G++ 2.91.57，cygnus\cygwin-b20\include\g++**std\complext.cc** 完整列表

```
// Member templates for the -*- C++ -*- complex number classes.
// Copyright (C) 1994 Free Software Foundation

// This file is part of the GNU ANSI C++ Library.  This library is free
// software; you can redistribute it and/or modify it under the
// terms of the GNU General Public License as published by the
// Free Software Foundation; either version 2, or (at your option)
// any later version.

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

// You should have received a copy of the GNU General Public License
// along with this library; see the file COPYING.  If not, write to the Free
// Software Foundation, 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.

// As a special exception, if you link this library with files
// compiled with a GNU compiler to produce an executable, this does not cause
// the resulting executable to be covered by the GNU General Public License.
// This exception does not however invalidate any other reasons why
// the executable file might be covered by the GNU General Public License.

// Written by Jason Merrill based upon the specification in the 27 May 1994
// C++ working paper, ANSI document X3J16/94-0098.

#include <complex>

// 以下是各種複數運算

extern "C++" {
template <class FLOAT> complex<FLOAT>
cos (const complex<FLOAT>& x)
{
  return complex<FLOAT> (cos (real (x)) * cosh (imag (x)),
                - sin (real (x)) * sinh (imag (x)));
}

template <class FLOAT> complex<FLOAT>
cosh (const complex<FLOAT>& x)
{
  return complex<FLOAT> (cosh (real (x)) * cos (imag (x)),
                sinh (real (x)) * sin (imag (x)));
}

template <class FLOAT> complex<FLOAT>
exp (const complex<FLOAT>& x)
```

```
{
  return polar (FLOAT (exp (real (x))), imag (x));
}

template <class FLOAT> complex<FLOAT>
log (const complex<FLOAT>& x)
{
  return complex<FLOAT> (log (abs (x)), arg (x));
}

template <class FLOAT> complex<FLOAT>
pow (const complex<FLOAT>& x, const complex<FLOAT>& y)
{
  FLOAT logr = log (abs (x));
  FLOAT t = arg (x);

  return polar (FLOAT (exp (logr * real (y) - imag (y) * t)),
        FLOAT (imag (y) * logr + real (y) * t));
}

template <class FLOAT> complex<FLOAT>
pow (const complex<FLOAT>& x, FLOAT y)
{
  return exp (FLOAT (y) * log (x));
}

template <class FLOAT> complex<FLOAT>
pow (FLOAT x, const complex<FLOAT>& y)
{
  return exp (y * FLOAT (log (x)));
}

template <class FLOAT> complex<FLOAT>
sin (const complex<FLOAT>& x)
{
  return complex<FLOAT> (sin (real (x)) * cosh (imag (x)),
                cos (real (x)) * sinh (imag (x)));
}

template <class FLOAT> complex<FLOAT>
sinh (const complex<FLOAT>& x)
{
  return complex<FLOAT> (sinh (real (x)) * cos (imag (x)),
                cosh (real (x)) * sin (imag (x)));
}

#include <iostream.h>

template <class FLOAT> istream&
```

```
operator >> (istream& is, complex<FLOAT>& x)
{
  FLOAT re, im = 0;              // 虛部預設為 0
  char ch = 0;

  // 以下，ipfx0() 是 ipfx(0) 的最佳化版本，定義於 <iostream.h>。
  // ipfx(n) 用來判斷 istream 的狀態好壞，並跳過空白輸入。
  if (is.ipfx0 ())
    {
      if (is.peek () == '(')     // 看到緩衝區內有個'('
         is >> ch;               //   讀入'('
      is >> re;                  // 讀入複數實部
      if (ch == '(')             // 確實有'('
         {
           is >> ch;             //   讀入下一個字元
           if (ch == ',')        //   如果是個','
              is >> im >> ch;    //     讀入複數虛部
         }
    }
  is.isfx ();                    // isfx() 定義於 <iostream.h>。

  if (ch != 0 && ch != ')')
    is.setstate (ios::failbit);      // 輸入型式有誤
  else if (is.good ())               // 如果輸入格式正確
    x = complex<FLOAT> (re, im);     //   就產生出一個複數

  return is;   // 注意，傳回的是個 istream&，如此才能做連串輸入動作
}

template <class FLOAT> ostream&
operator << (ostream& os, const complex<FLOAT>& x)
{
  return os << '(' << real (x) << ',' << imag (x) << ')';
}

// The code below is adapted from f2c's libF77, and is subject to this
// copyright:

/****************************************************************
Copyright 1990, 1991, 1992, 1993 by AT&T Bell Laboratories and Bellcore.

Permission to use, copy, modify, and distribute this software
and its documentation for any purpose and without fee is hereby
granted, provided that the above copyright notice appear in all
copies and that both that the copyright notice and this
permission notice and warranty disclaimer appear in supporting
documentation, and that the names of AT&T Bell Laboratories or
Bellcore or any of their entities not be used in advertising or
publicity pertaining to distribution of the software without
```

```
specific, written prior permission.

AT&T and Bellcore disclaim all warranties with regard to this
software, including all implied warranties of merchantability
and fitness.  In no event shall AT&T or Bellcore be liable for
any special, indirect or consequential damages or any damages
whatsoever resulting from loss of use, data or profits, whether
in an action of contract, negligence or other tortious action,
arising out of or in connection with the use or performance of
this software.
*****************************************************************/

// 複數除法 division
template <class FLOAT> complex<FLOAT>&
__doadv (complex<FLOAT>* ths, const complex<FLOAT>& y)
{
  FLOAT ar = abs (y.re);
  FLOAT ai = abs (y.im);
  FLOAT nr, ni;
  FLOAT t, d;
  if (ar <= ai)
    {
      t = y.re / y.im;
      d = y.im * (1 + t*t);
      nr = (ths->re * t + ths->im) / d;
      ni = (ths->im * t - ths->re) / d;
    }
  else
    {
      t = y.im / y.re;
      d = y.re * (1 + t*t);
      nr = (ths->re + ths->im * t) / d;
      ni = (ths->im - ths->re * t) / d;
    }
  ths->re = nr;
  ths->im = ni;
  return *ths;
}

// 全域函式，兩複數相除
template <class FLOAT> complex<FLOAT>
operator / (const complex<FLOAT>& x, const complex<FLOAT>& y)
{
  FLOAT ar = abs (real (y));
  FLOAT ai = abs (imag (y));
  FLOAT nr, ni;
  FLOAT t, d;
  if (ar <= ai)
    {
```

```
      t = real (y) / imag (y);
      d = imag (y) * (1 + t*t);
      nr = (real (x) * t + imag (x)) / d;
      ni = (imag (x) * t - real (x)) / d;
    }
  else
    {
      t = imag (y) / real (y);
      d = real (y) * (1 + t*t);
      nr = (real (x) + imag (x) * t) / d;
      ni = (imag (x) - real (x) * t) / d;
    }
  return complex<FLOAT> (nr, ni);
}

// 全域函式，實數除以複數
template <class FLOAT> complex<FLOAT>
operator / (FLOAT x, const complex<FLOAT>& y)
{
  FLOAT ar = abs (real (y));
  FLOAT ai = abs (imag (y));
  FLOAT nr, ni;
  FLOAT t, d;
  if (ar <= ai)
    {
      t = real (y) / imag (y);
      d = imag (y) * (1 + t*t);
      nr = x * t / d;
      ni = -x / d;
    }
  else
    {
      t = imag (y) / real (y);
      d = real (y) * (1 + t*t);
      nr = x / d;
      ni = -x * t / d;
    }
  return complex<FLOAT> (nr, ni);
}

// 全域函式，複數的
template <class FLOAT> complex<FLOAT>
pow (const complex<FLOAT>& xin, int y)
{
  if (y == 0)
    return complex<FLOAT> (1.0);
  complex<FLOAT> r (1.0);
  complex<FLOAT> x (xin);
  if (y < 0)
```

```
    {
      y = -y;
      x = 1/x;
    }
  for (;;)
    {
      if (y & 1)
    r *= x;
      if (y >>= 1)
    x *= x;
      else
    return r;
    }
}

template <class FLOAT> complex<FLOAT>
sqrt (const complex<FLOAT>& x)
{
  FLOAT r = abs (x);
  FLOAT nr, ni;
  if (r == 0.0)
    nr = ni = r;
  else if (real (x) > 0)
    {
      nr = sqrt (0.5 * (r + real (x)));
      ni = imag (x) / nr / 2;
    }
  else
    {
      ni = sqrt (0.5 * (r - real (x)));
      if (imag (x) < 0)
    ni = - ni;
      nr = imag (x) / ni / 2;
    }
  return complex<FLOAT> (nr, ni);
}
} // extern "C++"
```