Illustrations List ***(Main Page)***

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

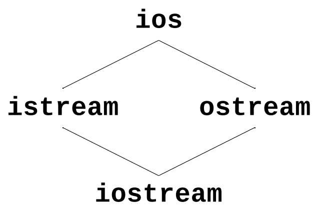

Fig. 11.1 Portion of the stream I/O class hierarchy.

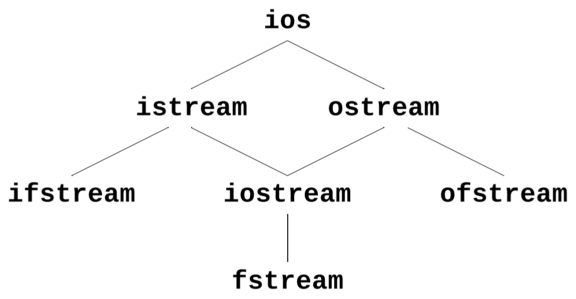

Fig. 11.2 Portion of stream-I/O class hierarchy with key file-processing classes.

```
// Fig. 11.3: fig11_03.cpp
// Outputting a string using stream insertion.
#include <iostream.h>

int main()
{
   cout << "Welcome to C++!\n";

   return 0;
}
```

```
Welcome to C++!
```

Fig. 11.3 Outputting a string using stream insertion.

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```
// Fig. 11.4: fig11_04.cpp
// Outputting a string using two stream insertions.
#include <iostream.h>

int main()
{
   cout << "Welcome to ";
   cout << "C++!\n";

   return 0;
}
```

```
Welcome to C++!
```

Fig. 11.4 Outputting a string using two stream insertions.

```
// Fig. 11.5: fig11_05.cpp
// Using the endl stream manipulator.
#include <iostream.h>

int main()
{
   cout << "Welcome to ";
   cout << "C++!";
   cout << endl;   // end line stream manipulator

   return 0;
}
```

```
Welcome to C++!
```

Fig. 11.5 Using the `endl` stream manipulator.

```
// Fig. 11.6: fig11_06.cpp
// Outputting expression values.
#include <iostream.h>

int main()
{
   cout << "47 plus 53 is ";

   // parentheses not needed; used for clarity
   cout << ( 47 + 53 );      // expression
   cout << endl;

   return 0;
}
```

```
47 plus 53 is 100
```

Fig. 11.6 Outputting expression values.

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```
// Fig. 11.7: fig11_07.cpp
// Cascading the overloaded << operator.
#include <iostream.h>

int main()
{
   cout << "47 plus 53 is " << ( 47 + 53 ) << endl;

   return 0;
}
```

```
47 plus 53 is 100
```

Fig. 11.7 Cascading the overloaded `<<` operator.

```
// Fig. 11.8: fig11_08.cpp
// Printing the address stored in a char* variable
#include <iostream.h>

int main()
{
   char *string = "test";

   cout << "Value of string is: " << string
        << "\nValue of static_cast< void *>( string ) is: "
        << static_cast< void *>( string ) << endl;
   return 0;
}
```

```
Value of string is: test
Value of static_cast< void *>( string ) is: 0x00416D50
```

Fig. 11.8 Printing the address stored in a `char *` variable.

```
// Fig. 11.9: fig11_09.cpp
// Calculating the sum of two integers input from the keyboard
// with the cin object and the stream-extraction operator.
#include <iostream.h>

int main()
{
   int x, y;

   cout << "Enter two integers: ";
   cin >> x >> y;
   cout << "Sum of " << x << " and " << y << " is: "
        << ( x + y ) << endl;

   return 0;
}
```

```
Enter two integers: 30 92
Sum of 30 and 92 is: 122
```

Fig. 11.9 Calculating the sum of two integers input from the keyboard with `cin` and the stream-extraction operator.

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```
// Fig. 11.10: fig11_10.cpp
// Avoiding a precedence problem between the stream-insertion
// operator and the conditional operator.
// Need parentheses around the conditional expression.
#include <iostream.h>

int main()
{
   int x, y;

   cout << "Enter two integers: ";
   cin >> x >> y;
   cout << x << ( x == y ? " is" : " is not" )
        << " equal to " << y << endl;

```

Fig. 11.10 Avoiding a precedence problem between the stream-insertion operator and the conditional operator (part 1 of 2).

```
   return 0;
}
```

```
Enter two integers: 7 5
7 is not equal to 5
```

```
Enter two integers: 8 8
8 is equal to 8
```

Fig. 11.10 Avoiding a precedence problem between the stream-insertion operator and the conditional operator (part 2 of 2).

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```
// Fig. 11.11: fig11_11.cpp
// Stream-extraction operator returning false on end-of-file.
#include <iostream.h>

int main()
{
   int grade, highestGrade = -1;

   cout << "Enter grade (enter end-of-file to end): ";
   while ( cin >> grade ) {
      if ( grade > highestGrade )
         highestGrade = grade;

      cout << "Enter grade (enter end-of-file to end): ";
   }

   cout << "\n\nHighest grade is: " << highestGrade << endl;
   return 0;
}
```

```
Enter grade (enter end-of-file to end): 67
Enter grade (enter end-of-file to end): 87
Enter grade (enter end-of-file to end): 73
Enter grade (enter end-of-file to end): 95
Enter grade (enter end-of-file to end): 34
Enter grade (enter end-of-file to end): 99
Enter grade (enter end-of-file to end): ^Z
Highest grade is: 99
```

Fig. 11.11 Stream-extraction operator returning false on end-of-file.

```
// Fig. 11.12: fig11_12.cpp
// Using member functions get, put, and eof.
#include <iostream.h>

int main()
{
   char c;

   cout << "Before input, cin.eof() is " << cin.eof()
        << "\nEnter a sentence followed by end-of-file:\n";

   while ( ( c = cin.get() ) != EOF)
      cout.put( c );

   cout << "\nEOF in this system is: " << c;
   cout << "\nAfter input, cin.eof() is " << cin.eof() << endl;
   return 0;
}
```

```
Before input, cin.eof() is 0
Enter a sentence followed by end-of-file:
Testing the get and put member functions^Z
Testing the get and put member functions
EOF in this system is: -1
After input cin.eof() is 1
```

Fig. 11.12 Using member functions `get`, `put`, and `eof`.

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```
// Fig. 11.13: fig11_13.cpp
// Contrasting input of a string with cin and cin.get.
#include <iostream.h>

int main()
{
   const int SIZE = 80;
   char buffer1[ SIZE ], buffer2[ SIZE ];

   cout << "Enter a sentence:\n";
   cin >> buffer1;
   cout << "\nThe string read with cin was:\n"
        << buffer1 << "\n\n";

   cin.get( buffer2, SIZE );
   cout << "The string read with cin.get was:\n"
        << buffer2 << endl;

   return 0;
}
```

```
Enter a sentence:
Contrasting string input with cin and cin.get

The string read with cin was:
Contrasting

The string read with cin.get was:
 string input with cin and cin.get
```

Fig. 11.13 Contrasting input of a string using `cin` with stream extraction and input with `cin.get`.

```
// Fig. 11.14: fig11_14.cpp
// Character input with member function getline.
#include <iostream.h>

int main()
{
   const SIZE = 80;
   char buffer[ SIZE ];

   cout << "Enter a sentence:\n";
   cin.getline( buffer, SIZE );

   cout << "\nThe sentence entered is:\n" << buffer << endl;
   return 0;
}
```

Fig. 11.14 Character input with member function `getline` (part 1 of 2).

```
Enter a sentence:
Using the getline member function

The sentence entered is:
Using the getline member function
```

Fig. 11.14 Character input with member function `getline` (part 2 of 2).

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```cpp
// Fig. 11.15: fig11_15.cpp
// Unformatted I/O with read, gcount and write.
#include <iostream.h>

int main()
{
   const int SIZE = 80;
   char buffer[ SIZE ];

   cout << "Enter a sentence:\n";
   cin.read( buffer, 20 );
   cout << "\nThe sentence entered was:\n";
   cout.write( buffer, cin.gcount() );
   cout << endl;
   return 0;
}
```

```
Enter a sentence:
Using the read, write, and gcount member functions

The sentence entered was:
Using the read, writ
```

Fig. 11.15 Unformatted I/O with the `read`, `gcount` and `write` member functions.

```cpp
// Fig. 11.16: fig11_16.cpp
// Using hex, oct, dec and setbase stream manipulators.
#include <iostream.h>
#include <iomanip.h>

int main()
{
   int n;

   cout << "Enter a decimal number: ";
   cin >> n;

   cout << n << " in hexadecimal is: "
        << hex << n << '\n'
        << dec << n << " in octal is: "
        << oct << n << '\n'
        << setbase( 10 ) << n << " in decimal is: "
        << n << endl;

   return 0;
}
```

Fig. 11.16 Using the `hex`, `oct`, `dec` and `setbase` stream manipulators (part 1 of 2).

```
Enter a decimal number: 20
20 in hexadecimal is: 14
20 in octal is: 24
20 in decimal is: 20
```

Fig. 11.16 Using the `hex`, `oct`, `dec` and `setbase` stream manipulators (part 2 of 2).

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```cpp
// Fig. 11.17: fig11_17.cpp
// Controlling precision of floating-point values
#include <iostream.h>
#include <iomanip.h>
#include <math.h>

int main()
{
   double root2 = sqrt( 2.0 );
   int places;

   cout << setiosflags( ios::fixed)
        << "Square root of 2 with precisions 0-9.\n"
        << "Precision set by the "
        << "precision member function:" << endl;

   for ( places = 0; places <= 9; places++ ) {
      cout.precision( places );
      cout << root2 << '\n';
   }

   cout << "\nPrecision set by the "
        << "setprecision manipulator:\n";

   for ( places = 0; places <= 9; places++ )
      cout << setprecision( places ) << root2 << '\n';

   return 0;
}
```

Fig. 11.17 Controlling precision of floating-point values (part 1 of 2).

```
Square root of 2 with precisions 0-9.
Precision set by the precision member function:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562

Precision set by the setprecision manipulator:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562
```

Fig. 11.17 Controlling precision of floating-point values (part 2 of 2).

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```
// fig11_18.cpp
// Demonstrating the width member function
#include <iostream.h>

int main()
{
   int w = 4;
   char string[ 10 ];

   cout << "Enter a sentence:\n";
   cin.width( 5 );

   while ( cin >> string ) {
      cout.width( w++ );
      cout << string << endl;
      cin.width( 5 );
   }

   return 0;
}
```

```
Enter a sentence:
This is a test of the width member function
This
   is
     a
   test
      of
      the
      widt
          h
        memb
           er
          func
           tion
```

Fig. 11.18 Demonstrating the `width` member function.

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```cpp
// Fig. 11.19: fig11_19.cpp
// Creating and testing user-defined, nonparameterized
// stream manipulators.
#include <iostream.h>

// bell manipulator (using escape sequence \a)
ostream& bell( ostream& output ) { return output << '\a'; }

// ret manipulator (using escape sequence \r)
ostream& ret( ostream& output ) { return output << '\r'; }

// tab manipulator (using escape sequence \t)
ostream& tab( ostream& output ) { return output << '\t'; }

// endLine manipulator (using escape sequence \n
// and the flush member function)
ostream& endLine( ostream& output )
{
   return output << '\n' << flush;
}

int main()
{
   cout << "Testing the tab manipulator:" << endLine
        << 'a' << tab << 'b' << tab << 'c' << endLine
        << "Testing the ret and bell manipulators:"
        << endLine << "..........";
   cout << bell;
   cout << ret << "-----" << endLine;
   return 0;
}
```

```
Testing the tab manipulator:
a       b       c
Testing the ret and bell manipulators:
-----.....
```

Fig. 11.19 Creating and testing user-defined, nonparameterized stream manipulators.

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

Format state flag	Description
`ios::skipws`	Skip whitespace characters on an input stream.
`ios::left`	Left justify output in a field. Padding characters appear to the right if necessary.
`ios::right`	Right justify output in a field. Padding characters appear to the left if necessary.
`ios::internal`	Indicate that a number's sign should be left justified in a field and a number's magnitude should be right justified in that same field (i.e., padding characters appear between the sign and the number).
`ios::dec`	Specify that integers should be treated as decimal (base 10) values.
`ios::oct`	Specify that integers should be treated as octal (base 8) values.
`ios::hex`	Specify that integers should be treated as hexadecimal (base 16) values.
`ios::showbase`	Specify that the base of a number to be output ahead of the number (a leading **`0`** for octals; a leading **`0x`** or **`0X`** for hexadecimals).
`ios::showpoint`	Specify that floating-point numbers should be output with a decimal point. This is normally used with **`ios::fixed`** to guarantee a certain number of digits to the right of the decimal point.
`ios::uppercase`	Specify that uppercase **`X`** should be used in the **`0X`** before a hexadecimal integer and that uppercase **`E`** should be used when representing a floating-point value in scientific notation.
`ios::showpos`	Specify that positive and negative numbers should be preceded by a **`+`** or **`-`** sign, respectively.
`ios::scientific`	Specify output of a floating-point value in scientific notation.
`ios::fixed`	Specify output of a floating-point value in fixed-point notation with a specific number of digits to the right of the decimal point.

Fig. 11.20 Format state flags.

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```
// Fig. 11.21: fig11_21.cpp
// Controlling the printing of trailing zeros and decimal
// points for floating-point values.
#include <iostream.h>
#include <iomanip.h>
#include <math.h>

int main()
{
   cout << "Before setting the ios::showpoint flag\n"
        << "9.9900 prints as: " << 9.9900
        << "\n9.9000 prints as: " << 9.9000
        << "\n9.0000 prints as: " << 9.0000
        << "\n\nAfter setting the ios::showpoint flag\n";
   cout.setf( ios::showpoint );
   cout << "9.9900 prints as: " << 9.9900
        << "\n9.9000 prints as: " << 9.9000
        << "\n9.0000 prints as: " << 9.0000 << endl;
   return 0;
}
```

```
Before setting the ios::showpoint flag
9.9900 prints as: 9.99
9.9000 prints as: 9.9
9.0000 prints as: 9

After setting the ios::showpoint flag
9.9900 prints as: 9.99000
9.9000 prints as: 9.90000
9.0000 prints as: 9.00000
```

Fig. 11.21 Controlling the printing of trailing zeros and decimal points with float values.

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```
// Fig. 11.22: fig11_22.cpp
// Left-justification and right-justification.
#include <iostream.h>
#include <iomanip.h>

int main()
{
   int x = 12345;

   cout << "Default is right justified:\n"
        << setw(10) << x << "\n\nUSING MEMBER FUNCTIONS"
        << "\nUse setf to set ios::left:\n" << setw(10);

   cout.setf( ios::left, ios::adjustfield );
   cout << x << "\nUse unsetf to restore default:\n";
   cout.unsetf( ios::left );
   cout << setw( 10 ) << x
        << "\n\nUSING PARAMETERIZED STREAM MANIPULATORS"
        << "\nUse setiosflags to set ios::left:\n"
        << setw( 10 ) << setiosflags( ios::left ) << x
        << "\nUse resetiosflags to restore default:\n"
        << setw( 10 ) << resetiosflags( ios::left )
        << x << endl;
   return 0;
}
```

```
Default is right justified:
     12345

USING MEMBER FUNCTIONS
Use setf to set ios::left:
12345
Use unsetf to restore default:
     12345

USING PARAMETERIZED STREAM MANIPULATORS
Use setiosflags to set ios::left:
12345
Use resetiosflags to restore default:
     12345
```

Fig. 11.22 Left-justification and right-justification.

```
// Fig. 11.23: fig11_23.cpp
// Printing an integer with internal spacing and
// forcing the plus sign.
#include <iostream.h>
#include <iomanip.h>

int main()
{
   cout << setiosflags( ios::internal | ios::showpos )
        << setw( 10 ) << 123 << endl;
   return 0;
}
```

```
+      123
```

Fig. 11.23 Printing an integer with internal spacing and forcing the plus sign.

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```
// Fig. 11.24: fig11_24.cpp
// Using the fill member function and the setfill
// manipulator to change the padding character for
// fields larger than the values being printed.
#include <iostream.h>
#include <iomanip.h>

int main()
{
   int x = 10000;

   cout << x << " printed as int right and left justified\n"
        << "and as hex with internal justification.\n"
        << "Using the default pad character (space):\n";
```

Fig. 11.24 Using the `fill` member function and the `setfill` manipulator to change the padding character for fields larger than the values being printed (part 1 of 2).

```
   cout.setf( ios::showbase );
   cout << setw( 10 ) << x << '\n';
   cout.setf( ios::left, ios::adjustfield );
   cout << setw( 10 ) << x << '\n';
   cout.setf( ios::internal, ios::adjustfield );
   cout << setw( 10 ) << hex << x;

   cout << "\n\nUsing various padding characters:\n";
   cout.setf( ios::right, ios::adjustfield );
   cout.fill( '*' );
   cout << setw( 10 ) << dec << x << '\n';
   cout.setf( ios::left, ios::adjustfield );
   cout << setw( 10 ) << setfill( '%' ) << x << '\n';
   cout.setf( ios::internal, ios::adjustfield );
   cout << setw( 10 ) << setfill( '^' ) << hex << x << endl;
   return 0;
}
```

```
10000 printed as int right and left justified
and as hex with internal justification.
Using the default pad character (space):
     10000
10000
0x    2710

Using various padding characters:
*****10000
10000%%%%%
0x^^^^2710
```

Fig. 11.24 Using the `fill` member function and the `setfill` manipulator to change the padding character for fields larger than the values being printed (part 2 of 2).

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```
// Fig. 11.25: fig11_25.cpp
// Using the ios::showbase flag
#include <iostream.h>
#include <iomanip.h>

int main()
{
   int x = 100;

   cout << setiosflags( ios::showbase )
        << "Printing integers preceded by their base:\n"
        << x << '\n'
        << oct << x << '\n'
        << hex << x << endl;
   return 0;
}
```

```
Printing integers preceded by their base:
100
0144
0x64
```

Fig. 11.25 Using the `ios::showbase` flag.

```
// Fig. 11.26: fig11_26.cpp
// Displaying floating-point values in system default,
// scientific, and fixed formats.
#include <iostream.h>

int main()
{
   double x = .001234567, y = 1.946e9;

   cout << "Displayed in default format:\n"
        << x << '\t' << y << '\n';
   cout.setf( ios::scientific, ios::floatfield );
   cout << "Displayed in scientific format:\n"
        << x << '\t' << y << '\n';
   cout.unsetf( ios::scientific );
   cout << "Displayed in default format after unsetf:\n"
        << x << '\t' << y << '\n';
   cout.setf( ios::fixed, ios::floatfield );
   cout << "Displayed in fixed format:\n"
        << x << '\t' << y << endl;
   return 0;
}
```

```
Displayed in default format:
0.00123457      1.946e+009
Displayed in scientific format:
1.234567e-003   1.946000e+009
Displayed in default format after unsetf:
0.00123457      1.946e+009
Displayed in fixed format:
0.001235        1946000000.000000
```

Fig. 11.26 Displaying floating-point values in system default, scientific, and fixed formats.

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```
// Fig. 11.27: fig11_27.cpp
// Using the ios::uppercase flag
#include <iostream.h>
#include <iomanip.h>
```

Fig. 11.27 Using the `ios::uppercase` flag (part 1 of 2).

```

int main()
{
   cout << setiosflags( ios::uppercase )
        << "Printing uppercase letters in scientific\n"
        << "notation exponents and hexadecimal values:\n"
        << 4.345e10 << '\n' << hex << 123456789 << endl;
   return 0;
}
```

```
Printing uppercase letters in scientific
notation exponents and hexadecimal values:
4.345E+010
75BCD15
```

Fig. 11.27 Using the `ios::uppercase` flag (part 2 of 2).

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```
// Fig. 11.28: fig11_28.cpp
// Demonstrating the flags member function.
#include <iostream.h>

int main()
{
   int i = 1000;
   double d = 0.0947628;

   cout << "The value of the flags variable is: "
        << cout.flags()
        << "\nPrint int and double in original format:\n"
        << i << '\t' << d << "\n\n";
   long originalFormat =
           cout.flags( ios::oct | ios::scientific );
```

Fig. 11.28 Demonstrating the `flags` member function (part 1 of 2).

```
   cout << "The value of the flags variable is: "
        << cout.flags()
        << "\nPrint int and double in a new format\n"
        << "specified using the flags member function:\n"
        << i << '\t' << d << "\n\n";
   cout.flags( originalFormat );
   cout << "The value of the flags variable is: "
        << cout.flags()
        << "\nPrint values in original format again:\n"
        << i << '\t' << d << endl;
   return 0;
}
```

```
The value of the flags variable is: 0
Print int and double in original format:
1000    0.0947628

The value of the flags variable is: 4040
Print int and double in a new format
specified using the flags member function:
1750    9.476280e-002

The value of the flags variable is: 0
Print values in original format again:
1000    0.0947628
```

Fig. 11.28 Demonstrating the `flags` member function (part 2 of 2).

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.

```
// Fig. 11.29: fig11_29.cpp
// Testing error states.
#include <iostream.h>

int main()
{
   int x;
   cout << "Before a bad input operation:"
        << "\ncin.rdstate(): " << cin.rdstate()
        << "\n    cin.eof(): " << cin.eof()
        << "\n   cin.fail(): " << cin.fail()
        << "\n    cin.bad(): " << cin.bad()
        << "\n   cin.good(): " << cin.good()
        << "\n\nExpects an integer, but enter a character: ";
   cin >> x;

   cout << "\nEnter a bad input operation:"
        << "\ncin.rdstate(): " << cin.rdstate()
        << "\n    cin.eof(): " << cin.eof()
        << "\n   cin.fail(): " << cin.fail()
        << "\n    cin.bad(): " << cin.bad()
        << "\n   cin.good(): " << cin.good() << "\n\n";

   cin.clear();

   cout << "After cin.clear()"
        << "\ncin.fail(): " << cin.fail()
        << "\ncin.good(): " << cin.good() << endl;
   return 0;
}
```

```
Before a bad input operation:
cin.rdstate(): 0
    cin.eof(): 0
   cin.fail(): 0
    cin.bad(): 0
   cin.good(): 1

Expects an integer, but enter a character: A

After a bad input operation:
cin.rdstate(): 2
    cin.eof(): 0
   cin.fail(): 2
    cin.bad(): 0
   cin.good(): 0

After cin.clear()
cin.fail(): 0
cin.good(): 1
```

Fig. 11.29 Testing error states.

© Copyright 1998 by Prentice Hall. All Rights Reserved.
For use only by instructors in courses for which C++ How to Program, Second Editon is the required textbook.