).

The main body of operators are friend type, sparing memory usage for large arrays of `cpx` objects (by not incorporating them).

This is essential in applications such as Fourier transform, or noise analysis, where long vectors of complex type scalars are created.

There are 2 phase returning functions, one in mathematics mode [0 ... 2π] and one in electronics-style mode [- π ... + π].

For obvious reasons `fabs()` keeps the (rather counter-intuitive) notation used for absolute value in `<cmath>`.

The **`makefile`** is designed to work modularly with other projects.

The **`test file`** is illustrative for some of the more interesting aspects coded in the class.

Note: the class offers also π as `cpx4::PI`.

```
// section : NXV4 package
// code    : CPX class _____
//
// supports: move semantix
//           operators + - * / ^   L,R-val context
//           += -= *= /=
//           fabs()
//           phi () [ 0 ... 2.pi] o/p
//           phi2() [-pi ... +pi] o/p
//           ~ - unary operators L,R-val
//           cpx<arithmetic> i.e. int
//                           double
//                           long double
//           type_casts between the types
//           cout << overload
//
//           TOTAL = 27+504 instantiated op/f's
//
// author  : M. Dima
//           NXV4 Solutions
//
// date    : CPX4/v1.0.alfa   / Sat Dec  3 17:13:57 CET 2016
//           CPX4/v2.0.alfa   / Fri Oct 20 17:13:57 CET 2017
// ****
// #pragma once
// #include<iostream>
// #include<type_traits>
// #include<utility>

using std::cout ;
using std::endl ;
using std::ostream ;
using std::decay ;
using std::enable_if ;
using std::is_same ;
using std::is_arithmetic ;
using std::declval ;

namespace cpx4
{
//=====
extern double cosmetic = 1.0e-4 ;
long double PI = 3.14159265359 ;
long double lim = 1.0e-160 ;
long double lm2 = std::sqrt(lim) ;
long double LIM = 1.0/lim ;
long double LD2 = std::sqrt(LIM) ;
long double AD2 = std::log(LD2) ;
}
```

```

template<typename scalar=double>
class cpx
{public:      scalar x, y ; 
  // 
  cpx()           // plain ; } 
  {x = y = 0 ; } 
  // 
  cpx(      scalar a )    // manual ; } 
  {x = a ; } 
  y = 0 ; } 
  // 
  cpx(      scalar a,      scalar b )    // manual ; } 
  {x = a ; } 
  y = b ; } 
  // 
  cpx(const cpx& A)           // copy ; } 
  {x = A.x ; } 
  y = A.y ; } 
template<typename U>
cpx(cpx<U> const& A)
{x = A.x ; } 
  y = A.y ; } 
// ..... 
explicit operator int()      {return x ; } 
explicit operator double()    {return x ; } 
explicit operator long double() {return x ; } 
// ..... 
// 
scalar& operator= (const scalar& A)
{x = A ; } 
  y = 0 ; } 
  return const_cast<scalar&>(A) ; } 
  cpx& operator= (const      cpx& A) ; } 
  {x = A.x ; } 
  y = A.y ; } 
  return const_cast<cpx&>(A) ; } 
template<typename U>
cpx<U>& operator= (cpx<U> const& A)
{x = A.x ; } 
  y = A.y ; } 
  return const_cast<cpx<U>&>(A) ; } 
// 
// ----- 
// 
template<typename T>
friend cpx<T> operator- (cpx<T> const&) ; } 
template<typename T>
friend cpx<T>&& operator- (cpx<T>&& ) ; } 
template<typename T>
friend cpx<T> operator~ (cpx<T> const&) ; } 
template<typename T>
friend cpx<T>&& operator~ (cpx<T>&& ) ; } 

```

```

template<typename T>
friend T fabs(cpx<T> const&) ;  

template<typename T>
friend T phi (cpx<T> const&) ; // [ 0 ... 2PI]  

template<typename T>
friend T phi2(cpx<T> const&) ; // [-PI ... +PI]  

//  

#include "friend_op_plus.cxx"  

#include "friend_op_mnus.cxx"  

#include "friend_op_mult.cxx"  

#include "friend_op_divd.cxx"  

#include "friend_op_expo.cxx"  

#include "friend_op_alog.cxx"  

#include "friend_op_comp.cxx"  

#include "friend_op_equl.cxx"  

//  

// .....  

//  

friend std::ostream& operator<<(std::ostream& ,  

                                     cpx<auto> const& ) ;  

//  

//-----  

//  

~cpx()  

{  

};  

//=====  

};  

  

template<typename T>
friend  

cpx<T>  

log (cpx<T> const&) ;  

  

template<typename T>
friend  

cpx<T>&&  

log (cpx<T> &&) ;  

  

template<typename T>
friend  

cpx<T>  

sqrt (cpx<T> const&) ;  

  

template<typename T>
friend  

cpx<T>&&  

sqrt (cpx<T> &&) ;  

  

template<typename T>
friend  

cpx<T>  

exp (cpx<T> const&) ;  

  

template<typename T>
friend  

cpx<T>&&  

exp (cpx<T> &&) ;  


```

```

template<typename L, typename R>
friend
bool
operator>(cpx<L> const&, cpx<R> const&) ;;

template<typename L, typename R>
friend
bool
operator<(cpx<L> const&, cpx<R> const&) ;;

template<typename L, typename R>
friend
bool
operator<(cpx<L> const&, R const&) ;;

template<typename L, typename R>
friend
bool
operator>(cpx<L> const&, R const&) ;;

template<typename L, typename R>
friend
bool
operator>(L const&, cpx<R> const&) ;;

template<typename L, typename R>
friend
bool
operator<(L const&, cpx<R> const&) ;;

template<typename L, typename R>
friend
cpx<typename decay<decltype(declval<L>() / declval<R>())>::type>
operator/(cpx<L> const& A, cpx<R> const& B);
template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>() / declval<R>())>::type,
R>::value,
cpx<R>
>::type&&
operator/(cpx<L> const& A, cpx<R> && B);

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>() / declval<R>())>::type,
R>::value,
cpx<L>
>::type
operator/(cpx<L> const& A, cpx<R> && B);

```

```

template<typename L, typename R>
friend
    typename
enable_if<is_same<typename decay<decltype(declval<L>()) /
                           declval<R>() )>::type,
    L>::value,
    cpx<L>
    >::type&&
operator/(cpx<L> && A, cpx<R> const& B);

template<typename L, typename R>
friend
    typename
enable_if<!is_same<typename decay<decltype(declval<L>()) /
                           declval<R>() )>::type,
    L>::value,
    cpx<R>
    >::type
operator/(cpx<L> && A, cpx<R> const& B);

template<typename L, typename R>
friend
    typename
enable_if<is_same<typename decay<decltype(declval<L>()) /
                           declval<R>() )>::type,
    L>::value,
    cpx<L>
    >::type&&
operator/(cpx<L> && A, cpx<R> && B);

template<typename L, typename R>
friend
    typename
enable_if<is_same<typename decay<decltype(declval<L>()) /
                           declval<R>() )>::type,
    R>::value
    &&
    !is_same<typename decay<decltype(declval<L>()) /
                           declval<R>() )>::type,
    L>::value,
    cpx<R>
    >::type&&
operator/(cpx<L> && A, cpx<R> && B);

template<typename L, typename R>
friend
    typename
enable_if<is_arithmetic<typename decay<decltype(declval<L>())>::type
                           >::value,
                           cpx<typename decay<decltype(declval<L>()) /
                           declval<R>() )>::type>
                           >::type
operator/(L const& A, cpx<R> const& B);

```

```

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>()) />::type,
                                         declval<R>() )>::type,
R>::value
&&
is_arithmetic<typename decay<decltype(declval<L>()) />::type
                                         >::value
                                         >::value,
cpx<L>
>::type
operator/(L const& A, cpx<R>&& B);

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>()) />::type,
                                         declval<R>() )>::type,
R>::value
&&
is_arithmetic<typename decay<decltype(declval<L>()) />::type
                                         >::value
                                         >::value,
cpx<R>
>::type&&
operator/(L const& A, cpx<R>&& B);

template<typename L, typename R>
friend
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>()) />::type
                                         >::value,
cpx<typename decay<decltype(declval<L>()) />::type>
                                         declval<R>() )>::type>
                                         >::type
operator/(cpx<L> const& A, R const& B);

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>()) />::type,
                                         declval<R>() )>::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>()) />::type
                                         >::value
                                         >::value,
cpx<R>
>::type
operator/(cpx<L> && A, R const& B);

```

```

template<typename L, typename R>
friend
    typename
    enable_if<is_same<typename decay<decltype(declval<L>()) /
                                declval<R>() )>>
        >::type,
    L>::value
    &&
    is_arithmetic<typename decay<decltype(declval<R>())>>
        >::type
    >::value,
    cpx<L>
    >::type&&
operator/(cpx<L> && A, R const& B);

template<typename L, typename R>
friend
    cpx<R>&
operator/=(cpx<L>& A, cpx<R>& B);

template<typename L, typename R>
friend
    typename
    enable_if<is_arithmetic<typename decay<decltype(declval<R>())>>
        >::type
    >::value
    R>::type&
operator/=(cpx<L>& A, R& B);

template<typename L, typename R>
friend
    R&&
operator/=(cpx<L>& A, R && B);

template<typename L, typename R>
friend
    cpx<R>&
operator/=(L& A, cpx<R>& B);

template<typename L, typename R>
friend
    cpx<R>&&
operator/=(L& A, cpx<R>&& B);

template<typename L, typename R>
friend
    bool
operator==(cpx<L> const&, cpx<R> const&) ;;

template<typename L, typename R>
friend
    bool
operator==(cpx<L> const&, R const&) ;;

```

```

template<typename L, typename R>
friend
bool
operator==(L const&, cpx<R> const&) ;;

template<typename L, typename R>
friend
bool
operator!=(cpx<L> const&, cpx<R> const&) ;;

template<typename L, typename R>
friend
bool
operator!=(cpx<L> const&, R const&) ;;

template<typename L, typename R>
friend
bool
operator!=(L const&, cpx<R> const&) ;;

template<typename L, typename R>
friend
cpx<typename decay<decltype(declval<L>() * declval<R>())>::type>
operator^(cpx<L> const&, cpx<R> const&) ;;

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>() * declval<R>())>::type,
R>::value,
cpx<R>
>::type&&
operator^(cpx<L> const&, cpx<R> &&) ;;

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>() * declval<R>())>::type,
R>::value,
cpx<L>
>::type
operator^(cpx<L> const&, cpx<R> &&) ;;

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>() * declval<R>())>::type,
L>::value,
cpx<L>
>::type&&
operator^(cpx<L> &&, cpx<R> const&) ;;

```

```

template<typename L, typename R>
friend
    typename
enable_if<!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>
                           >::type,
                           L>::value,
                           cpx<R>
                           >::type
operator^(cpx<L> && , cpx<R> const& ) ;


template<typename L, typename R>
friend
    typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>
                           >::type,
                           L>::value,
                           cpx<L>
                           >::type&&
operator^(cpx<L> && , cpx<R> && ) ;


template<typename L, typename R>
friend
    typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>
                           >::type,
                           R>::value
                           &&
                           !is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>
                           >::type,
                           L>::value,
                           cpx<R>
                           >::type&&
operator^(cpx<L> && , cpx<R> && ) ;


template<typename L, typename R>
friend
    typename
enable_if<is_arithmetic<typename decay<decltype(declval<L>())>
                           >::type
                           >::value,
                           cpx<typename decay<decltype(declval<L>())*>
                           declval<R>()>::type>
                           >::type
operator^(L const& , cpx<R> const& ) ;
```

```

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
R>::value
&&
is_arithmetic<typename decay<decltype(declval<L>())>::type
                           >::value
cpx<L>
>::type
operator^(L const& , cpx<R>&& ) ; 

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
R>::value
&&
is_arithmetic<typename decay<decltype(declval<L>())>::type
                           >::value
cpx<R>
>::type&&
operator^(L const& , cpx<R>&& ) ; 

template<typename L, typename R>
friend
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>::type
                           >::value
cpx<typename decay<decltype(declval<L>())*>
                           declval<R>()>::type>
                           >::type
operator^(cpx<L> const& , R const& ) ; 

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>::type
                           >::value
cpx<R>
>::type
operator^(cpx<L> && , R const& ) ; 

```

```

template<typename L, typename R>
friend
    typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>
                           >::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>
                           >::type
                           >::value,
cpx<L>
>::type&&
operator^(cpx<L> && , R const& ) ;
;

template<typename L, typename R>
friend
    cpx<typename decay<decltype(declval<L>() - declval<R>())>::type>
operator-(cpx<L> const& A, cpx<R> const& B);

template<typename L, typename R>
friend
    typename
enable_if<is_same<typename decay<decltype(declval<L>() - declval<R>())>
                           >::type,
R>::value,
cpx<R>
>::type&&
operator-(cpx<L> const& A, cpx<R> && B);
template<typename L, typename R>
friend
    typename
enable_if<!is_same<typename decay<decltype(declval<L>() - declval<R>())>
                           >::type,
R>::value,
cpx<L>
>::type
operator-(cpx<L> const& A, cpx<R> && B);

template<typename L, typename R>
friend
    typename
enable_if<is_same<typename decay<decltype(declval<L>() - declval<R>())>
                           >::type,
L>::value,
cpx<L>
>::type&&
operator-(cpx<L> && A, cpx<R> const& B);

```

```

template<typename L, typename R>
friend
    typename
enable_if<!is_same<typename decay<decltype(declval<L>())-
                           declval<R>() >
                           >::type,
                           L>::value,
                           cpx<R>
                           >::type
operator-(cpx<L> && A, cpx<R> const& B);

template<typename L, typename R>
friend
    typename
enable_if<is_same<typename decay<decltype(declval<L>())-
                           declval<R>() >
                           >::type,
                           L>::value,
                           cpx<L>
                           >::type&&
operator-(cpx<L> && A, cpx<R> && B);

template<typename L, typename R>
friend
    typename
enable_if<is_same<typename decay<decltype(declval<L>())-
                           declval<R>() >
                           >::type,
                           R>::value
                           &&
                           !is_same<typename decay<decltype(declval<L>())-
                           declval<R>() >
                           >::type,
                           L>::value,
                           cpx<R>
                           >::type&&
operator-(cpx<L> && A, cpx<R> && B);

template<typename L, typename R>
friend
    typename
enable_if<is_arithmetic<typename decay<decltype(declval<L>())>
                           >::type
                           >::value,
                           cpx<typename decay<decltype(declval<L>())-
                           declval<R>() >>::type>
                           >::type
operator-(L const& A, cpx<R> const& B);

```

```

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>())-
                           declval<R>() )>
                           >::type,
R>::value
&&
is_arithmetic<typename decay<decltype(declval<L>())>::type
                           >::value
                           >::value,
cpx<L>
>::type
operator-(L const& A, cpx<R>&& B);

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>())-
                           declval<R>() )>
                           >::type,
R>::value
&&
is_arithmetic<typename decay<decltype(declval<L>())>::type
                           >::value
                           >::value,
cpx<R>
>::type&&
operator-(L const& A, cpx<R>&& B);

template<typename L, typename R>
friend
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>::type
                           >::value,
cpx<typename decay<decltype(declval<L>())-
                           declval<R>()>::type>
                           >::type
operator-(cpx<L> const& A, R const& B);

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>())-
                           declval<R>() )>
                           >::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>::type
                           >::value
                           >::value,
cpx<R>
>::type
operator-(cpx<L> && A, R const& B);

```

```

template<typename L, typename R>
friend
    typename
enable_if<is_same<typename decay<decltype(declval<L>())-
                           declval<R>() )>>
                           >::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>>
                           >::type
                           >::value,
cpx<L>
>::type&&
operator-(cpx<L> && A, R const& B);

template<typename L, typename R>
friend
    cpx<R>&
operator=(cpx<L>& A, cpx<R>& B);

template<typename L, typename R>
friend
    typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>>
                           >::type
                           >::value,
R>::type&
operator=(cpx<L>& A, R& B);

template<typename L, typename R>
friend
    R&&
operator=(cpx<L>& A, R && B);

template<typename L, typename R>
friend
    cpx<R>&
operator=(L& A, cpx<R>& B);

template<typename L, typename R>
friend
    cpx<R>&&
operator=(L& A, cpx<R>&& B);

template<typename L, typename R>
friend
    cpx<typename decay<decltype(declval<L>() * declval<R>())>>
                           >::type>
operator*(cpx<L> const& A, cpx<R> const& B);

```

```

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
R>::value,
cpx<R>
>::type&&
operator*(cpx<L> const& A, cpx<R> && B);

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
R>::value,
cpx<L>
>::type
operator*(cpx<L> const& A, cpx<R> && B);

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
L>::value,
cpx<L>
>::type&&
operator*(cpx<L> && A, cpx<R> const& B);

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
L>::value,
cpx<R>
>::type
operator*(cpx<L> && A, cpx<R> const& B);

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
L>::value,
cpx<L>
>::type&&
operator*(cpx<L> && A, cpx<R> && B);

```

```

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
R>::value
&&
!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
L>::value,
cpx<R>
>::type&&
operator*(cpx<L> && A, cpx<R> && B);

template<typename L, typename R>
friend
typename
enable_if<is_arithmetic<typename decay<decltype(declval<L>())>::type
                           >::value,
cpx<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type>
>::type
operator*(L const& A, cpx<R> const& B);

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
R>::value
&&
is_arithmetic<typename decay<decltype(declval<L>())>::type
                           >::value,
cpx<L>
>::type
operator*(L const& A, cpx<R>&& B);

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
R>::value
&&
is_arithmetic<typename decay<decltype(declval<L>())>::type
                           >::value,
cpx<R>
>::type&&
operator*(L const& A, cpx<R>&& B);

```

```

template<typename L, typename R>
friend
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>>>
>::value,
cpx<typename decay<decltype(declval<L>() * declval<R>())>>>
>::type
operator*(cpx<L> const& A, R const& B);

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>() * declval<R>())>>>
>::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>>>
>::value
cpx<R>
>::type
operator*(cpx<L> && A, R const& B);

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>() * declval<R>())>>>
>::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>>>
>::value
cpx<L>
>::type&&
operator*(cpx<L> && A, R const& B);

template<typename L, typename R>
friend
cpx<R>&
operator*=(cpx<L>& A, cpx<R>& B);

template<typename L, typename R>
friend
cpx<R>&&
operator*=(cpx<L>& A, cpx<R> && B);

template<typename L, typename R>
friend
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>>>
>::value,
R>::type&
operator*=(cpx<L>& A, R& B);

```

```

template<typename L, typename R>
friend
R&&
operator*(cpx<L>& A, R && B);

template<typename L, typename R>
friend
cpx<R>&
operator*(L& A, cpx<R>& B);

template<typename L, typename R>
friend
cpx<R>&&
operator*(L& A, cpx<R>&& B);

template<typename L, typename R>
friend
cpx<typename decay<decltype(declval<L>() +
                           declval<R>())>::type>
operator+(cpx<L> const& A, cpx<R> const& B);

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>() +
                           declval<R>())>::type,
                           R>::value,
                           cpx<R>
                           >::type>&&
operator+(cpx<L> const& A, cpx<R> && B);

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>() +
                           declval<R>())>::type,
                           R>::value,
                           cpx<L>
                           >::type>
operator+(cpx<L> const& A, cpx<R> && B);

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>() +
                           declval<R>())>::type,
                           L>::value,
                           cpx<L>
                           >::type>&&
operator+(cpx<L> && A, cpx<R> const& B);

```

```

template<typename L, typename R>
friend
    typename
enable_if<!is_same<typename decay<decltype(declval<L>()) +
                           declval<R>() >>
                           >::type,
                           L>::value,
                           cpx<R>
                           >::type
operator+(cpx<L> && A, cpx<R> const& B);

template<typename L, typename R>
friend
    typename
enable_if<is_same<typename decay<decltype(declval<L>()) +
                           declval<R>() >>
                           >::type,
                           L>::value,
                           cpx<L>
                           >::type&&
operator+(cpx<L> && A, cpx<R> && B);

template<typename L, typename R>
friend
    typename
enable_if<is_same<typename decay<decltype(declval<L>()) +
                           declval<R>() >>
                           >::type,
                           R>::value
                           &&
                           !is_same<typename decay<decltype(declval<L>()) +
                           declval<R>() >>
                           >::type,
                           L>::value,
                           cpx<R>
                           >::type&&
operator+(cpx<L> && A, cpx<R> && B);

template<typename L, typename R>
friend
    typename
enable_if<is_arithmetic<typename decay<decltype(declval<L>()) >>
                           >::type
                           >::value,
                           cpx<typename decay<decltype(declval<L>() +
                           declval<R>())>>::type>
                           >::type
operator+(L const& A, cpx<R> const& B);

```

```

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>()) +
                           declval<R>() )>
                           >::type,
R>::value
&&
is_arithmetic<typename decay<decltype(declval<L>())>::type
                           >::value
                           >::value,
cpx<L>
>::type
operator+(L const& A, cpx<R>&& B);

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>()) +
                           declval<R>() )>
                           >::type,
R>::value
&&
is_arithmetic<typename decay<decltype(declval<L>())>::type
                           >::value
                           >::value,
cpx<R>
>::type&&
operator+(L const& A, cpx<R>&& B);

template<typename L, typename R>
friend
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>::type
                           >::value,
cpx<typename decay<decltype(declval<L>()) +
                           declval<R>() )>::type>
                           >::type
operator+(cpx<L> const& A, R const& B);

template<typename L, typename R>
friend
typename
enable_if<!is_same<typename decay<decltype(declval<L>()) +
                           declval<R>() )>
                           >::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>::type
                           >::value
                           >::value,
cpx<R>
>::type
operator+(cpx<L> && A, R const& B);

```

```

template<typename L, typename R>
friend
typename
enable_if<is_same<typename decay<decltype(declval<L>() +
                                         declval<R>())>>
            >::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>>
            >::type
            >::value,
cpx<L>
>::type&&
operator+(cpx<L> && A, R const& B);

template<typename L, typename R>
friend
cpx<R>&&
operator+=(cpx<L>& A, cpx<R>& B);

template<typename L, typename R>
friend
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>>
            >::type
            >::value,
R>::type&
operator+=(cpx<L>& A, R& B);

template<typename L, typename R>
friend
R&&
operator+=(cpx<L>& A, R && B);

template<typename L, typename R>
friend
cpx<R>&
operator+=(L& A, cpx<R>& B);

template<typename L, typename R>
friend
cpx<R>&&
operator+=(L& A, cpx<R>&& B);

```

```
// section : NXV4 package
// code    : CPX class _____
//
// supports: move semantix
//           operators + - * / ^   L,R-val context
//           += -= *= /=
//           fabs()
//           phi () [ 0 ... 2.pi] o/p
//           phi2() [-pi ... +pi] o/p
//           ~ - unary operators L,R-val
//           cpx<arithmetic> i.e. int
//                           double
//                           long double
//           type_casts between the types
//           cout << overload
//
//           TOTAL = 27+504 instantiated op/f's
//
// author  : M. Dima
//           NXV4 Solutions
//
// date    : CPX4/v1.0.alfa   / Sat Dec  3 17:13:57 CET 2016
//           CPX4/v2.0.alfa   / Fri Oct 20 17:13:57 CET 2017
// ****
#include <iostream>
#include <cmath>
#include <type_traits>

#include "cpx.hh"
//
using std::cout ;
using std::endl ;
//
namespace cpx4
{
// =====
template<typename T>
cpx<T>
operator- (cpx<T> const& A)
{return cpx<T> (-A.x,-A.y) ; }

template<typename T>
cpx<T>&&
operator- (cpx<T> && A)
{A.x = -A.x ; A.y = -A.y
 return std::forward<cpx<T>>(A) ; }
```

```

template<typename T>
cpx<T>
operator~ (cpx<T> const& A)
{return cpx<T>( A.x,-A.y) ;}

template<typename T>
cpx<T>&&
operator~ (cpx<T> && A)
{A.x = A.x ; A.y = -A.y ;
 return std::forward<cpx<T>>(A) ;}
// _____

template<typename T>
T fabs(cpx<T> const& A)
{long double tmp = A.x*A.x+A.y*A.y;return T(std::sqrt(tmp));}

template<typename T> // o/p = [0 ... +2PI]
T phi (const cpx<T>& A)
{long double zm = A.x*A.x+A.y*A.y ;
 if (zm < lim) zm = lim ;
 if (A.y >=0) return T(acos(A.x/std::sqrt(zm))) ;
 else return T(2*PI - acos(A.x/std::sqrt(zm))) ;}

template<typename T> // o/p = [-PI ... +PI]
T phi2(const cpx<T>& A)
{T p = cpx4::phi(A) ; if (p > PI) return T(p-2*PI) ;
 return p ;}
// _____

std::ostream& operator<<(std::ostream& os, const cpx<auto>& Z)
{char W = '\b' ;
 if (is_same<typename decay<decltype(Z)>::type,
      cpx<int>
      >::value)
 {W = 'I' ;}
 if (is_same<typename decay<decltype(Z)>::type,
      cpx<long double>
      >::value)
 {W = 'L' ;}
 if (Z.y > 0)
 {if (std::fabs(Z.x) < cosmetic)
   {if (std::fabs(Z.y-1) < cosmetic)
     {os << "i " << W ; return os;}
     if (std::fabs(Z.y) > cosmetic)
     {os << Z.y << "i " << W ; return os;}
     os << "0 " << W ; return os;}
   else {if (std::fabs(Z.y-1) < cosmetic)
     {os << Z.x << "+i " << W ;
      if (std::fabs(Z.y) > cosmetic)
      {os << Z.x << "+" << Z.y << "i " << W;return os;}
      os << Z.x << " " << W ; return os;}}
   }
 if (std::fabs(Z.y) < cosmetic)
 {if (std::fabs(Z.x) < cosmetic)
   {os << "0 " << W ; return os;}
   else {os << Z.x << " " << W ; return os;}}
 if (std::fabs(Z.x) < cosmetic)
 {if (std::fabs(Z.y+1) < cosmetic)
   {os << "-i " << W ;
    os << Z.y << "i " << W ; return os;}
   return os;}}

```

```

else {if (std::fabs(Z.y+1) < cosmetic)
      {os << Z.x << "-i " << W ;           return os; }
      os << Z.x << "--" << -Z.y << "i " << W ;   return os; } }

// _____
#include "op_plus.cxx"
#include "op_mnus.cxx"
#include "op_mult.cxx"
#include "op_divd.cxx"
#include "op_expo.cxx"
#include "op_alog.cxx"
#include "op_comp.cxx"
#include "op_equl.cxx"

//=====
// 
#include "cpx.ie"
};

template<typename T>
cpx<T>
log (cpx<T> const& A)
{T r = A.x * A.x + A.y * A.y ;
 if (r < lim) return cpx<T>(-184.2, 0.0) ;
 return cpx<T>(std::log(r)/2, cpx4::phi(A)) ;
}

template<typename T>
cpx<T>&&
log (cpx<T> && A)
{T r = A.x * A.x + A.y * A.y ;
 if (r < lim) {A.x = -184.2 ;
               A.y = 0.0 ;
               return std::forward<cpx<T>>(A) ;
}
 A.y = cpx4::phi(A) ; A.x = std::log(r)/2 ;
 return std::forward<cpx<T>>(A) ;
}

template<typename T>
cpx<T>
sqrt (cpx<T> const& A)
{T r = A.x * A.x + A.y * A.y ;
 r = std::sqrt(r) ;
 T x = std::sqrt(( A.x+r)/2) ;
 T y = std::sqrt((-A.x+r)/2) ;
 if (A.y < 0) return cpx<T>(-x, y) ;
 return cpx<T>(x, y) ;
}

template<typename T>
cpx<T>&&
sqrt (cpx<T> && A)
{T r = A.x * A.x + A.y * A.y ;
 r = std::sqrt(r) ;
 T x = std::sqrt(( A.x+r)/2) ;
 T y = std::sqrt((-A.x+r)/2) ;
 if (A.y < 0) {A.x = -x ; A.y = y ;
               return std::forward<cpx<T>>(A) ;
}
 A.x = x ; A.y = y ;
 return std::forward<cpx<T>>(A) ;
}

```

```

template<typename T>
cpx<T>
exp (cpx<T> const& A)
{T x = A.y - 2*PI*int(A.y/2.0/PI);
 T y = std::sin(x); x = std::cos(x);
 if (A.x > AD2) return cpx<T>(LD2*x, LD2*y);
 T r = std::exp(A.x);
 return cpx<T>(r*x, r*y);
}

template<typename T>
cpx<T>&&
exp (cpx<T> && A)
{T x = A.y - 2*PI*int(A.y/2.0/PI);
 T y = std::sin(x); x = std::cos(x);
 if (A.x > AD2) {A.x = LD2*x; A.y = LD2*y;
 return std::forward<cpx<T>>(A);}
 T r = std::exp(A.x);
 A.x = r*x; A.y = r*y;
 return std::forward<cpx<T>>(A);
}

template<typename L, typename R>
bool
operator>(cpx<L> const& A, cpx<R> const& B)
{if (A.x*A.x+A.y*A.y >
    B.x*B.x+B.y*B.y) return true;
 return false;
}

template<typename L, typename R>
bool
operator<(cpx<L> const& A, cpx<R> const& B)
{if (A.x*A.x+A.y*A.y <
    B.x*B.x+B.y*B.y) return true;
 return false;
}

template<typename L, typename R>
bool
operator<(cpx<L> const& A, R const& B)
{if (B < 0) return false;
 if (A.x*A.x+A.y*A.y <
     B*B) return true;
 return false;
}

template<typename L, typename R>
bool
operator>(cpx<L> const& A, R const& B)
{if (B < 0) return true;
 if (A.x*A.x+A.y*A.y >
     B*B) return true;
 return false;
}

template<typename L, typename R>
bool
operator>(L const& A, cpx<R> const& B)
{if (A < 0) return false;
 if (A*A >
     B.x*B.x+B.y*B.y) return true;
 return false;
}

```

```

template<typename L, typename R>
bool
operator<(L const& A, cpx<R> const& B)
{if (A < 0) return true
 if (A*A < B.x*B.x+B.y*B.y) return true
 return false ;}

template<typename L, typename R>
cpx<typename decay<decltype(declval<L>()) /
 declval<R>()>::type>
operator/(cpx<L> const& A, cpx<R> const& B)
{using S = typename decay<decltype(declval<L>()) /
 declval<R>()>::type ;
 long double tmp = B.x * B.x + B.y * B.y ;
 if (tmp < lim) tmp = lim ;
 long double tmx = A.x * B.x + A.y * B.y ;
 return cpx<S>(tmx/tmp, (A.y*B.x-A.x*B.y)/tmp) ;}

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>()) /
 declval<R>()>::type,
R>::value,
cpx<R>
>::type&&
operator/(cpx<L> const& A, cpx<R> && B)
{long double tmp = B.x * B.x + B.y * B.y ;
 if (tmp < lim) tmp = lim ;
 long double tmx = A.x * B.x + A.y * B.y ;
 B.y = (A.y*B.x-A.x*B.y)/tmp ; B.x = tmx/tmp ;
 return std::forward<cpx<R>>(B) ;}

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>()) /
 declval<R>()>::type,
R>::value,
cpx<L>
>::type
operator/(cpx<L> const& A, cpx<R> && B)
{long double tmp = B.x * B.x + B.y * B.y ;
 if (tmp < lim) tmp = lim ;
 long double tmx = A.x * B.x + A.y * B.y ;
 cpx<L> Z(tmx/tmp, (A.y*B.x-A.x*B.y)/tmp) ; return Z ;}

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>()) /
 declval<R>()>::type,
L>::value,
cpx<L>
>::type&&
operator/(cpx<L> && A, cpx<R> const& B)
{long double tmp = B.x * B.x + B.y * B.y ;
 if (tmp < lim) tmp = lim ;
 long double tmx = A.x * B.x + A.y * B.y ;
 A.y = (A.y*B.x-A.x*B.y)/tmp ; A.x = tmx/tmp ;
 return std::forward<cpx<L>>(A) ;}

```

```

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>()) />>
            declval<R>() )>
            >::type,
L>::value,
cpx<R>
>::type
operator/(cpx<L> && A, cpx<R> const& B)
{long double tmp = B.x * B.x + B.y * B.y ;
 if (tmp < lim) tmp = lim ;
 long double tmx = A.x * B.x + A.y * B.y ;
 return cpx<R>(tmx/tmp, (A.y*B.x-A.x*B.y)/tmp) ;
}

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>()) />>
            declval<R>() )>
            >::type,
L>::value,
cpx<L>
>::type&&
operator/(cpx<L> && A, cpx<R> && B)
{long double tmp = B.x * B.x + B.y * B.y ;
 if (tmp < lim) tmp = lim ;
 long double tmx = A.x * B.x + A.y * B.y ;
 A.y = (A.y*B.x-A.x*B.y)/tmp ; A.x = tmx/tmp ;
 return std::forward<cpx<L>>(A) ;
}

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>()) />>
            declval<R>() )>
            >::type,
R>::value
&&
!is_same<typename decay<decltype(declval<L>()) />>
            declval<R>() )>
            >::type,
L>::value,
cpx<R>
>::type&&
operator/(cpx<L> && A, cpx<R> && B)
{long double tmp = B.x * B.x + B.y * B.y ;
 if (tmp < lim) tmp = lim ;
 long double tmx = A.x * B.x + A.y * B.y ;
 B.y = (A.y*B.x-A.x*B.y)/tmp ; B.x = tmx/tmp ;
 return std::forward<cpx<R>>(B) ;
}

```

```

template<typename L, typename R>
typename
enable_if<is_arithmetic<typename decay<decltype(declval<L>()) />>::type
           >::value,
cpx<typename decay<decltype(declval<L>() / declval<R>())>>::type>
           >::type
operator/(L const& A, cpx<R> const& B)
{using S = typename decay<decltype(declval<L>() / declval<R>())>>::type ; ;
long double tmp = B.x * B.x + B.y * B.y ; ;
if (tmp < lim) tmp = lim ; ;
long double tmx = A * B.x ; ;
return cpx<S>(tmx/tmp, (-A*B.y)/tmp) ; }

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>() / declval<R>())>>::type,
           R>::value
           &&
           is_arithmetic<typename decay<decltype(declval<L>())>>::type
           >::value,
           cpx<L>
           >::type
operator/(L const& A, cpx<R>&& B)
{long double tmp = B.x * B.x + B.y * B.y ; ;
if (tmp < lim) tmp = lim ; ;
long double tmx = A * B.x ; ;
return cpx<L>(tmx/tmp, (-A*B.y)/tmp) ; }

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>() / declval<R>())>>::type,
           R>::value
           &&
           is_arithmetic<typename decay<decltype(declval<L>())>>::type
           >::value,
           cpx<R>
           >::type&&
operator/(L const& A, cpx<R>&& B)
{long double tmp = B.x * B.x + B.y * B.y ; ;
if (tmp < lim) tmp = lim ; ;
long double tmx = A * B.x ; ;
B.y = -A*B.y/tmp ; B.x = tmx/tmp ; ;
return std::forward<cpx<R>>(B) ; }

```

```

template<typename L, typename R>
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>>::type
          >::value,
cpx<typename decay<decltype(declval<L>() / declval<R>())>>::type>
          >::type
operator/(cpx<L> const& A, R const& B)
{using S = typename decay<decltype(declval<L>() / declval<R>())>>::type ; ;
long double tmp = B * B ; ;
if (tmp < lim) tmp = B / lim ; ;
else tmp = 1.0 / B ; ;
return cpx<S>(A.x*tmp, A.y*tmp) ; }

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>() / declval<R>())>>::type,
          L>::value
          &&
          is_arithmetic<typename decay<decltype(declval<R>())>>::type
          >::value,
          cpx<R>
          >::type
operator/(cpx<L> && A, R const& B)
{long double tmp = B * B ; ;
if (tmp < lim) tmp = B / lim ; ;
else tmp = 1.0 / B ; ;
return cpx<R>(A.x*tmp, A.y*tmp) ; }

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>() / declval<R>())>>::type,
          L>::value
          &&
          is_arithmetic<typename decay<decltype(declval<R>())>>::type
          >::value,
          cpx<L>
          >::type&&
operator/(cpx<L> && A, R const& B)
{long double tmp = B * B ; ;
if (tmp < lim) tmp = B / lim ; ;
else tmp = 1.0 / B ; ;
A.x = A.x*tmp ; A.y = A.y*tmp ; ;
return std::forward<cpx<L>>(A) ; }

```

```

template<typename L, typename R>
cpx<R>&
operator/=(cpx<L>& A, cpx<R>& B)
{long double tmp = B.x * B.x + B.y * B.y ; ;
 if (tmp < lim) tmp = lim ;
 long double tmx = A.x * B.x + A.y * B.y ;
 A.y = (A.y*B.x-A.x*B.y)/tmp ; A.x = tmx/tmp ;
 return B ; }

template<typename L, typename R>
cpx<R>&&
operator/=(cpx<L>& A, cpx<R> && B)
{long double tmp = B.x * B.x + B.y * B.y ; ;
 if (tmp < lim) tmp = lim ;
 long double tmx = A.x * B.x + A.y * B.y ;
 A.y = (A.y*B.x-A.x*B.y)/tmp ; A.x = tmx/tmp ;
 return std::forward<cpx<R>>(B) ; }

template<typename L, typename R>
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>::type
>::value,
R>::type&
operator/=(cpx<L>& A, R& B)
{long double tmp = B * B ; ;
 if (tmp < lim) tmp = B / lim ;
 else tmp = 1.0 / B ;
 A.x = A.x * tmp ; A.y = A.y * tmp ;
 return B ; }

template<typename L, typename R>
R&&
operator/=(cpx<L>& A, R && B)
{long double tmp = B * B ; ;
 if (tmp < lim) tmp = B / lim ;
 else tmp = 1.0 / B ;
 A.x = A.x * tmp ; A.y = A.y * tmp ;
 return std::forward<R>(B) ; }

template<typename L, typename R>
cpx<R>&
operator/=(L& A, cpx<R>& B)
{long double tmp = B.x * B.x + B.y * B.y ; ;
 if (tmp < lim) tmp = lim ;
 long double tmx = A * B.x ;
 A = tmx/tmp ; return B ; }

template<typename L, typename R>
cpx<R>&&
operator/=(L& A, cpx<R>&& B)
{long double tmp = B.x * B.x + B.y * B.y ; ;
 if (tmp < lim) tmp = lim ;
 long double tmx = A * B.x ;
 A = tmx/tmp ;
 return std::forward<cpx<R>>(B) ; }

```

```

template<typename L, typename R>
bool
operator==(cpx<L> const& A, cpx<R> const& B)
{return ( (A.x==B.x) && (A.y==B.y) ) ;}

template<typename L, typename R>
bool
operator==(cpx<L> const& A, R const& B)
{return ( A.x==B && A.y==0 ) ;}

template<typename L, typename R>
bool
operator==(L const& A, cpx<R> const& B)
{return ( A==B.x && B.y==0 ) ;}

template<typename L, typename R>
bool
operator!=(cpx<L> const& A, cpx<R> const& B)
{return ( (A.x!=B.x) || (A.y!=B.y) ) ;}

template<typename L, typename R>
bool
operator!=(cpx<L> const& A, R const& B)
{return ( A.x!=B || A.y!=0 ) ;}

template<typename L, typename R>
bool
operator!=(L const& A, cpx<R> const& B)
{return ( A!=B.x || B.y!=0 ) ;}

template<typename L, typename R>
cpx<typename decay<decltype(declval<L>()*
                           declval<R>())>::type>
operator^(cpx<L> const& A, cpx<R> const& B)
{using S = typename decay<decltype(declval<L>()*
                                    declval<R>())>::type ;
 L r = fabs(A) ; if (r == 0) return cpx<S>(0,0) ;
 r = std::log(r) ;
 L p = cpx4::phi(A) ;
 S W = std::exp(r * B.x - p * B.y) ;
 S P = p * B.x + r * B.y ;
 int n = P / (2*PI) ;
 P = P - n * (2*PI) ;
 return cpx<S>(W*cos(P), W*sin(P)) ;}

```

```

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
R>::value,
cpx<R>
>::type&&
operator^(cpx<L> const& A, cpx<R> && B)
{L r = fabs(A)
 if (r == 0) return std::forward<cpx<R>>(B);
 r = std::log(r);
 L p = cpx4::phi(A);
 R W = std::exp(r * B.x - p * B.y);
 R P = p * B.x + r * B.y;
 int n = P / (2*PI);
 P = P - n * (2*PI);
 B.x = W*cos(P);
 B.y = W*sin(P);
 return std::forward<cpx<R>>(B);}

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
R>::value,
cpx<L>
>::type
operator^(cpx<L> const& A, cpx<R> && B)
{L r = fabs(A); if (r == 0) return cpx<L>(0,0);
 r = std::log(r);
 L p = cpx4::phi(A);
 L W = std::exp(r * B.x - p * B.y);
 L P = p * B.x + r * B.y;
 int n = P / (2*PI);
 P = P - n * (2*PI);
 return cpx<L>(W*cos(P), W*sin(P));}

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>() )>::type,
L>::value,
cpx<L>
>::type&&
operator^(cpx<L> && A, cpx<R> const& B)
{L r = fabs(A)
 if (r == 0) return std::forward<cpx<L>>(A);
 r = std::log(r);
 L p = cpx4::phi(A);
 L W = std::exp(r * B.x - p * B.y);
 L P = p * B.x + r * B.y;
 int n = P / (2*PI);
 P = P - n * (2*PI);
 A.x = W*cos(P); A.y = W*sin(P);
 return std::forward<cpx<L>>(A);}
```

```

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>>
                           >::type, L>::value,
                           cpx<R> >::type
operator^(cpx<L> && A, cpx<R> const& B)
{L r = fabs(A) ; if (r == 0) return cpx<R>(0,0) ;
 r = std::log(r);
L p = cpx4::phi(A);
R W = std::exp(r * B.x - p * B.y);
R P = p * B.x + r * B.y;
int n = P / (2*PI);
P = P - n * (2*PI);
return cpx<R>(W*cos(P), W*sin(P)) ;}

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>>
                           >::type,
                           L>::value, cpx<L>
                           >::type&&
operator^(cpx<L> && A, cpx<R> && B)
{L r = fabs(A);
 if (r == 0) return std::forward<cpx<L>>(A);
 r = std::log(r);
L p = cpx4::phi(A);
L W = std::exp(r * B.x - p * B.y);
L P = p * B.x + r * B.y;
int n = P / (2*PI);
P = P - n * (2*PI);
A.x = W*cos(P);
A.y = W*sin(P);
return std::forward<cpx<L>>(A); }

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>>
                           >::type,
                           R>::value
                           &&
!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>>
                           >::type,
                           L>::value,
                           cpx<R>
                           >::type&&
operator^(cpx<L> && A, cpx<R> && B)
{L r = fabs(A);
 if (r == 0) return std::forward<cpx<R>>(B);
 r = std::log(r);
L p = cpx4::phi(A);
R W = std::exp(r * B.x - p * B.y);
R P = p * B.x + r * B.y;
int n = P / (2*PI);
P = P - n * (2*PI);
B.x = W*cos(P);
B.y = W*sin(P);
return std::forward<cpx<R>>(B); }

```

```

template<typename L, typename R>
typename
enable_if<is_arithmetic<typename decay<decltype(declval<L>())>::value,
           cpx<typename decay<decltype(declval<L>() * declval<R>())>::type>>::type
operator^(L const& A, cpx<R> const& B)
{using S = typename decay<decltype(declval<L>() * declval<R>())>::type ; ;
 L r = std::fabs(A) ; if (r == 0) return cpx<S>(0,0) ; ;
 r = std::log(r) ; ;
 S W = std::exp(r * B.x) ; ;
 S P = r * B.y ; ;
 int n = P / (2*PI) ; ;
 P = P - n * (2*PI) ; ;
 return cpx<S>(W*cos(P), W*sin(P)) ; }

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>() * declval<R>())>::type,
           R>::value
           &&
           is_arithmetic<typename decay<decltype(declval<L>())>::value>>::type
           cpx<L>>::type
operator^(L const& A, cpx<R>&& B)
{L r = std::fabs(A) ; if (r == 0) return cpx<L>(0,0) ; ;
 r = std::log(r) ; ;
 L W = std::exp(r * B.x) ; ;
 L P = r * B.y ; ;
 int n = P / (2*PI) ; ;
 P = P - n * (2*PI) ; ;
 return cpx<L>(W*cos(P), W*sin(P)) ; }

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>() * declval<R>())>::type,
           R>::value
           &&
           is_arithmetic<typename decay<decltype(declval<L>())>::value>>::type
           cpx<R>>::type&&
operator^(L const& A, cpx<R>&& B)
{L r = std::fabs(A) ; if (r == 0) return std::forward<cpx<R>>(B) ; ;
 r = std::log(r) ; ;
 R W = std::exp(r * B.x) ; ;
 R P = r * B.y ; ;
 int n = P / (2*PI) ; ;
 P = P - n * (2*PI) ; ;
 B.x = W*cos(P) ; B.y = W*sin(P) ; ;
 return std::forward<cpx<R>>(B) ; }

```

```

template<typename L, typename R>
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>>::value,
cpx<typename decay<decltype(declval<L>() * declval<R>())>>::type>
>::type
operator^(cpx<L> const& A, R const& B)
{using S = typename decay<decltype(declval<L>() * declval<R>())>>::type ;
L r = fabs(A) ; if (r == 0) return cpx<S>(0,0) ;
r = std::log(r) ;
L p = cpx4::phi(A) ;
S W = std::exp(r * B) ; S P = p * B ;
int n = P / (2*PI) ;
P = P - n * (2*PI) ;
return cpx<S>(W*cos(P), W*sin(P)) ; }

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>() * declval<R>())>>::type,
L>>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>>::value, cpx<R> >>::type
operator^(cpx<L> && A, R const& B)
{L r = fabs(A) ; if (r == 0) return cpx<R>(0,0) ;
r = std::log(r) ;
L p = cpx4::phi(A) ;
R W = std::exp(r * B) ;
R P = p * B ;
int n = P / (2*PI) ;
P = P - n * (2*PI) ;
return cpx<R>(W*cos(P), W*sin(P)) ; }

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>() * declval<R>())>>::type,
L>>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>>::value, cpx<L>
>>::type&&
operator^(cpx<L> && A, R const& B)
{L r = fabs(A) ;
if (r < lm2) return std::forward<cpx<L>>(A) ;
r = std::log(r) ;
L p = cpx4::phi(A) ;
L W = std::exp(r * B) ;
L P = p * B ;
int n = P / (2*PI) ;
P = P - n * (2*PI) ;
A.x = W*cos(P) ; A.y = W*sin(P) ;
return std::forward<cpx<L>>(A) ; }

```

```

template<typename L, typename R>
    cpx<typename decay<decltype(declval<L>()-
                                declval<R>())>::type>
operator-(cpx<L> const& A, cpx<R> const& B)
{using S = typename decay<decltype(declval<L>()-
                                declval<R>())>::type ; ;
 return cpx<S>(A.x-B.x, A.y-B.y) ; }

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>()-
                                declval<R>())>::type,
R>::value,
cpx<R>
>::type&&
operator-(cpx<L> const& A, cpx<R> && B)
{B.x = A.x-B.x ; B.y = A.y-B.y ; ;
 return std::forward<cpx<R>>(B) ; }

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>()-
                                declval<R>())>::type,
R>::value,
cpx<L>
>::type
operator-(cpx<L> const& A, cpx<R> && B)
{cpx<L> Z(A.x-B.x,A.y-B.y) ; return Z ; }

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>()-
                                declval<R>())>::type,
L>::value,
cpx<L>
>::type&&
operator-(cpx<L> && A, cpx<R> const& B)
{A.x == B.x ; A.y == B.y ; return std::forward<cpx<L>>(A) ; }

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>()-
                                declval<R>())>::type,
L>::value,
cpx<R>
>::type
operator-(cpx<L> && A, cpx<R> const& B)
{cpx<R> Z(A.x-B.x,A.y-B.y) ; return Z ; }

```

```

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>())-
                           declval<R>() )>::type,
L>::value,
cpx<L>
>::type&&
operator-(cpx<L> && A, cpx<R> && B)
{A.x -= B.x ; A.y -= B.y ; return std::forward<cpx<L>>(A) ; }

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>())-
                           declval<R>() )>::type,
R>::value
&&
!is_same<typename decay<decltype(declval<L>())-
                           declval<R>() )>::type,
L>::value,
cpx<R>
>::type&&
operator-(cpx<L> && A, cpx<R> && B)
{B.x = A.x-B.x ; B.y = A.y-B.y
 return std::forward<cpx<R>>(B) ; }

template<typename L, typename R>
typename
enable_if<is_arithmetic<typename decay<decltype(declval<L>())>::type
cpx<typename decay<decltype(declval<L>())-
                           declval<R>() )>::type>
>::type
operator-(L const& A, cpx<R> const& B)
{using S = typename decay<decltype(declval<L>())-
                           declval<R>() )>::type ;
return cpx<S>(A-B.x, -B.y) ; }

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>())-
                           declval<R>() )>::type,
R>::value
&&
is_arithmetic<typename decay<decltype(declval<L>())>::type
>::value,
cpx<L>
>::type
operator-(L const& A, cpx<R>&& B)
{return cpx<L>(A-B.x,-B.y) ; }

```

```

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>())-
                           declval<R>() )>
                           >::type,
R>::value
&&
is_arithmetic<typename decay<decltype(declval<L>())>
                           >::type
                           >::value,
cpx<R>
>::type&&
operator-(L const& A, cpx<R>&& B)
{B.x = A-B.x; B.y = -B.y ; return std::forward<cpx<R>>(B); }

template<typename L, typename R>
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>
                           >::type
                           >::value,
cpx<typename decay<decltype(declval<L>())-
                           declval<R>()>>::type>
                           >::type
operator-(cpx<L> const& A, R const& B)
{using S = typename decay<decltype(declval<L>())-
                           declval<R>()>::type ; ;
return cpx<S>(A.x-B, A.y) ; }

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>())-
                           declval<R>() )>
                           >::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>
                           >::type
                           >::value,
cpx<R>
>::type
operator-(cpx<L> && A, R const& B)
{return cpx<R>(A.x-B,A.y) ; }

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>())-
                           declval<R>() )>
                           >::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>
                           >::type
                           >::value,
cpx<L>
>::type&&
operator-(cpx<L> && A, R const& B)
{A.x -= B ; return std::forward<cpx<L>>(A); }

```

```

template<typename L, typename R>
    cpx<R>&
operator=(cpx<L>& A, cpx<R>& B)
{A.x -= B.x ; A.y -= B.y ; return B ;}

template<typename L, typename R>
    cpx<R>&&
operator=(cpx<L>& A, cpx<R> && B)
{A.x -= B.x ; A.y -= B.y ; return std::forward<cpx<R>>(B) ;}

template<typename L, typename R>
    typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>::type
>::value,
        R>::type&
operator=(cpx<L>& A, R& B)
{A.x -= B ; return B ;}

template<typename L, typename R>
    R&&
operator=(cpx<L>& A, R && B)
{A.x -= B ; return std::forward<R>(B) ;}

template<typename L, typename R>
    cpx<R>&
operator=(L& A, cpx<R>& B)
{A -= L(B.x) ; return B ;}

template<typename L, typename R>
    cpx<R>&&
operator=(L& A, cpx<R>&& B)
{A -= L(B.x) ; return std::forward<cpx<R>>(B) ;}

template<typename L, typename R>
    cpx<typename decay<decltype(declval<L>() * declval<R>())>::type>
operator*(cpx<L> const& A, cpx<R> const& B)
{using S = typename decay<decltype(declval<L>() * declval<R>())>::type ;
return cpx<S>(A.x*B.x-A.y*B.y, A.x*B.y+A.y*B.x) ;}

template<typename L, typename R>
    typename
enable_if<is_same<typename decay<decltype(declval<L>() * declval<R>())>::type,
        R>::value,
        cpx<R>
>::type&&
operator*(cpx<L> const& A, cpx<R> && B)
{R tm = A.x*B.x-A.y*B.y; B.y = A.x*B.y+A.y*B.x ; B.x = tm;
return std::forward<cpx<R>>(B) ;}

```

```

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>>
                           >::type,
R>::value,
cpx<L>
>::type
operator*(cpx<L> const& A, cpx<R> && B)
{cpx<L> Z(A.x*B.x-A.y*B.y, A.x*B.y+A.y*B.x) ; return Z ;}

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>>
                           >::type,
L>::value,
cpx<L>
>::type&&
operator*(cpx<L> && A, cpx<R> const& B)
{L tm = A.x*B.x-A.y*B.y; A.y = A.x*B.y+A.y*B.x; A.x = tm ;
 return std::forward<cpx<L>>(A) ;}

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>>
                           >::type,
L>::value,
cpx<R>
>::type
operator*(cpx<L> && A, cpx<R> const& B)
{cpx<R> Z(A.x*B.x-A.y*B.y, A.x*B.y+A.y*B.x) ; return Z ;}

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>>
                           >::type,
L>::value,
cpx<L>
>::type&&
operator*(cpx<L> && A, cpx<R> && B)
{L tm = A.x*B.x-A.y*B.y; A.y = A.x*B.y+A.y*B.x; A.x = tm ;
 return std::forward<cpx<L>>(A) ;}

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>>
                           >::type,
R>::value
&&
!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>>
                           >::type,
L>::value, cpx<R>
>::type&&
operator*(cpx<L> && A, cpx<R> && B)
{R tm = A.x*B.x-A.y*B.y; B.y = A.x*B.y+A.y*B.x; B.x = tm ;
 return std::forward<cpx<R>>(B) ;}

```

```

template<typename L, typename R>
typename
enable_if<is_arithmetic<typename decay<decltype(declval<L>())>::value,
           cpx<typename decay<decltype(declval<L>() * declval<R>())>::type>>::type
operator*(L const& A, cpx<R> const& B)
{using S = typename decay<decltype(declval<L>() * declval<R>())>::type ; ;
return cpx<S>(A*B.x, A*B.y) ; }

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>() * declval<R>())>::type,
           R>::value && is_arithmetic<typename decay<decltype(declval<L>())>::value>::type>::value,
           cpx<L>>::type
operator*(L const& A, cpx<R>&& B)
{return cpx<L>(A*B.x, A*B.y) ; }

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>() * declval<R>())>::type,
           R>::value && is_arithmetic<typename decay<decltype(declval<L>())>::value>::type>::value,
           cpx<R>>::type&&
operator*(L const& A, cpx<R>&& B)
{B.x *= A ; B.y *= A ; return std::forward<cpx<R>>(B) ; }

template<typename L, typename R>
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>::value,
           cpx<typename decay<decltype(declval<L>() * declval<R>())>::type>>::type>::value
operator*(cpx<L> const& A, R const& B)
{using S = typename decay<decltype(declval<L>() * declval<R>())>::type ; ;
return cpx<S>(A.x*B, A.y*B) ; }

```

```

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>
                           >::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>::type
                           >::value
                           ,
cpx<R>
>::type
operator*(cpx<L> && A, R const& B)
{return cpx<R>(A.x*B,A.y*B) ;}

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>())*>
                           declval<R>()>
                           >::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>::type
                           >::value
                           ,
cpx<L>
>::type&&
operator*(cpx<L> && A, R const& B)
{A.x *= B ; A.y *= B ; return std::forward<cpx<L>>(A); }

template<typename L, typename R>
cpx<R>&
operator*=(cpx<L>& A, cpx<R>& B)
{L tm = A.x*B.x-A.y*B.y; A.y = A.x*B.y+A.y*B.x; A.x = tm ;
return B ;}

template<typename L, typename R>
cpx<R>&&
operator*=(cpx<L>& A, cpx<R> && B)
{L tm = A.x*B.x-A.y*B.y; A.y = A.x*B.y+A.y*B.x; A.x = tm ;
return std::forward<cpx<R>>(B) ;}

template<typename L, typename R>
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>::type
                           >::value
                           ,
R>::type&
operator*=(cpx<L>& A, R& B)
{A.x *= B ; A.y *= B ; return B ;}

template<typename L, typename R>
R&&
operator*=(cpx<L>& A, R && B)
{A.x *= B ; A.y *= B ; return std::forward<R>(B) ;}

template<typename L, typename R>
cpx<R>&
operator*=(L& A, cpx<R>& B)
{A *= L(B.x) ; return B ;}

```

```

template<typename L, typename R>
    cpx<R>&&
operator*(L& A, cpx<R>&& B)
{A *= L(B.x) ; return std::forward<cpx<R>>(B);}

template<typename L, typename R>
    cpx<typename decay<decltype(declval<L>() +
                                declval<R>())>::type>
operator+(cpx<L> const& A, cpx<R> const& B)
{using S = typename decay<decltype(declval<L>() +
                                declval<R>())>::type ;
return cpx<S>(A.x+B.x, A.y+B.y) ;}

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>() +
                                declval<R>())>::type,
R>::value,
cpx<R>
>::type&&
operator+(cpx<L> const& A, cpx<R> && B)
{B.x += A.x ; B.y += A.y ; return std::forward<cpx<R>>(B);}

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>() +
                                declval<R>())>::type,
R>::value,
cpx<L>
>::type
operator+(cpx<L> const& A, cpx<R> && B)
{cpx<L> Z(A.x+B.x,A.y+B.y) ; return Z} ;}

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>() +
                                declval<R>())>::type,
L>::value,
cpx<L>
>::type&&
operator+(cpx<L> && A, cpx<R> const& B)
{A.x += B.x ; A.y += B.y ; return std::forward<cpx<L>>(A);}

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>() +
                                declval<R>())>::type,
L>::value,
cpx<R>
>::type
operator+(cpx<L> && A, cpx<R> const& B)
{cpx<R> Z(A.x+B.x,A.y+B.y) ; return Z} ;}

```

```

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>() +
                                         declval<R>()) >>
                           >::type,
                           L>::value,
                           cpx<L>
                           >::type&&
operator+(cpx<L> && A, cpx<R> && B)
{A.x += B.x ; A.y += B.y ; return std::forward<cpx<L>>(A) ;}

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>() +
                                         declval<R>()) >>
                           >::type,
                           R>::value
                           &&
                           !is_same<typename decay<decltype(declval<L>() +
                                         declval<R>()) >>
                           >::type,
                           L>::value,
                           cpx<R>
                           >::type&&
operator+(cpx<L> && A, cpx<R> && B)
{B.x += A.x ; B.y += A.y ; return std::forward<cpx<R>>(B) ;}

template<typename L, typename R>
typename
enable_if<is_arithmetic<typename decay<decltype(declval<L>()) >>
                           >::type
                           >::value,
                           cpx<typename decay<decltype(declval<L>() +
                                         declval<R>())>>::type>
                           >::type
operator+(L const& A, cpx<R> const& B)
{using S = typename decay<decltype(declval<L>() +
                                         declval<R>())>>::type ; ;
return cpx<S>(A+B.x, B.y) ; ; }

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>() +
                                         declval<R>()) >>
                           >::type,
                           R>::value
                           &&
                           is_arithmetic<typename decay<decltype(declval<L>()) >>
                           >::type
                           >::value,
                           cpx<L>
                           >::type
operator+(L const& A, cpx<R>&& B)
{return cpx<L>(A+B.x, B.y) ; ; }

```

```

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>()) +
                           declval<R>() )>::type,
R>::value
&&
is_arithmetic<typename decay<decltype(declval<L>())>::type
>::value,
cpx<R>
>::type&&
operator+(L const& A, cpx<R>&& B)
{B.x += A ; return std::forward<cpx<R>>(B); }

template<typename L, typename R>
typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>::type
>::value,
cpx<typename decay<decltype(declval<L>()) +
                           declval<R>() )>::type>
>::type
operator+(cpx<L> const& A, R const& B)
{using S = typename decay<decltype(declval<L>()) +
                           declval<R>() )>::type ;
return cpx<S>(A.x+B, A.y) ; }

template<typename L, typename R>
typename
enable_if<!is_same<typename decay<decltype(declval<L>()) +
                           declval<R>() )>::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>::type
>::value,
cpx<R>
>::type
operator+(cpx<L> && A, R const& B)
{return cpx<R>(A.x+B,A.y) ; }

template<typename L, typename R>
typename
enable_if<is_same<typename decay<decltype(declval<L>()) +
                           declval<R>() )>::type,
L>::value
&&
is_arithmetic<typename decay<decltype(declval<R>())>::type
>::value,
cpx<L>
>::type&&
operator+(cpx<L> && A, R const& B)
{A.x += B ; return std::forward<cpx<L>>(A); }

```

```

template<typename L, typename R>
    cpx<R>&
operator+=(cpx<L>& A, cpx<R>& B)
{A.x += B.x ; A.y += B.y ; return B ;}

template<typename L, typename R>
    cpx<R>&&
operator+=(cpx<L>& A, cpx<R> && B)
{A.x += B.x ; A.y += B.y ; return std::forward<cpx<R>>(B) ;}

template<typename L, typename R>
    typename
enable_if<is_arithmetic<typename decay<decltype(declval<R>())>::type
>::value,
R>::type&
operator+=(cpx<L>& A, R& B)
{A.x += B ; return B ;}

template<typename L, typename R>
    R&&
operator+=(cpx<L>& A, R && B)
{A.x += B ; return std::forward<R>(B) ;}

template<typename L, typename R>
    cpx<R>&
operator+=(L& A, cpx<R>& B)
{A += L(B.x) ; return B ;}

template<typename L, typename R>
    cpx<R>&&
operator+=(L& A, cpx<R>&& B)
{A += L(B.x) ; return std::forward<cpx<R>>(B) ;}

```

```
// section : NXV4 package
// code    : CPX class -----
//           supports: move semantics
//                         operators + - * / ^   L,R-val context
//                         += -= *= /=
//                         fabs()
//                         phi () [ 0 ... 2.pi] o/p
//                         phi2() [-pi ... +pi] o/p
//                         ~ - unary operators L,R-val
//                         cpx<arithmetic> i.e. int
//                         double
//                         long double
//                         type_casts between the types
//                         cout << overload
//
//                         TOTAL = 27+504 instantiated op/f's
// -----
// author  : M. Dima
//           NXV4 Solutions
//
// date    : CPX4/v1.0.alfa   / Sat Dec  3 17:13:57 CET 2016
//           CPX4/v2.0.alfa   / Fri Oct 20 17:13:57 CET 2017
// -----
// ****
// 
// using dblx =      double ; ;
// using real = long double ; ;
// -----
// 
template class cpx<int > ; ;
template class cpx<dbl> ; ;
template class cpx<real> ; ;

template cpx<int >::cpx(cpx<dbl> const&) ; ;
template cpx<int >::cpx(cpx<real> const&) ; ;

template cpx<dbl>::cpx(cpx<int > const&) ; ;
template cpx<dbl>::cpx(cpx<real> const&) ; ;

template cpx<real>::cpx(cpx<int > const&) ; ;
template cpx<real>::cpx(cpx<dbl> const&) ; ;

template cpx<dbl>& cpx<int >::operator=(cpx<dbl> const&) ; ;
template cpx<real>& cpx<int >::operator=(cpx<real> const&) ; ;
template cpx<int >& cpx<dbl>::operator=(cpx<int > const&) ; ;
template cpx<real>& cpx<dbl>::operator=(cpx<real> const&) ; ;
template cpx<int >& cpx<real>::operator=(cpx<int > const&) ; ;
template cpx<dbl>& cpx<real>::operator=(cpx<dbl> const&) ; ;

template cpx<int > operator-(cpx<int > const&) ; ;
template cpx<dbl> operator-(cpx<dbl> const&) ; ;
```



```

template cpx<real> operator^(cpx<real>const&, dblx const&) ;
template cpx<real> operator^(cpx<real>const&, real const&) ;

// -----
template cpx<int >&& operator^( int const&, cpx<int > &&) ;
template cpx<dbl> operator^( dblx const&, cpx<int > &&) ;
template cpx<real> operator^( real const&, cpx<int > &&) ;

template cpx<dbl>&& operator^( int const&, cpx<dbl> &&) ;
template cpx<dbl>&& operator^( dblx const&, cpx<dbl> &&) ;
template cpx<real> operator^( real const&, cpx<dbl> &&) ;

template cpx<real>&& operator^( int const&, cpx<real> &&) ;
template cpx<real>&& operator^( dblx const&, cpx<real> &&) ;
template cpx<real>&& operator^( real const&, cpx<real> &&) ;

template cpx<int >&& operator^(cpx<int > &&, int const&) ;
template cpx<dbl> operator^(cpx<int > &&, dblx const&) ;
template cpx<real> operator^(cpx<int > &&, real const&) ;

template cpx<dbl>&& operator^(cpx<dbl> &&, int const&) ;
template cpx<dbl>&& operator^(cpx<dbl> &&, dblx const&) ;
template cpx<real> operator^(cpx<dbl> &&, real const&) ;

template cpx<real>&& operator^(cpx<real> &&, int const&) ;
template cpx<real>&& operator^(cpx<real> &&, dblx const&) ;
template cpx<real>&& operator^(cpx<real> &&, real const&) ;

// -----
// OPERATOR: +
template cpx<int > operator+(cpx<int > const&, cpx<int > const&) ;
template cpx<dbl> operator+(cpx<int > const&, cpx<dbl> const&) ;
template cpx<real> operator+(cpx<int > const&, cpx<real> const&) ;

template cpx<dbl> operator+(cpx<dbl> const&, cpx<int > const&) ;
template cpx<dbl> operator+(cpx<dbl> const&, cpx<dbl> const&) ;
template cpx<real> operator+(cpx<dbl> const&, cpx<real> const&) ;

template cpx<real> operator+(cpx<real> const&, cpx<int > const&) ;
template cpx<real> operator+(cpx<real> const&, cpx<dbl> const&) ;
template cpx<real> operator+(cpx<real> const&, cpx<real> const&) ;

template cpx<int >&& operator+(cpx<int > &&, cpx<int > const&) ;
template cpx<dbl> operator+(cpx<int > &&, cpx<dbl> const&) ;
template cpx<real> operator+(cpx<int > &&, cpx<real> const&) ;

template cpx<dbl>&& operator+(cpx<dbl> &&, cpx<int > const&) ;
template cpx<dbl>&& operator+(cpx<dbl> &&, cpx<dbl> const&) ;
template cpx<real> operator+(cpx<dbl> &&, cpx<real> const&) ;

template cpx<real>&& operator+(cpx<real> &&, cpx<int > const&) ;
template cpx<real>&& operator+(cpx<real> &&, cpx<dbl> const&) ;
template cpx<real>&& operator+(cpx<real> &&, cpx<real> const&) ;

template cpx<int >&& operator+(cpx<int > const&, cpx<int > &&) ;
template cpx<dbl>&& operator+(cpx<int > const&, cpx<dbl> &&) ;
template cpx<real>&& operator+(cpx<int > const&, cpx<real> &&) ;

template cpx<dbl> operator+(cpx<dbl> const&, cpx<int > &&) ;
template cpx<dbl> operator+(cpx<dbl> const&, cpx<dbl> &&) ;

```



```

template cpx<real>&& operator+(cpx<real>      &&,      int   const&) ;
template cpx<real>&& operator+(cpx<real>      &&,      dblx  const&) ;
template cpx<real>&& operator+(cpx<real>      &&,      real   const&) ;

// -----
// OPERATOR: +=

template cpx<int >& operator+=(cpx<int >&, cpx<int >& ) ;
template cpx<int >& operator+=(cpx<dbl>&, cpx<int >& ) ;
template cpx<int >& operator+=(cpx<real>&, cpx<int >& ) ;

template cpx<dbl>& operator+=(cpx<int >&, cpx<dbl>& ) ;
template cpx<dbl>& operator+=(cpx<dbl>&, cpx<dbl>& ) ;
template cpx<dbl>& operator+=(cpx<real>&, cpx<dbl>& ) ;

template cpx<real>& operator+=(cpx<int >&, cpx<real>& ) ;
template cpx<real>& operator+=(cpx<dbl>&, cpx<real>& ) ;
template cpx<real>& operator+=(cpx<real>&, cpx<real>& ) ;

template cpx<int >&& operator+=(cpx<int >&, cpx<int > &&) ;
template cpx<int >&& operator+=(cpx<dbl>&, cpx<int > &&) ;
template cpx<int >&& operator+=(cpx<real>&, cpx<int > &&) ;

template cpx<dbl>&& operator+=(cpx<int >&, cpx<dbl> &&) ;
template cpx<dbl>&& operator+=(cpx<dbl>&, cpx<dbl> &&) ;
template cpx<dbl>&& operator+=(cpx<real>&, cpx<dbl> &&) ;

template cpx<real>&& operator+=(cpx<int >&, cpx<real> &&) ;
template cpx<real>&& operator+=(cpx<dbl>&, cpx<real> &&) ;
template cpx<real>&& operator+=(cpx<real>&, cpx<real> &&) ;

template int & operator+<int ,int >(cpx<int >&, int &) ;
template int & operator+<dbl, int >(cpx<dbl>&, int &) ;
template int & operator+<real,int >(cpx<real>&, int &) ;

template dblx& operator+<int ,dbl>(cpx<int >&, dblx&) ;
template dblx& operator+<dbl, dbl>(cpx<dbl>&, dblx&) ;
template dblx& operator+<real, dbl>(cpx<real>&, dblx&) ;

template real& operator+<int ,real>(cpx<int >&, real&) ;
template real& operator+<dbl, real>(cpx<dbl>&, real&) ;
template real& operator+<real, real>(cpx<real>&, real&) ;

template int && operator+=(cpx<int >&, int && ) ;
template int && operator+=(cpx<dbl>&, int && ) ;
template int && operator+=(cpx<real>&, int && ) ;

template dblx&& operator+=(cpx<int >&, dblx&& ) ;
template dblx&& operator+=(cpx<dbl>&, dblx&& ) ;
template dblx&& operator+=(cpx<real>&, dblx&& ) ;

template real&& operator+=(cpx<int >&, real&& ) ;
template real&& operator+=(cpx<dbl>&, real&& ) ;
template real&& operator+=(cpx<real>&, real&& ) ;

template cpx<int >& operator+=( int &, cpx<int >& ) ;
template cpx<int >& operator+=( dblx&, cpx<int >& ) ;
template cpx<int >& operator+=( real&, cpx<int >& ) ;

template cpx<dbl>& operator+=( int &, cpx<dbl>& ) ;
template cpx<dbl>& operator+=( dblx&, cpx<dbl>& ) ;
template cpx<dbl>& operator+=( real&, cpx<dbl>& ) ;

template cpx<real>& operator+=( int &, cpx<real>& ) ;

```



```

template cpx<real>&& operator-(cpx<real>      &&, cpx<int >      &&) ;
template cpx<real>&& operator-(cpx<real>      &&, cpx<dbl>      &&) ;
template cpx<real>&& operator-(cpx<real>      &&, cpx<real>      &&) ;

// -----
template cpx<int >    operator-(    int    const&, cpx<int >const&) ;
template cpx<dbl>    operator-(    dblx   const&, cpx<int >const&) ;
template cpx<real>   operator-(    real    const&, cpx<int >const&) ;

template cpx<dbl>    operator-(    int    const&, cpx<dbl>const&) ;
template cpx<dbl>    operator-(    dblx   const&, cpx<dbl>const&) ;
template cpx<real>   operator-(    real    const&, cpx<dbl>const&) ;

template cpx<real>   operator-(    int    const&, cpx<real>const&) ;
template cpx<real>   operator-(    dblx   const&, cpx<real>const&) ;
template cpx<real>   operator-(    real    const&, cpx<real>const&) ;

template cpx<int >   operator-(cpx<int >const&,      int    const&) ;
template cpx<dbl>    operator-(cpx<int >const&,      dblx   const&) ;
template cpx<real>   operator-(cpx<int >const&,      real    const&) ;

template cpx<dbl>    operator-(cpx<dbl>const&,      int    const&) ;
template cpx<dbl>    operator-(cpx<dbl>const&,      dblx   const&) ;
template cpx<real>   operator-(cpx<dbl>const&,      real    const&) ;

template cpx<real>   operator-(cpx<real>const&,      int    const&) ;
template cpx<real>   operator-(cpx<real>const&,      dblx   const&) ;
template cpx<real>   operator-(cpx<real>const&,      real    const&) ;

// -----
template cpx<int >&& operator-(    int    const&, cpx<int >      &&) ;
template cpx<dbl> && operator-(    dblx   const&, cpx<int >      &&) ;
template cpx<real> && operator-(    real    const&, cpx<int >      &&) ;

template cpx<dbl>&& operator-(    int    const&, cpx<dbl>      &&) ;
template cpx<dbl>&& operator-(    dblx   const&, cpx<dbl>      &&) ;
template cpx<real> && operator-(    real    const&, cpx<dbl>      &&) ;

template cpx<real>&& operator-(    int    const&, cpx<real>      &&) ;
template cpx<real>&& operator-(    dblx   const&, cpx<real>      &&) ;
template cpx<real>&& operator-(    real    const&, cpx<real>      &&) ;

template cpx<int >&& operator-(cpx<int >      &&,      int    const&) ;
template cpx<dbl> && operator-(cpx<int >      &&,      dblx   const&) ;
template cpx<real> && operator-(cpx<int >      &&,      real    const&) ;

template cpx<dbl>&& operator-(cpx<dbl>      &&,      int    const&) ;
template cpx<dbl>&& operator-(cpx<dbl>      &&,      dblx   const&) ;
template cpx<real> && operator-(cpx<dbl>      &&,      real    const&) ;

template cpx<real>&& operator-(cpx<real>      &&,      int    const&) ;
template cpx<real>&& operator-(cpx<real>      &&,      dblx   const&) ;
template cpx<real>&& operator-(cpx<real>      &&,      real    const&) ;

// -----
// OPERATOR: ==
template cpx<int >& operator==(cpx<int >&, cpx<int >&) ;
template cpx<int >& operator==(cpx<dbl>&, cpx<int >&) ;
template cpx<int >& operator==(cpx<real>&, cpx<int >&) ;

template cpx<dbl>& operator==(cpx<int >&, cpx<dbl>&) ;

```

```

template cpx<dbl>& operator==(cpx<dbl>&, cpx<dbl>&) ;
template cpx<dbl>& operator==(cpx<real>&, cpx<dbl>&) ;

template cpx<real>& operator==(cpx<int >&, cpx<real>&) ;
template cpx<real>& operator==(cpx<dbl>&, cpx<real>&) ;
template cpx<real>& operator==(cpx<real>&, cpx<real>&) ;

template cpx<int >&& operator==(cpx<int >&, cpx<int > &&) ;
template cpx<int >&& operator==(cpx<dbl>&, cpx<int > &&) ;
template cpx<int >&& operator==(cpx<real>&, cpx<int > &&) ;

template cpx<dbl>&& operator==(cpx<int >&, cpx<dbl> &&) ;
template cpx<dbl>&& operator==(cpx<dbl>&, cpx<dbl> &&) ;
template cpx<dbl>&& operator==(cpx<real>&, cpx<dbl> &&) ;

template cpx<real>&& operator==(cpx<int >&, cpx<real> &&) ;
template cpx<real>&& operator==(cpx<dbl>&, cpx<real> &&) ;
template cpx<real>&& operator==(cpx<real>&, cpx<real> &&) ;

template int & operator==<int ,int >(cpx<int >&, int &) ;
template int & operator==<dbl, int >(cpx<dbl>&, int &) ;
template int & operator==<real, int >(cpx<real>&, int &) ;

template dblx& operator==<int ,dbl>(cpx<int >&, dblx&) ;
template dblx& operator==<dbl, dbl>(cpx<dbl>&, dblx&) ;
template dblx& operator==<real, dbl>(cpx<real>&, dblx&) ;

template real& operator==<int ,real>(cpx<int >&, real&) ;
template real& operator==<dbl, real>(cpx<dbl>&, real&) ;
template real& operator==<real, real>(cpx<real>&, real&) ;

template int && operator==(cpx<int >&, int &&) ;
template int && operator==(cpx<dbl>&, int &&) ;
template int && operator==(cpx<real>&, int &&) ;

template dblx&& operator==(cpx<int >&, dblx&&) ;
template dblx&& operator==(cpx<dbl>&, dblx&&) ;
template dblx&& operator==(cpx<real>&, dblx&&) ;

template real&& operator==(cpx<int >&, real&&) ;
template real&& operator==(cpx<dbl>&, real&&) ;
template real&& operator==(cpx<real>&, real&&) ;

template cpx<int >& operator==( int &, cpx<int >&) ;
template cpx<int >& operator==( dblx&, cpx<int >&) ;
template cpx<int >& operator==( real&, cpx<int >&) ;

template cpx<dbl>& operator==( int &, cpx<dbl>&) ;
template cpx<dbl>& operator==( dblx&, cpx<dbl>&) ;
template cpx<dbl>& operator==( real&, cpx<dbl>&) ;

template cpx<real>& operator==( int &, cpx<real>&) ;
template cpx<real>& operator==( dblx&, cpx<real>&) ;
template cpx<real>& operator==( real&, cpx<real>&) ;

template cpx<int >&& operator==( int &, cpx<int >&&) ;
template cpx<int >&& operator==( dblx&, cpx<int >&&) ;
template cpx<int >&& operator==( real&, cpx<int >&&) ;

template cpx<dbl>&& operator==( int &, cpx<dbl>&&) ;
template cpx<dbl>&& operator==( dblx&, cpx<dbl>&&) ;
template cpx<dbl>&& operator==( real&, cpx<dbl>&&) ;

template cpx<real>&& operator==( int &, cpx<real>&&) ;
template cpx<real>&& operator==( dblx&, cpx<real>&&) ;
template cpx<real>&& operator==( real&, cpx<real>&&) ;

```



```

template cpx<real> operator*( real const&, cpx<real>const&) ;

template cpx<int > operator*(cpx<int >const&, int const&) ;
template cpx<dbl> operator*(cpx<int >const&, dblx const&) ;
template cpx<real> operator*(cpx<int >const&, real const&) ;

template cpx<dbl> operator*(cpx<dbl>const&, int const&) ;
template cpx<dbl> operator*(cpx<dbl>const&, dblx const&) ;
template cpx<real> operator*(cpx<dbl>const&, real const&) ;

template cpx<real> operator*(cpx<real>const&, int const&) ;
template cpx<real> operator*(cpx<real>const&, dblx const&) ;
template cpx<real> operator*(cpx<real>const&, real const&) ;

// -----
template cpx<int >&& operator*( int const&, cpx<int > &&) ;
template cpx<dbl> operator*( dblx const&, cpx<int > &&) ;
template cpx<real> operator*( real const&, cpx<int > &&) ;

template cpx<dbl>&& operator*( int const&, cpx<dbl> &&) ;
template cpx<dbl>&& operator*( dblx const&, cpx<dbl> &&) ;
template cpx<real> operator*( real const&, cpx<dbl> &&) ;

template cpx<real>&& operator*( int const&, cpx<real> &&) ;
template cpx<real>&& operator*( dblx const&, cpx<real> &&) ;
template cpx<real>&& operator*( real const&, cpx<real> &&) ;

template cpx<int >&& operator*(cpx<int > &&, int const&) ;
template cpx<dbl> operator*(cpx<int > &&, dblx const&) ;
template cpx<real> operator*(cpx<int > &&, real const&) ;

template cpx<dbl>&& operator*(cpx<dbl> &&, int const&) ;
template cpx<dbl>&& operator*(cpx<dbl> &&, dblx const&) ;
template cpx<real> operator*(cpx<dbl> &&, real const&) ;

template cpx<real>&& operator*(cpx<real> &&, int const&) ;
template cpx<real>&& operator*(cpx<real> &&, dblx const&) ;
template cpx<real>&& operator*(cpx<real> &&, real const&) ;

// -----
// OPERATOR: *=

template cpx<int >& operator*=(cpx<int >&, cpx<int >& ) ;
template cpx<int >& operator*=(cpx<dbl>&, cpx<int >& ) ;
template cpx<int >& operator*=(cpx<real>&, cpx<int >& ) ;

template cpx<dbl>& operator*=(cpx<int >&, cpx<dbl>& ) ;
template cpx<dbl>& operator*=(cpx<dbl>&, cpx<dbl>& ) ;
template cpx<dbl>& operator*=(cpx<real>&, cpx<dbl>& ) ;

template cpx<real>& operator*=(cpx<int >&, cpx<real>& ) ;
template cpx<real>& operator*=(cpx<dbl>&, cpx<real>& ) ;
template cpx<real>& operator*=(cpx<real>&, cpx<real>& ) ;

template cpx<int >&& operator*=(cpx<int >&, cpx<int > &&) ;
template cpx<int >&& operator*=(cpx<dbl>&, cpx<int > &&) ;
template cpx<int >&& operator*=(cpx<real>&, cpx<int > &&) ;

template cpx<dbl>&& operator*=(cpx<int >&, cpx<dbl> &&) ;
template cpx<dbl>&& operator*=(cpx<dbl>&, cpx<dbl> &&) ;
template cpx<dbl>&& operator*=(cpx<real>&, cpx<dbl> &&) ;

template cpx<real>&& operator*=(cpx<int >&, cpx<real> &&) ;
template cpx<real>&& operator*=(cpx<dbl>&, cpx<real> &&) ;

```

```

template cpx<real>&& operator*=(cpx<real>&, cpx<real> &&) ;

template int & operator*=<int ,int >(cpx<int >&, int &) ;
template int & operator*=<dblx,int >(cpx<dbl>&, int &) ;
template int & operator*=<real,int >(cpx<real>&, int &) ;

template dblx& operator*=<int ,dbl>(cpx<int >&, dblx&) ;
template dblx& operator*=<dbl,dbl>(cpx<dbl>&, dblx&) ;
template dblx& operator*=<real,dbl>(cpx<real>&, dblx&) ;

template real& operator*=<int ,real>(cpx<int >&, real&) ;
template real& operator*=<dbl,real>(cpx<dbl>&, real&) ;
template real& operator*=<real,real>(cpx<real>&, real&) ;

template int && operator*=(cpx<int >&, int && ) ;
template int && operator*=(cpx<dbl>&, int && ) ;
template int && operator*=(cpx<real>&, int && ) ;

template dblx&& operator*=(cpx<int >&, dblx&& ) ;
template dblx&& operator*=(cpx<dbl>&, dblx&& ) ;
template dblx&& operator*=(cpx<real>&, dblx&& ) ;

template real&& operator*=(cpx<int >&, real&& ) ;
template real&& operator*=(cpx<dbl>&, real&& ) ;
template real&& operator*=(cpx<real>&, real&& ) ;

template cpx<int >& operator*=( int &, cpx<int >& ) ;
template cpx<int >& operator*=( dblx&, cpx<int >& ) ;
template cpx<int >& operator*=( real&, cpx<int >& ) ;

template cpx<dbl>& operator*=( int &, cpx<dbl>& ) ;
template cpx<dbl>& operator*=( dblx&, cpx<dbl>& ) ;
template cpx<dbl>& operator*=( real&, cpx<dbl>& ) ;

template cpx<real>& operator*=( int &, cpx<real>& ) ;
template cpx<real>& operator*=( dblx&, cpx<real>& ) ;
template cpx<real>& operator*=( real&, cpx<real>& ) ;

template cpx<int >&& operator*=( int &, cpx<int >&& ) ;
template cpx<int >&& operator*=( dblx&, cpx<int >&& ) ;
template cpx<int >&& operator*=( real&, cpx<int >&& ) ;

template cpx<dbl>&& operator*=( int &, cpx<dbl>&& ) ;
template cpx<dbl>&& operator*=( dblx&, cpx<dbl>&& ) ;
template cpx<dbl>&& operator*=( real&, cpx<dbl>&& ) ;

template cpx<real>&& operator*=( int &, cpx<real>&& ) ;
template cpx<real>&& operator*=( dblx&, cpx<real>&& ) ;
template cpx<real>&& operator*=( real&, cpx<real>&& ) ;

// -----
// OPERATOR: /

template cpx<int > operator/(cpx<int > const&, cpx<int > const&) ;
template cpx<dbl> operator/(cpx<int > const&, cpx<dbl> const&) ;
template cpx<real> operator/(cpx<int > const&, cpx<real> const&) ;

template cpx<dbl> operator/(cpx<dbl> const&, cpx<int > const&) ;
template cpx<dbl> operator/(cpx<dbl> const&, cpx<dbl> const&) ;
template cpx<dbl> operator/(cpx<dbl> const&, cpx<real> const&) ;

template cpx<real> operator/(cpx<real> const&, cpx<int > const&) ;
template cpx<real> operator/(cpx<real> const&, cpx<dbl> const&) ;
template cpx<real> operator/(cpx<real> const&, cpx<real> const&) ;

```



```

template cpx<int >&& operator/( int const&, cpx<int > &&) ;
template cpx<dbl> operator/( dblx const&, cpx<int > &&) ;
template cpx<real> operator/( real const&, cpx<int > &&) ;

template cpx<dbl>&& operator/( int const&, cpx<dbl> &&) ;
template cpx<dbl>&& operator/( dblx const&, cpx<dbl> &&) ;
template cpx<real> operator/( real const&, cpx<dbl> &&) ;

template cpx<real>&& operator/( int const&, cpx<real> &&) ;
template cpx<real>&& operator/( dblx const&, cpx<real> &&) ;
template cpx<real>&& operator/( real const&, cpx<real> &&) ;

template cpx<int >&& operator/(cpx<int > &&, int const&) ;
template cpx<dbl> operator/(cpx<int > &&, dblx const&) ;
template cpx<real> operator/(cpx<int > &&, real const&) ;

template cpx<dbl>&& operator/(cpx<dbl> &&, int const&) ;
template cpx<dbl>&& operator/(cpx<dbl> &&, dblx const&) ;
template cpx<real> operator/(cpx<dbl> &&, real const&) ;

template cpx<real>&& operator/(cpx<real> &&, int const&) ;
template cpx<real>&& operator/(cpx<real> &&, dblx const&) ;
template cpx<real>&& operator/(cpx<real> &&, real const&) ;

// -----
// OPERATOR: /=

template cpx<int >& operator/=(cpx<int >&, cpx<int >& ) ;
template cpx<int >& operator/=(cpx<dbl>&, cpx<int >& ) ;
template cpx<int >& operator/=(cpx<real>&, cpx<int >& ) ;

template cpx<dbl>& operator/=(cpx<int >&, cpx<dbl>& ) ;
template cpx<dbl>& operator/=(cpx<dbl>&, cpx<dbl>& ) ;
template cpx<dbl>& operator/=(cpx<real>&, cpx<dbl>& ) ;

template cpx<real>& operator/=(cpx<int >&, cpx<real>& ) ;
template cpx<real>& operator/=(cpx<dbl>&, cpx<real>& ) ;
template cpx<real>& operator/=(cpx<real>&, cpx<real>& ) ;

template cpx<int >&& operator/=(cpx<int >&, cpx<int > &&) ;
template cpx<int >&& operator/=(cpx<dbl>&, cpx<int > &&) ;
template cpx<int >&& operator/=(cpx<real>&, cpx<int > &&) ;

template cpx<dbl>&& operator/=(cpx<int >&, cpx<dbl> &&) ;
template cpx<dbl>&& operator/=(cpx<dbl>&, cpx<dbl> &&) ;
template cpx<dbl>&& operator/=(cpx<real>&, cpx<dbl> &&) ;

template cpx<real>&& operator/=(cpx<int >&, cpx<real> &&) ;
template cpx<real>&& operator/=(cpx<dbl>&, cpx<real> &&) ;
template cpx<real>&& operator/=(cpx<real>&, cpx<real> &&) ;

template int & operator/=<int ,int >(cpx<int >&, int &) ;
template int & operator/=<dbl,dbl >(cpx<dbl>&, int &) ;
template int & operator/=<real,real >(cpx<real>&, int &) ;

template dblx& operator/=<int ,dbl >(cpx<int >&, dblx&) ;
template dblx& operator/=<dbl,dbl >(cpx<dbl>&, dblx&) ;
template dblx& operator/=<real,dbl >(cpx<real>&, dblx&) ;

template real& operator/=<int ,real >(cpx<int >&, real&) ;
template real& operator/=<dbl,real >(cpx<dbl>&, real&) ;
template real& operator/=<real,real >(cpx<real>&, real&) ;

template int && operator/=(cpx<int >&, int && ) ;

```

```

template      int && operator/=(cpx<dbl> &, int && ) ;
template      int && operator/=(cpx<real> &, int && ) ;

template      dbl && operator/=(cpx<int > &, dbl && ) ;
template      dbl && operator/=(cpx<dbl> &, dbl && ) ;
template      dbl && operator/=(cpx<real> &, dbl && ) ;

template      real && operator/=(cpx<int > &, real && ) ;
template      real && operator/=(cpx<dbl> &, real && ) ;
template      real && operator/=(cpx<real> &, real && ) ;

template cpx<int >& operator/=( int &, cpx<int >& ) ;
template cpx<int >& operator/=( dbl &, cpx<int >& ) ;
template cpx<int >& operator/=( real &, cpx<int >& ) ;

template cpx<dbl>& operator/=( int &, cpx<dbl>& ) ;
template cpx<dbl>& operator/=( dbl &, cpx<dbl>& ) ;
template cpx<dbl>& operator/=( real &, cpx<dbl>& ) ;

template cpx<real>& operator/=( int &, cpx<real>& ) ;
template cpx<real>& operator/=( dbl &, cpx<real>& ) ;
template cpx<real>& operator/=( real &, cpx<real>& ) ;

template cpx<int >&& operator/=( int &, cpx<int >&& ) ;
template cpx<int >&& operator/=( dbl &, cpx<int >&& ) ;
template cpx<int >&& operator/=( real &, cpx<int >&& ) ;

template cpx<dbl>&& operator/=( int &, cpx<dbl>&& ) ;
template cpx<dbl>&& operator/=( dbl &, cpx<dbl>&& ) ;
template cpx<dbl>&& operator/=( real &, cpx<dbl>&& ) ;

template cpx<real>&& operator/=( int &, cpx<real>&& ) ;
template cpx<real>&& operator/=( dbl &, cpx<real>&& ) ;
template cpx<real>&& operator/=( real &, cpx<real>&& ) ;

// -----
template cpx<int > sqrt(cpx<int > const&) ;
template cpx<dbl> sqrt(cpx<dbl> const&) ;
template cpx<real> sqrt(cpx<real> const&) ;

template cpx<int > exp (cpx<int > const&) ;
template cpx<dbl> exp (cpx<dbl> const&) ;
template cpx<real> exp (cpx<real> const&) ;

template cpx<int > log (cpx<int > const&) ;
template cpx<dbl> log (cpx<dbl> const&) ;
template cpx<real> log (cpx<real> const&) ;

template cpx<int >&& sqrt(cpx<int > &&) ;
template cpx<dbl>&& sqrt(cpx<dbl> &&) ;
template cpx<real>&& sqrt(cpx<real> &&) ;

template cpx<int >&& exp (cpx<int > &&) ;
template cpx<dbl>&& exp (cpx<dbl> &&) ;
template cpx<real>&& exp (cpx<real> &&) ;

template cpx<int >&& log (cpx<int > &&) ;
template cpx<dbl>&& log (cpx<dbl> &&) ;
template cpx<real>&& log (cpx<real> &&) ;

// -----
// -----
template bool operator>(cpx<int > const&, int const&) ;
template bool operator>(cpx<int > const&, dbl const&) ;

```



```
vpath %.hh      include
vpath     ..../CPX/include
vpath %.cc  src
vpath %.ie  src
vpath %.o   obj

all: main
    export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:lib:../CPX/lib; \
        ./test

libs: objects

objects: cpx.hh cpx.cc
    cd obj ; rm *.o ; g++ -std=c++14 \
        -O3 \
        -z muldefs \
        -fopenmp \
        -floop-block \
        -ftree-loop-distribution \
        -floop-parallelize-all \
        -ftree-parallelize-loops=4 \
        -ftree-vectorize \
        -fconcepts \
        -mavx2 \
        -I ../include \
        -I ../../CPX/include \
        -c \
        ../../src/*.cc
    cd obj ; ar rs libcpx4.a *.o
    cd obj ; mv libcpx4* ../lib
    cd obj ; for filename in *.o; \
        do echo mv \"${filename}\" \"${filename}..o/- \
static.u\"; \
        done|/bin/bash
    cd obj ; g++ -std=c++14 \
        -O3 \
        -z muldefs \
        -fopenmp \
        -floop-block \
        -ftree-loop-distribution \
        -floop-parallelize-all \
        -ftree-parallelize-loops=4 \
        -ftree-vectorize \
        -fconcepts \
        -mavx2 \
        -I ../include \
        -I ../../CPX/include \
        -c \
        -fPIC \
        ../../src/*.cc
    cd obj ; g++ -std=c++14 \
        -O3 \
        -z muldefs \
        -fopenmp \
        -fPIC
```

```

        -floop-block          \
        -ftree-loop-distribution \
        -floop-parallelize-all \
        -ftree-parallelize-loops=4 \
        -ftree-vectorize       \
        -fconcepts             \
        -mavx2                 \
        -I ../include           \
        -I ../../CPX/include    \
        -shared                \
        -o                     \
            libcpx4.so *.o
    cd obj ; mv libcpx4.so ../../lib
    cd obj ; for filename in *.*; \
        do echo mv \"\$filename\" \\"$filename//.o/-dynamic.o\";
    done|/bin/bash
    cd obj ; for filename in *.u; \
        do echo mv \\"$filename//.u/\\".o\";
    done|/bin/bash

main: test.cc
    g++ test.cc -std=c++14 \
        -O3 \
        -z muldefs \
        -fopenmp \
        -floop-block \
        -ftree-loop-distribution \
        -floop-parallelize-all \
        -ftree-parallelize-loops=4 \
        -ftree-vectorize \
        -fconcepts \
        -mavx2 \
        -I include \
        -I ./CPX/include \
        -o test \
        -L lib \
        -l cpx4 \
        -L ./CPX/lib \
        -l cpx4

test: test.cc
    g++ test.cc -std=c++14 \
        -O3 \
        -z muldefs \
        -fopenmp \
        -floop-block \
        -ftree-loop-distribution \
        -floop-parallelize-all \
        -ftree-parallelize-loops=4 \
        -ftree-vectorize \
        -fconcepts \
        -mavx2 \
        -I include \
        -I ./CPX/include \
        -o test \
        -L lib \
        -l cpx4 \
        -L ./CPX/lib \
        -l cpx4

```

```
clean:
    rm -fr lib/* ; rm -fr obj/*
    rm -fr test

run:
    export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:lib:../CPX/lib; \
        ./test
```

```

#include <cmath>
#include <string>
#include <sstream>
#include <stdio.h>

#include <iostream>
#include "cpx.hh"
using namespace std;

using namespace cpx4;

int main()
{
    cpx4::cosmetic = 1.0e-2

    using dblx = double
    using real = long double

    double e = exp(1)
    double pi= 3.14159265853

    cpx<int > ii(0,1)
    cpx<dbl> id(0,1)
    cpx<real> ir(0,1)

    int si = 1
    dblx sd = 1
    real sr = 1

    cout << si*ii << endl;
    cout << si*id << endl;
    cout << si*ir << endl;

    cout << sd*ii << endl;
    cout << sd*id << endl;
    cout << sd*ir << endl;

    cout << sr*ii << endl;
    cout << sr*id << endl;
    cout << sr*ir << endl;

    cout << 1/(7+4*id) << endl;

    cout << cpx<int>(9,0)*1 << endl;
    cout << cpx<int>(9,0)*1.0 << endl;
    cout << cpx<int>(9,0)+1.0L << endl;

    cout << sqrt(cpx<real>(1,-0.0000000001)) << endl;
}

```