VisualAge® C++ Professional for AIX®

IBM® Open Class™: Math

Version 5.0

Note!
Before using this information and the product it supports, be sure to read the general information under "Notices" on page v.

Edition Notice

This edition applies to Version 5.0 of IBM VisualAge C++ and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Contents

Notices

Note to U.S. Government Users Restricted Rights -- use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd.
1150 Eglinton Avenue East
Toronto, Ontario M3C 1H7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

® (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. ® Copyright IBM Corp. 1998, 2000. All rights reserved.

Programming Interface Information

Programming interface information is intended to help you create application software using this program.

General-use programming interface allow the customer to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification, and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and Service Marks

The following terms are trademarks of the International Business Machines Corporation in the United States, or other countries, or both:

AIX
AS/400
DB2
CICS
C Set ++
IBM
Network Station
Object Connection
OS/2
OS/390
OS/400
Open Class
Operating System/2
Operating System/400
PowerPC 403
PowerPC 601
PowerPC 603
PowerPC 604
Presentation Manager
RS/6000
S/390
SAA
Systems Application Architechture
TeamConnection
VisualAge
WebSphere
Workplace Shell

Lotus, Lotus Notes, and Domino are trademarks or registered trademarks of the Lotus Development Corporation in the United States, or other countries, or both.

Tivoli Management Environment, TME 10, and Tivoli Module Designer are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

Encina and DCE Encina Lightweight Client are trademarks of Transarc Corporation in the United States, or other countries, or both.

Microsoft, Win32, Windows, Windows NT, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

UNIX is a registered trademark in the U.S. and other countries licensed exclusively through X/Open Company Limited.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks or registered trademarks of Intel Corporation in the United States, or other countries, or both.

C-bus is a registered trademark of Corollary, Inc.

PC Direct is a registered tradmark of Ziff Communicatoins Company and is used by IBM Corporation under license

Other company, product, and service names, which may be denoted by a double asterisk(**), may be trademarks or service marks of others.

Industry Standards

VisualAge C++ Professional for AIX, Version 5.0 supports the following standards:

- The C language is consistent with the International Standard C (ANSI/ISO-IEC 9899–1990 [1992]). This standard has officially replaced American National standard for Information Systems-Programming Language C (X3.159–1989) and is technically equivalent to the ANSI C standard. VisualAge C++ supports the changes adopted into the C Standard by ISO/IEC 9899:1990/Amendment 1:1994.
- The IBM Systems Application Architecture (SAA) C Level 2 language definition.
- The C++ language is consistent with the International Standard for Information Systems-Programming Language C++ (ISO/IEC 14882:1998).
- The ISO/IEC 9945–1:1990/IEEE POSIX 1003.-1990 standard.
- The X/Open Common Applications Environment Specifications, System Interfaces and Headers, Issue 4.

About This Book

The information in this PDF document is also available in the online help.

To find this information, or any topics listed in this document as Related Concepts, Related Tasks, or Related References, simply type the topic title into the search bar in the top frame of your browser in the online help.

For some topics, the suggested references may already be contained in this document. In such cases, there is a cross-reference to the page on which the related topic appears.

Chapter 1. The IBinaryCodedDecimal Class

The IBinaryCodedDecimal class represents exact numerical quantities in business and commercial applications for financial calculations.

The IBinaryCodedDecimal class allows representation of up to 31 significant digits, including integral and fractional parts. The fractional part of a dollar can be represented accurately by two digits following the decimal point. You do not have to use floating-point arithmetic, which is more suitable for scientific and engineering computations. These computations often use numbers much larger than the largest that the IBinaryCodedDecimal object can store.

The same declarations and operators that you use on other data types, such as float, can be applied to IBinaryCodedDecimal objects. You can declare typedefs, arrays, and structures that have IBinaryCodedDecimal objects. You can apply arithmetic, relational, assignment, comma, conditional, equality, logical, primary, and unary operators on the IBinaryCodedDecimal object. You can pass IBinaryCodedDecimal objects in function calls.

RELATED TASKS

"Represent Numerical Quantities Using IBinaryCodedDecimal"
"Perform Calculations Using IBinaryCodedDecimal" on page 3
"Assign One IBinaryCodedDecimal to Another" on page 4
"Assign an IBinaryCodedDecimal to a long" on page 5
"Assign an IBinaryCodedDecimal to a double" on page 5

Represent Numerical Quantities Using IBinaryCodedDecimal

You can use the IBinaryCodedDecimal constructor to construct IBinaryCodedDecimal objects or arrays of IBinaryCodedDecimal objects.When you create a IBinaryCodedDecimal object, you can specify the following arguments:

- Number of significant digits: this number includes both integral and fractional parts of the number.
- Precision: this describes the number of digits used in the fraction part.
- Value of your number.

There are five basic ways that you can construct an IBinaryCodedDecimal object:

1. Create an IBinaryCodedDecimal object with no arguments.

   ```
   IBinaryCodedDecimal firstBCD();
   ```

   This is the default constructor. The above example creates an object with a value of zero, DFT_DIG number of digits (15), and precision of DFT_PREC digits (5).
2. Create an IBinaryCodedDecimal object with one argument:

   ```
   IBinaryCodedDecimal secondBCD(aNumber);
   ```

   The above example creates an object with a value of `aNumber`. The number of significant digits and precision depends on the data type of `aNumber`. For example, The following line of code creates an IBinaryCodedDecimal object to have a value of 12 with DFT_LNG_DIG number of digits (20) and number of precisions (0):

```
IBinaryCodedDecimal exampleBCD(12L);
```

3. Create an IBinaryCodedDecimal object with a string:

```
IBinaryCodedDecimal thirdBCD("12345.6789");
```

   The above example creates an object with a value of 12345.6789, nine significant digits, and four digits of precision. Use this constructor if you want to ensure accuracy. For example, the following line of code will store something close to 12345.6789 (such as 12345.6788999999999). This behavior will vary according to the floating type representation:

```
IBinaryCodedDecimal inaccurateBCD(12345.6789);
```

4. Create an IBinaryCodedDecimal object with two arguments:

```
IBinaryCodedDecimal fourthBCD(nDig, nPrec);
```

   The above example creates an object with a value of zero, `nDig` number of significant digits, and `nPrec` digits of precision.

5. Create an IBinaryCodedDecimal object with three arguments:

```
IBinaryCodedDecimal fifthBCD(nDig, nPrec, aNumber);
```

   The above example creates an object with a value of `aNumber`, `nDig` number of significant digits, and `nPrec` digits of precision. The following example shows how to construct an IBinaryCodedDecimal object to have a value INT_MAX with sixteen digits and five digits of precision.

```
IBinaryCodedDecimal example2BCD(16, 5, INT_MAX);
```

You may lose some precision during the conversion from a given data type to IBinaryCodedDecimal when you create IBinaryCodedDecimal objects.

When you use the member function precisionOf() with an IBinaryCodedDecimal object, you can find out the number of decimal digits in an IBinaryCodedDecimal object:

```
int p;
IBinaryCodedDecimal x(5, 2);
p=x.precisionOf();
// The result is p=2
```

When you use the member function digitsOf() with an IBinaryCodedDecimal object, you can find out the total number of digits in an IBinaryCodedDecimal object:

```
int n;
IBinaryCodedDecimal x(5, 2);
n = x.digitsOf();
// the result is n=5
```

The table below lists the binary coded decimal constants defined by the Binary Coded Decimal Class Library. You can find these constants in the idecimal.hpp file:

Constant name	Description
DEC_DIG	The maximum number of significant digits (31) that IBinaryCodedDecimal can hold.
DFT_INT_DIG	The default number of digits (10) for an integer type.
DEC_MIN	The minimum value that IBinaryCodedDecimal can hold.

Constant name	Description
DEC_MAX	The maximum value that IBinaryCodedDecimal can hold.
DEC_EPSILON	The smallest incremental or decremental value that IBinaryCodedDecimal can hold.
DFT_DIG	The default number of digits (15) for the default constructor.
DFT_PREC	The default number of precision (5) for the default constructor.
DFT_LNG_DIG	The default number of digits (20) for a long type.

RELATED CONCEPTS

"Chapter 1. The IBinaryCodedDecimal Class" on page 1

RELATED TASKS

"Perform Calculations Using IBinaryCodedDecimal"
"Assign One IBinaryCodedDecimal to Another" on page 4
"Assign an IBinaryCodedDecimal to a long" on page 5
"Assign an IBinaryCodedDecimal to a double" on page 5

Perform Calculations Using IBinaryCodedDecimal

The IBinaryCodedDecimal class defines a set of operators with the same precedence as the corresponding real operators. With these operators, you can code expressions on IBinaryCodedDecimal objects such as the expressions shown in the example below:

```
IBinaryCodedDecimal value1("123.78");
IBinaryCodedDecimal value2("345.12");
IBinaryCodedDecimal value3("77.457");
IBinaryCodedDecimal Sum, Average;

Sum = value1 + value2;
Sum = Sum + value3;
// Sum should have value 546.357

Average = Sum / 3;
// Average should have value 182.119
```

The IDecimalDataError exception class is thrown whenever the integral part is truncated as the result of any mathematical operation.

You can use the relational operators < > <= >= for IBinaryCodedDecimal objects and compare IBinaryCodedDecimal objects with other arithmetic types (integer, float, double, and long double):

```
IBinaryCodedDecimal BCD_1(15);
IBinaryCodedDecimal BCD_2(-15);

if (BCD_1 < BCD_2)
...
```

You can use equality operators with IBinaryCodedDecimal objects to compare IBinaryCodedDecimal objects for equality:

```
IBinaryCodedDecimal BCD_1(15);
IBinaryCodedDecimal BCD_2(-15);
```

```
if ( BCD_1 != BCD_2 )
...
```

RELATED CONCEPTS

"Chapter 1. The IBinaryCodedDecimal Class" on page 1

RELATED TASKS

"Represent Numerical Quantities Using IBinaryCodedDecimal" on page 1
"Assign One IBinaryCodedDecimal to Another"
"Assign an IBinaryCodedDecimal to a long" on page 5
"Assign an IBinaryCodedDecimal to a double" on page 5

Convert Between IBinaryCodedDecimal and Other Numeric Types

Assign One IBinaryCodedDecimal to Another

If the value of an IBinaryCodedDecimal object that is to be converted to another IBinaryCodedDecimal object is not within the range of values that can be represented exactly, the value of the IBinaryCodedDecimal object to be converted is truncated. If truncation occurs in the fractional part, there is no exception raised. If assignment causes truncation in the integral part, then there is an exception in which a IDecimalDataError object is thrown. This exception occurs when an integral value is lost during conversion to a different type, regardless of what operation requires the conversion:

```
IBinaryCodedDecimal targ_1(4,2);
IBinaryCodedDecimal targ_2(4,2);
IBinaryCodedDecimal op_1("1234.56");
IBinaryCodedDecimal op_2("12.34");

targ_1=op_1;   // An exception is generated because the integral
               // part is truncated; targ_1=("34.56").

targ_2=op_2;   // No exception is generated because neither the
               // integral nor the fractional part is truncated;
               // targ_2=("12.34").
```

An exception occurs on assignment to a smaller target only when the integral part is truncated.

When one IBinaryCodedDecimal object is assigned to another IBinaryCodedDecimal object with a smaller precision, the result is truncation of the fractional part:

```
IBinaryCodedDecimal x("123.4567");
IBinaryCodedDecimal y(7,1);

y = x;    // y = ("123.4")
```

When one IBinaryCodedDecimal object is assigned to another IBinaryCodedDecimal object with a smaller integral part, the result is truncation of the integral part. An exception occurs:

```
IBinaryCodedDecimal x("123456.78");
IBinaryCodedDecimal y(5,2);

y = x;   // y = ("456.78")
```

When one IBinaryCodedDecimal object is assigned to another IBinaryCodedDecimal object with a smaller integral part, and smaller precision, the result is truncation of the integral, and fractional parts. An exception occurs:

```
IBinaryCodedDecimal x("123456.78");
IBinaryCodedDecimal y(4,1);
y = x;  // y = ("456.7")
```

RELATED CONCEPTS

"Chapter 1. The IBinaryCodedDecimal Class" on page 1

RELATED TASKS

"Represent Numerical Quantities Using IBinaryCodedDecimal" on page 1
"Perform Calculations Using IBinaryCodedDecimal" on page 3
"Assign an IBinaryCodedDecimal to a long"
"Assign an IBinaryCodedDecimal to a double"

Assign an IBinaryCodedDecimal to a long

Convert an IBinaryCodedDecimal object with a fractional part to a long type:

```
long op;
IBinaryCodedDecimal op1("12345.67");
op = op1.asLong();
                    // Truncation on the fractional
                    // part. op=12345
```

Convert an IBinaryCodedDecimal object with less than 10 digits in the integral part to a long type:

```
long op;
IBinaryCodedDecimal op2("123");
op = op2.asLong();       // No truncation; op=123
```

Convert an IBinaryCodedDecimal object with more than 10 digits in the integral part to a long type:

```
long op2;
IBinaryCodedDecimal op3("123456789012");
op2 = op3.asLong();
                     // Truncation occurs on the integral
                     // part. op2=3456789012; Exception thrown.
```

RELATED CONCEPTS

"Chapter 1. The IBinaryCodedDecimal Class" on page 1

RELATED TASKS

"Represent Numerical Quantities Using IBinaryCodedDecimal" on page 1
"Perform Calculations Using IBinaryCodedDecimal" on page 3
"Assign One IBinaryCodedDecimal to Another" on page 4
"Assign an IBinaryCodedDecimal to a double"

Assign an IBinaryCodedDecimal to a double

To convert an IBinaryCodedDecimal object to a double-precision floating-point type call the asDouble member function:

```
#include <idecimal.hpp>
#include <iostream.h>
int main(void)
{
   IBinaryCodedDecimal dec_1("123.45");
   IBinaryCodedDecimal dec_2("-123456.12345");
   double f1,f2;
```

```
    f1=dec_1.asDouble();
    f2=dec_2.asDouble();
    cout <<"f1=" <<f1 <<endl <<"f2=" <<f2 <<endl <<endl;
    return 0;
}
```

The following is the output of this example:

```
f1=123
f2=-123456
```

The representation of a float will not exactly match the value of the IBinaryCodedDecimal object being converted.

RELATED CONCEPTS

"Chapter 1. The IBinaryCodedDecimal Class" on page 1

RELATED TASKS

"Represent Numerical Quantities Using IBinaryCodedDecimal" on page 1
"Perform Calculations Using IBinaryCodedDecimal" on page 3
"Assign One IBinaryCodedDecimal to Another" on page 4
"Assign an IBinaryCodedDecimal to a long" on page 5

Chapter 2. Complex Mathematics Library Overview

The Complex Mathematics Library provides you with the facilities to manipulate complex numbers and to perform standard mathematical operations on them. This library is comprised of two classes:

- complex is the class that lets you manipulate complex numbers
- c_exception is the class that you use to handle errors created by the functions and operations in the complex class.

The Complex Mathematics Library provides you with the following functionality:

- Mathematical operators with the same precedence as the corresponding real operators. With these operators, you can code expressions on complex numbers.
- Mathematical, trigonometric, magnitude, and conversion functions as friend functions of complex objects.
- Predefined mathematical constants.
- Input and output operators for I/O Stream Library input and output: Complex numbers are written to the output stream in the format (real,imag). Complex numbers are read from the input stream in one of two formats: (real,imag) or real.
- The c_exception class to handle errors. You can also define your own version of the error handling function.

RELATED CONCEPTS

Complex Mathematics Library Overview
"Review of Complex Numbers"
"Header Files and Constants for the complex and c_exception Classes" on page 8
"Mathematical Operators for complex" on page 10
"Mathematical Functions for complex" on page 12
"Input and Output Operators for complex" on page 15
"Error Functions" on page 17

RELATED TASKS

"Construct complex Objects" on page 9
"Handle complex Mathematics Errors" on page 18
"Example: Calculate Roots" on page 20
"Example: Use Equality and Inequality Operators" on page 22

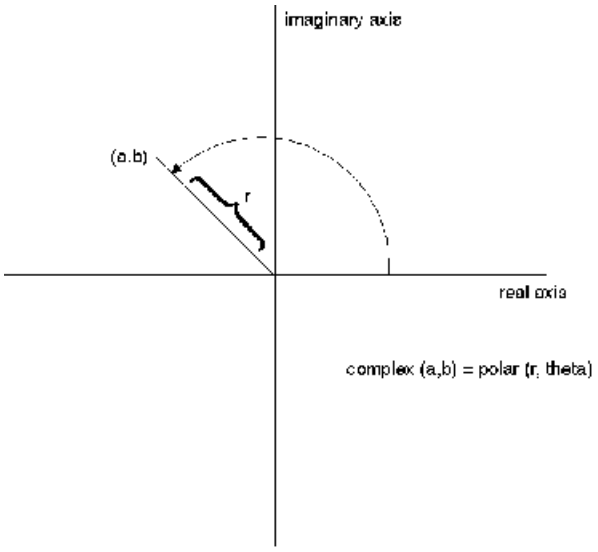
"Input and Output Operators for complex" on page 15

Review of Complex Numbers

A complex number is made up of two parts: a real part and an imaginary part. A complex number can be represented by an ordered pair (*a*,*b*), where *a* is the value of the real part of the number and *b* is the value of the imaginary part. If (*a*,*b*) and (*c*,*d*) are complex numbers, then the following statements are true:

- (*a*,*b*) + (*c*,*d*) = (*a*+*c*,*b*+*d*)
- (*a*,*b*) - (*c*,*d*) = (*a*-*c*,*b*-*d*)
- (*a*,*b*) * (*c*,*d*) = (*ac*-*bd*,*ad*+*bc*)

- $(a,b) / (c,d) = ((ac+bd) / (c^2+d^2), (bc-ad) / (c^2+d^2))$
- The conjugate of a complex number (a,b) is $(a,-b)$
- The absolute value or magnitude of a complex number (a,b) is the positive square root of the value $a^2 + b^2$
- The polar representation of (a,b) is $(r,theta)$, where r is the distance from the origin to the point (a,b) in the complex plane, and *theta* is the angle from the real axis to the vector (a,b) in the complex plane. The angle *theta* can be positive or negative. The following figure illustrates the polar representation $(r,theta)$ of the complex number (a,b).



RELATED CONCEPTS

Complex Mathematics Library Overview
"Header Files and Constants for the complex and c_exception Classes"
"Mathematical Operators for complex" on page 10
"Mathematical Functions for complex" on page 12
"Input and Output Operators for complex" on page 15
"Error Functions" on page 17

RELATED TASKS

"Construct complex Objects" on page 9
"Example: Calculate Roots" on page 20
"Example: Use Equality and Inequality Operators" on page 22

Header Files and Constants for the complex and c_exception Classes

To use the complex or c_exception classes, you must either:

- Specify the complex.h file in a source configuration file directive that applies to any build target using these classes.
- Include the following statement in any file using these classes:

```
#include <complex.h>
```

Constants Defined in complex.h

The following table lists the mathematical constants that the Complex Mathematics

Library defines.

Constant Name	Description
M_E	The constant *e*
M_LOG2E	The logarithm of *e* to the base 2
M_LOG10E	The logarithm of *e* to the base 10
M_LN2	The natural logarithm of 2
M_LN10	The natural logarithm of 10
M_PI	π (pi)
M_PI_2	pi divided by two
M_PI_4	pi divided by four
M_1_PI	1/ π (1/pi)
M_2_PI	2/ π (2/pi)
M_2_SQRTPI	2 divided by the square root of π (pi)
M_SQRT2	The square root of 2
M_SQRT1_2	The square root of 1/2

RELATED CONCEPTS

Complex Mathematics Library Overview
"Review of Complex Numbers" on page 7
"Mathematical Operators for complex" on page 10
"Mathematical Functions for complex" on page 12
"Input and Output Operators for complex" on page 15
"Error Functions" on page 17

RELATED TASKS

"Construct complex Objects"

Construct complex Objects

You can use the complex constructor to construct initialized or uninitialized complex objects or arrays of complex objects. The following example shows different ways of creating and initializing complex objects:

```
complex comp1;                  // Initialized to (0, 0)
complex comp2(3.14);            // Initialized to (3.14, 0)
complex comp3(3.14,2.72);       // Initialized to (3.14, 2.72)
complex comparr1[3]={
   1.0,                         // Initialized to (1.0, 0)
   complex(2.0,-2.0),           //                (2.0, -2.0)
   3.0                          //                (3.0, 0)
```

```
   };
complex comparr2[3]={
   complex(1.0,1.0),           // Initialized to (1.0, 1.0)
   2.0,                        // (2.0, 0)
   complex(3.0,-3.0)           // (3.0, -3.0)
   };
complex comparr3[3]={
   1.0,                        // Initialized to (1.0, 0)
   complex(M_PI_4,M_SQRT2),    // (0.785..., 1.414...)
   M_SQRT1_2                   // (0.707..., 0)
   };
```

RELATED CONCEPTS

Complex Mathematics Library Overview
"Header Files and Constants for the complex and c_exception Classes" on page 8

RELATED TASKS

"Use complex Input and Output Operators" on page 16
"Use Mathematical Operators for complex" on page 11
"Use Friend Functions with complex" on page 13

Mathematical Operators for complex

The complex class defines a set of mathematical operators with the same precedence as the corresponding real operators. With the following operators, you can code expressions on complex numbers:

- operator + (addition)
- operator * (multiplication)
- operator - (negation)
- operator - (subtraction)
- operator / (division)
- operator += (assignment)
- operator -= (assignment)
- operator *= (assignment)
- operator /= (assignment)
- operator == (equality)
- operator != (inequality)

Assignment operators do not produce an lvalue. The complex mathematical assignment operators (+=, -=, *=, /=) do not produce a value that can be used in an expression.

The equality and inequality operators test for an exact equality between the real parts of two numbers, and between their complex parts. Because both components are double values, two numbers may be "equal" within a certain tolerance, but unequal as far as these operators are concerned. If you want an equality or inequality operator that can test for an absolute difference within a certain tolerance between the two pairs of corresponding components, you should define your own equality functions rather than use the equality and inequality operators of the complex class.

Assignment Operators Do Not Produce an lvalue
The complex mathematical assignment operators (+=, -=, *=, /=) do not produce a value that can be used in an expression. The following code, for example, produces a compile-time error:

```
complex x, y, z; // valid declaration
x = (y += z );   // invalid assignment causes
                 // a compile-time error
```

RELATED CONCEPTS

Complex Mathematics Library Overview
"Review of Complex Numbers" on page 7
"Header Files and Constants for the complex and c_exception Classes" on page 8
"Mathematical Functions for complex" on page 12
"Input and Output Operators for complex" on page 15
"Error Functions" on page 17

RELATED TASKS

"Use Mathematical Operators for complex"

Use Mathematical Operators for complex

With these operators, you can code expressions on complex numbers such as the expressions shown in the example below. In the example, for each complex scalar x, the comments showing the results of operations use xr to denote the scalar's real part and xi to denote the scalar's imaginary part.

```
//   Using the complex mathematical operators
#include <complex.h>
#include <iostream.h>
complex a,b,c,d,e,f,g;
void main()
{
   cout << "Enter six complex numbers, separated by spaces:\n";
   cin >> b >> c >> d >> e >> f >> g;

   // assignment, multiplication, addition
   a=b*c+d; // a=( (br*cr)-(bi*ci)+dr , (br*ci)+(bi*cr)+di )

   // division
   a=b/d; // a=( (br*dr)+(bi*di) / ((br*br)+(bi*bi),
          //     (bi*dr)-(br*di) / ((br*br)+(bi*bi) )

   // subtraction
   a=b-f; // a=( (br-fr), (bi-fi) )

  // equality, multiplication assignment
   if (a==f) c*=e; // same as c=c*e;

  // inequality, addition assignment
   if (b!=f) d+=g; // same as d=d+g;

   cout << "Here are the seven numbers after calculations:\n"
        << "a=" << a << '\n'
        << "b=" << b << '\n'
        << "c=" << c << '\n'
        << "d=" << d << '\n'
        << "e=" << e << '\n'
        << "f=" << f << '\n'
        << "g=" << g << endl;
}
```

This example produces the output shown below in regular type, given the input shown in bold:

```
Enter six complex numbers, separated by spaces:
(1.14,2.28) (2.24,4.48) (1.17,12.18)
(4.4444444,5.12341) (12,7) 5
Here are the seven numbers after calculations:
a=( -10.86, -4.72)
b=( 1.14, 2.28)
c=( 2.24, 4.48)
d=( 6.17, 12.18)
e=( 4.44444, 5.12341)
f=( 12, 7)
g=( 5, 0)
```

Note that there are no increment or decrement operators for complex numbers.

RELATED CONCEPTS

Complex Mathematics Library Overview
"Header Files and Constants for the complex and c_exception Classes" on page 8
"Mathematical Operators for complex" on page 10

RELATED TASKS

"Construct complex Objects" on page 9
"Use complex Input and Output Operators" on page 16
"Use Friend Functions with complex" on page 13

Mathematical Functions for complex

The complex class defines a set of mathematical, trigonometric, magnitude, and conversion functions as friend functions of complex objects. They are:

- exp (exponent)
- log (natural logarithm)
- pow (power)
- sqrt (square root)
- cos (cosine)
- cosh (hyperbolic cosine)
- sin (sine)
- sinh (hyperbolic sine)
- abs (absolute value or magnitude)
- norm (square of magnitude)
- arg (polar angle)
- conj (conjugate)
- polar (polar to complex)
- real (real part)
- imag (imaginary part)

Because these functions are friend functions rather than member functions, you cannot use the dot or arrow operators. For example:

```
complex a,b,*c;
a=exp(b);   // correct - exp() is a friend function of complex
a=b.exp();  // error - exp() is not a member function of complex
a=c->exp(); // error - exp() is not a member function of complex
```

RELATED CONCEPTS

Complex Mathematics Library Overview
"Review of Complex Numbers" on page 7
"Header Files and Constants for the complex and c_exception Classes" on page 8
"Mathematical Operators for complex" on page 10
"Input and Output Operators for complex" on page 15
"Error Functions" on page 17

RELATED TASKS

"Use Friend Functions with complex"

Use Friend Functions with complex

The complex class defines a set of mathematical, trigonometric, magnitude and conversion functions as friend functions of complex objects. Because these functions are friend functions rather than member functions, you cannot use the dot or arrow operators. For example:

```
complex a, b, *c;
a - exp(b);     //correct - exp() is a friend function of complex
a = b.exp();    //error - exp() is not a member function of complex
a = c -> exp(); //error - exp() is not a member function of complex
```

Use Mathematical Functions for complex

The complex class defines four mathematical functions as friend functions of complex objects.

- exp - Exponent
- log - Logarithm
- pow - Power
- sqrt - Square Root

The following example shows uses of these mathematical functions:

```
// Using the complex mathematical functions

#include <complex.h>
#include <iostream.h>

void main()
{
   complex a, b;
   int i;
   double f;
   //
   // prompt the user for an argument for calls to
   // exp(), log(), and sqrt()
   //
   cout << "Enter a complex value\n";
   cin >> a;
   cout << "The value of exp() for " << a << " is: " << exp(a)
       << "\nThe natural logarithm of " << a << " is: " << log(a)
       << "\nThe square root of " << a << " is: " << sqrt(a) << "\n\n";
   //
   // prompt the user for arguments for calls to pow()
   //
   cout << "Enter 2 complex values (a and b), an integer (i),"
        << " and a floating point value (f)\n";
   cin >> a >> b >> i >> f;
   cout << "a is " << a << ", b is " << b << ", i is " << i
        << ", f is " << f << '\n'
        << "The value of f**a is: " << pow(f, a) << '\n'
        << "The value of a**i is: " << pow(a, i) << '\n'
        << "The value of a**f is: " << pow(a, f) << '\n'
        << "The value of a**b is: " << pow(a, b) << endl;
   }
```

This example produces the output shown below in regular type, given the input shown in bold:

```
Enter a complex value
(3.7,4.2)
The value of exp() for ( 3.7, 4.2)  is: ( -19.8297, -35.2529)
The natural logarithm of ( 3.7, 4.2) is: ( 1.72229, 0.848605)
The square root of ( 3.7, 4.2) is: ( 2.15608, 0.973992)

Enter 2 complex values (a and b), an integer (i), and a floating point value (f)
(2.6,9.39) (3.16,1.16) -7 33.16237
a is ( 2.6, 9.39), b is ( 3.16, 1.16), i is -7, f is 33.1624
The value of f**a is: ( 972.681, 8935.53)
The value of a**i is: ( -1.13873e-07, -3.77441e-08)
The value of a**f is: ( 4.05451e+32, -4.60496e+32)
The value of a**b is: ( 262.846, 132.782)
```

Use Trigonometric Functions for complex
The complex class defines four trigonometric functions as friend functions of complex objects.

- cos - Cosine
- cosh - Hyperbolic cosine
- sin - Sine
- sinh - Hyperbolic sine

The following example shows how you can use some of the complex trigonometric functions:

```
// Complex Mathematics Library trigonometric functions

#include <complex.h>
#include <istream.h>

void main()

{
   complex a = (M_PI, M_PI_2); // a = (pi,pi/2)
   // display the values of cos(), cosh(), sin(), and sinh()
   // for (pi,pi/2)
   cout << "The value of cos() for (pi,pi/2) is: " << cos(a) << '\n'
        << "The value of cosh() for (pi,pi/2) is: " << cosh(a) << '\n'
        << "The value of sin() for (pi,pi/2) is: " << sin(a) << '\n'
        << "The value of sinh() for (pi,pi/2) is: " << sinh(a) << endl;
}
```

This program produces the following output:

```
The value of cos()  for (pi,pi/2) is: ( 6.12323e-17, 0)
The value of cosh() for (pi,pi/2) is: ( 2.50918, 0)
The value of sin()  for (pi,pi/2) is: ( 1, -0)
The value of sinh() for (pi,pi/2) is: ( 2.3013, 0)
```

Use Magnitude Functions for complex
The magnitude functions for complex are:

- abs - Absolute value
- norm - Square magnitude

Use Conversion Functions for complex
The conversion functions in the Complex Mathematics Library allow you to convert between the polar and standard complex representations of a value and to extract the real and imaginary parts of a complex value.

The complex class provides the following conversion functions as friend functions of complex objects:

- arg - angle in radians

- conj - conjugation
- polar - polar to complex
- real -extract to real part
- imag - extract imaginary part

The following program shows how to use complex conversion functions:

```
// Using the complex conversion functions

#include <complex.h>
#include <iostream.h>

void main()
{
   complex a;

   //for a value supplied by the user, display the real part,
   //the imaginary part, and the polar representation.

   cout << "Enter a complex value" << endl;

   cin >> a;

   cout << "The real part of this value is " << real(a) << endl'
   cout << "The imaginary part of this value is " << img(a) << endl;
   cout << "The polar representation of this value is " << "( <<abs(a) << "," << arg(a) << ")"

}
```

This example produces the output shown below, given the input shown in bold:

```
Enter a complex value
(175,162)
The real part of this value is 175
The imaginary part of this value is 167
The polar representation of this value is (238,472,0.746842)
```

RELATED CONCEPTS

Complex Mathematics Library Overview
"Header Files and Constants for the complex and c_exception Classes" on page 8
"Mathematical Functions for complex" on page 12

RELATED TASKS

"Construct complex Objects" on page 9
"Use complex Input and Output Operators" on page 16
"Use Mathematical Operators for complex" on page 11

Input and Output Operators for complex

The complex class defines input and output operators for I/O Stream Library input and output:

- operator >> (input)
- operator << (output)

Complex numbers are written to the output stream in the format (real,imag). Complex numbers are read from the input stream in one of two formats: (real,imag) or real.

RELATED CONCEPTS

Complex Mathematics Library Overview
"Review of Complex Numbers" on page 7
"Header Files and Constants for the complex and c_exception Classes" on page 8

"Mathematical Operators for complex" on page 10
"Mathematical Functions for complex" on page 12
"Error Functions" on page 17

RELATED TASKS

"Use complex Input and Output Operators"

Use complex Input and Output Operators

The following example shows you how to use the complex input and output operators, and provides some sample input and the resulting output.

```
// An example of complex input and output
#include <complex.h>  // required for use of Complex
                      // Mathematics Library
#include <iostream.h> // required for use of I/O
                      // Stream input and output
void main()
{
   complex a[3]={1.0, 2.0, complex(3.0,-3.0)};
   complex b[3];
   complex c[3];
   complex d;

   // read input for all of arrays b and c
   // (you must specify each element individually)
   cout << "Enter three complex values separated "
        << "by spaces:\n";
   cin >> b[0] >> b[1] >> b[2];
   cout << "Enter three more complex values:\n";
   cin >> c[2] >> c[0] >> c[1];
   // read input for scalar d
   cout << "Enter one more complex value:\n";
   cin >> d;
   // Note that you cannot use the above notation
   // for arrays. For example,
   //    cin >> a;
   // is incorrect because a is a complex array.

   // display each array of three complex numbers,
   // then the complex scalar
   cout << "Here are some elements of arrays a, "
        << "b, and c:\n"
        << a[2] << '\n'
        << b[0] << b[1] << b[2] << '\n'
        << c[1] << '\n'
        << "Here is scalar d: " << d << '\n'
   // cout << a produces an address, not a list of
   // array elements:
        << "Here is the address of array a:\n"
        << a
        << endl; // endl flushes the output stream
}
```

This example produces the output shown below in regular type, given the input shown in bold. Notice that you can insert white space within a complex number, between the brackets, numbers, and comma. However, you cannot insert white space within the real or imaginary part of the number. The address displayed may be different, or in a different format, than the address shown, depending on the operating system, hardware, and other factors.

```
Enter three complex values separated by spaces:
38 (12.2,3.14159) (1712,-33)
Enter three more complex values:
(     17.1234  ,  1234.17) ( 27,    12) (-33    ,0)
Enter one more complex value:
```

```
17
Here are some elements of arrays a, b, and c:
( 3, -3)
( 38, 0)( 12.2, 3.14159)( 1712, -33)
( -33, 0)
Here is scalar d: ( 17, 0)
Here is the address of array a:
0x2ff7f9b8
```

RELATED CONCEPTS

Complex Mathematics Library Overview
"Header Files and Constants for the complex and c_exception Classes" on page 8
"Input and Output Operators for complex" on page 15
I/O Stream Classes

RELATED TASKS

"Construct complex Objects" on page 9
"Use Mathematical Operators for complex" on page 11
"Use Friend Functions with complex" on page 13
Combine Input and Output of Different Types
Receive Input from Standard Input
Display Output from Standard Output or Standard Error

Error Functions

Use the **c_exception** class to handle errors that are created by functions and operations in the complex class.

The c_exception class is not related to the C++ exception handling mechanism that uses the try, catch, and throw statements.

The c_exception class lets you handle errors that are created by the functions and operations in the complex class. When the Complex Mathematics Library detects an error in a complex operation or function, it invokes complex_error(). This friend function of c_exception has a c_exception object as its argument. When the function is invoked, the c_exception object contains data members that define the function name, arguments, and return value of the function that caused the error, as well as the type of error that has occurred. The data members are:

```
complex arg1;   // First argument of the error-causing function
complex arg2;   // Second argument of the error-causing function
char* name;     // Name of the error-causing function
complex retval; // Value returned by default definition of complex_error
int type;       // The type of error that has occurred.
```

If you do not define your own complex_error function, complex_error sets the complex return value and the errno error number.

Defining a Customized complex_error Function
You can either use the default version of complex_error() or define your own version of the function. If you define your own complex_error() function, and this function returns a nonzero value, no error message will be generated.

Handling Errors Outside of the Complex Mathematics Library
There are some cases where member functions of the Complex Mathematics Library call functions in the math library. These calls can cause underflow and

overflow conditions that are handled by the matherr() function that is declared in the math.h header file. For example, the overflow conditions that are caused by the following calls are handled by matherr():

- `exp(complex(DBL_MAX, DBL_MAX))`
- `pow(complex(DBL_MAX, DBL_MAX), INT_MAX)`
- `norm(complex(DBL_MAX, DBL_MAX))`

DBL_MAX is the maximum valid double value, and is defined in float.h.
INT_MAX is the maximum int value, and is defined in limits.h.

If you do not want the default error-handling defined by matherr(), you should define your own version of matherr().

RELATED CONCEPTS

Complex Mathematics Library Overview
"Review of Complex Numbers" on page 7
"Header Files and Constants for the complex and c_exception Classes" on page 8
"Mathematical Operators for complex" on page 10
"Mathematical Functions for complex" on page 12
"Input and Output Operators for complex" on page 15

RELATED TASKS

"Handle complex Mathematics Errors"
"Handle Errors Outside of the complex Mathematics Library" on page 20

Handle complex Mathematics Errors

This page outlines the following tasks:

- Use c_exception to Handle complex Mathematics Errors
- Define a Customized complex_error Function
- Compile a Program that Uses a Customized complex_error Function

Use c_exception to Handle complex Mathematics Errors
The c_exception class is not related to the C++ exception handling mechanism that uses the try, catch, and throw statements.

The c_exception class lets you handle errors that are created by the functions and operations in the complex class. When the Complex Mathematics Library detects an error in a complex operation or function, it invokes complex_error(). This friend function of c_exception has a c_exception object as its argument. When the function is invoked, the c_exception object contains data members that define the function name, arguments, and return value of the function that caused the error, as well as the type of error that has occurred. The data members are as follows:

```
complex arg1;   // First argument of the
                //    error-causing function
complex arg2;   // Second argument of the
                //    error-causing function
char* name;     // Name of the error-causing function
complex retval; // Value returned by default
                //    definition of complex_error
int type;       // The type of error that has occurred.
```

If you do not define your own complex_error function, complex_error sets the complex return value and the errno error number.

Define a Customized complex_error Function

You can either use the default version of complex_error() or define your own version of the function. In the following example, complex_error() is redefined:

```
// Redefinition of the complex_error function

#include <iostream.h>
#include <complex.h>
#include <float.h>

int complex_error(c_exception &c)
{
   cout << "================" << endl;
   cout << "   Exception " << endl;
   cout << "type = " << c.type << endl;
   cout << "name = " << c.name << endl;
   cout << "arg1 = " << c.arg1 << endl;
   cout << "arg2 = " << c.arg2 << endl;
   cout << "retval = " << c.retval << endl;
   cout << "================" << endl;
   return 0;
 }

void main()
{
   complex c1(DBL_MAX,0);
   complex result;
   result = exp(c1);
   cout << "exp" << c1 << "= " << result << endl;
}
```

This example produces the following output:

```
================
   Exception
type = 3
name = exp
arg1 = ( 1.79769e+308, 0)
arg2 = ( 0, 0)
retval = ( infinity, -infinity)
================
exp( 1.79769e+308, 0)= ( infinity, -infinity)
```

If the redefinition of complex_error() in the above code is commented out, the default definition of complex_error() is used, and the program produces the following output:

```
exp( 1.79769e+308, 0) = ( infinity, -infinity)
```

If the redefinition of complex_error() in the above code is commented out, the default definition of complex_error() is used, and the program produces the following output:

```
exp( 7.23701e+75, 0) = ( 7.23701e+75, -7.23701e+75)
```

Compile a Program that Uses a Customized complex_error Function

If you define your own version of complex_error, you must ensure that the name of the header file that contains your version of the complex_error is included in your source file when you compile you program.

RELATED CONCEPTS

Complex Mathematics Library Overview
"Header Files and Constants for the complex and c_exception Classes" on page 8
"Error Functions" on page 17

RELATED TASKS

"Handle Errors Outside of the complex Mathematics Library"

Handle Errors Outside of the complex Mathematics Library

There are some cases where member functions of the Complex Mathematics Library call functions in the math library. These calls can cause underflow and overflow conditions that are handled by the matherr() function that is declared in the math.h header file. For example, the overflow conditions that are caused by the following calls are handled by matherr():

- `exp(complex(DBL_MAX, DBL_MAX))`
- `pow(complex(DBL_MAX, DBL_MAX), INT_MAX)`
- `norm(complex(DBL_MAX, DBL_MAX))`

DBL_MAX is the maximum valid double value, and is defined in float.h. INT_MAX is the maximum int value, and is defined in limits.h.

If you do not want the default error-handling defined by matherr(), you should define your own version of matherr().

RELATED CONCEPTS

Complex Mathematics Library Overview
"Header Files and Constants for the complex and c_exception Classes" on page 8
"Error Functions" on page 17

RELATED TASKS

"Handle complex Mathematics Errors" on page 18

Example: Calculate Roots

The following example shows how you can use the complex Mathematics Library to calculate the roots of a complex number. For every positive integer *n*, each complex number *z* has exactly *n* distinct *n*th roots. Suppose that in the complex plane the angle between the real axis and point *z* is *Ï*, and the distance between the origin and the point *z* is *r*. Then *z* has the polar form `(r, Ï)`, and the *n* roots of *z* have the values:

.■
.■2
Ë.■3
·
·
·
·■n-1

where ■ is a complex number with the value:

■ = `( cos(2Â/n), sin(2Â/n) )`

and is a complex number with the value:

 = r^1/n `( cos(Ï/n), sin(Ï/n) )`

The following code includes two functions, `get_omega()` and `get_sigma()`, to calculate the values of ■ and . The user is prompted for the complex value *z* and the value of *n*. After the values of ■ and have been calculated, the *n* roots of *z* are calculated and printed.

```
// Calculating the roots of a complex number
#include <iostream.h>
#include <complex.h>
#include <math.h>

// Function to calculate the value of omega for a given value of n

complex get_omega(double n)

{
   complex omega = complex(cos((2.0*M_PI)/n), sin((2.0*M_PI)/n));
   return omega;
}

// function to calculate the value of sigma for a given value of
// n and a given complex value
complex get_sigma(complex comp_val, double n)
{
   double rn, r, theta;
   complex sigma;
   r = abs(comp_val);
   theta = arg(comp_val);
   rn = pow(r,(1.0/n));
   sigma = rn * complex(cos(theta/n),sin(theta/n));
   return sigma;
}

void main()
{
   double n;
   complex input, omega, sigma;
   //
   // prompt the user for a complex number
   //
   cout << "Please enter a complex number: ";
   cin >> input;
   //
   // prompt the user for the value of n
   //
   cout << "What root would you like of this number? ";
   cin >> n;
   //
   // calculate the value of omega
   //
   omega = get_omega(n);
   cout << "Here is omega " << omega << endl;
   //
   // calculate the value of sigma
   //
   sigma = get_sigma(input,n);
   cout << "Here is sigma " << sigma << '\n'
        << "Here are the " << n << " roots of " << input << endl;
   for (int i = 0; i < n ; i++)
{
       cout << sigma*(pow(omega,i)) << endl;
   }
}
```

This example produces the output shown below in regular type, given the input shown in bold:

```
Please enter a complex number: (-7, 24)
What root would you like of this number? 2
Here is omega ( -1, 1.22465e-16)
Here is sigma ( 3, 4)
Here are the 2 roots of ( -7, 24)
( 3, 4)
( -3, -4)
```

RELATED CONCEPTS

Complex Mathematics Library Overview
"Review of Complex Numbers" on page 7
"Header Files and Constants for the complex and c_exception Classes" on page 8

RELATED TASKS

"Example: Use Equality and Inequality Operators"

Example: Use Equality and Inequality Operators

The functions is_equal and is_not_equal in the following example provide a reliable comparison between two complex values:

```
// Testing complex values for equality within a certain tolerance

#include <complex.h>
#include <iostream.h>            // for output
#include <iomanip.h>             // for use of setw() manipulator

int is_equal(const complex &a, const complex &b,
             const double tol=0.0001)
{
   return (abs(real(a) - real(b)) < tol &&
           abs(imag(a) - imag(b)) < tol);
}

int is_not_equal(const complex &a, const complex &b,
                 const double tol=0.0001)
{
   return !is_equal(a, b, tol);
}

void main()
{
    complex c[4] = { complex(1.0, 2.0),
                     complex(1.0, 2.0),
                     complex(3.0, 4.0),
                     complex(1.0000163,1.999903581)};
   cout << "Comparison of array elements c[0] to c[3]\n"
        << "== means identical,\n!= means unequal,\n"
        << "~ means equal within tolerance of 0.0001.\n\n"
        << setw(10) << "Element"
        << setw(6) << 0
        << setw(6) << 1
        << setw(6) << 2
        << setw(6) << 3
        << endl;
   for (int i=0;i<4;i++) {
      cout << setw(10) << i;
      for (int j=0;j<4;j++) {
         if (c[i]==c[j]) cout << setw(6) << "==";
         else if (is_equal(c[i],c[j])) cout << setw(6) << "~";
             else if (is_not_equal(c[i],c[j])) cout << setw(6) << "!=";
            else cout << setw(6) << "???";
         }
      cout << endl;
      }
   }
```

This example produces the following output:

```
Comparison of array elements c[0] to c[3]
== means identical,
!= means unequal,
~ means equal within tolerance of 0.0001.
   Element     0     1     2     3
```

```
                          ~
0    ==     ==     !=     ~
1    ==     ==     !=
2    !=     !=     ==     !=
       ~      ~
3                  !=     ==
```

RELATED CONCEPTS

Complex Mathematics Library Overview
"Review of Complex Numbers" on page 7
"Header Files and Constants for the complex and c_exception Classes" on page 8

RELATED TASKS

"Example: Calculate Roots" on page 20