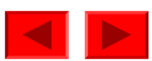
Lab7 - C++ Stream Input/Output

Outline

7.1 Introduction
7.2 Streams
  7.2.1 Classic Streams vs. Standard Streams
  7.2.2 iostream Library Header Files
  7.2.3 Stream Input/Output Classes and Objects
7.3 Stream Output
  7.3.1 Output of char * Variables
  7.3.2 Character Output using Member Function put
7.4 Stream Input
  7.4.1 get and getline Member Functions
  7.4.2 istream Member Functions peek, putback and ignore
  7.4.3 Type-Safe I/O
7.5 Unformatted I/O using read, write and gcount

© 2004 Benjamin X, Wang. All rights reserved.

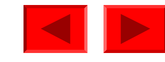


Lab7 - C++ Stream Input/Output

Outline

7.6 Introduction to Stream Manipulators
7.6.1 Integral Stream Base: dec, oct, hex and setbase
7.6.2 Floating-Point Precision (precision, setprecision)
7.6.3 Field Width (width, setw)
7.6.4 Programmer-Defined Manipulators
7.7 Stream Format States and Stream Manipulators
7.7.1 Trailing Zeros and Decimal Points (showpoint)
7.7.2 Justification (left, right and internal)
7.7.3 Padding (fill, setfill)
7.7.4 Integral Stream Base (dec, oct, hex, showbase)
7.7.5 Floating-Point Numbers; Scientific and Fixed Notation (scientific, fixed)
7.7.6 Uppercase/Lowercase Control (uppercase)
7.7.7 Specifying Boolean Format (boolalpha)
7.7.8 Setting and Resetting the Format State via Member-Function flags
7.8 Stream Error States
7.9 Tying an Output Stream to an Input Stream

© 2004 Benjamin X, Wang. All rights reserved.



7.1 Introduction

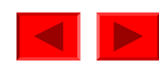
- Overview common I/O features
- C++ I/O
  - Object oriented
    - References, function overloading, operator overloading
  - Type safe
    - I/O sensitive to data type
    - Error if types do not match
  - User-defined and standard types
    - Makes C++ extensible

© 2004 Benjamin X, Wang. All rights reserved.

7.2 Streams

- Stream: sequence of bytes
  - Input: from device (keyboard, disk drive) to memory
  - Output: from memory to device (monitor, printer, etc.)
- I/O operations often bottleneck
  - Wait for disk drive/keyboard input
  - Low-level I/O
    - Unformatted (not convenient for people)
    - Byte-by-byte transfer
    - High-speed, high-volume transfers
  - High-level I/O
    - Formatted
    - Bytes grouped (into integers, characters, strings, etc.)
    - Good for most I/O needs

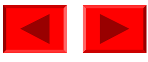
© 2004 Benjamin X, Wang. All rights reserved.



7.2.1 Classic Streams vs. Standard Streams

- Classic streams
  - Input/output **`char`**s (one byte)
  - Limited number of characters (ASCII)
- Standard stream libraries
  - Some languages need special alphabets
  - Unicode character set supports this
    - **`wchar_t`** character type
  - Can do I/O with Unicode characters

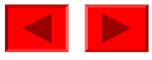
© 2004 Benjamin X, Wang. All rights reserved.



7.2.2 iostream Library Header Files

- **`iostream`** library
  - Has header files with hundreds of I/O capabilities
  - **`<iostream.h>`**
    - Standard input (**`cin`**)
    - Standard output (**`cout`**)
    - Unbuffered error (**`cerr`**)
    - Buffered error (**`clog`**)
  - **`<iomanip.h>`**
    - Formatted I/O with parameterized stream manipulators
  - **`<fstream.h>`**
    - File processing operations

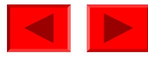
© 2004 Benjamin X, Wang. All rights reserved.



7.2.3 Stream Input/Output Classes and Objects

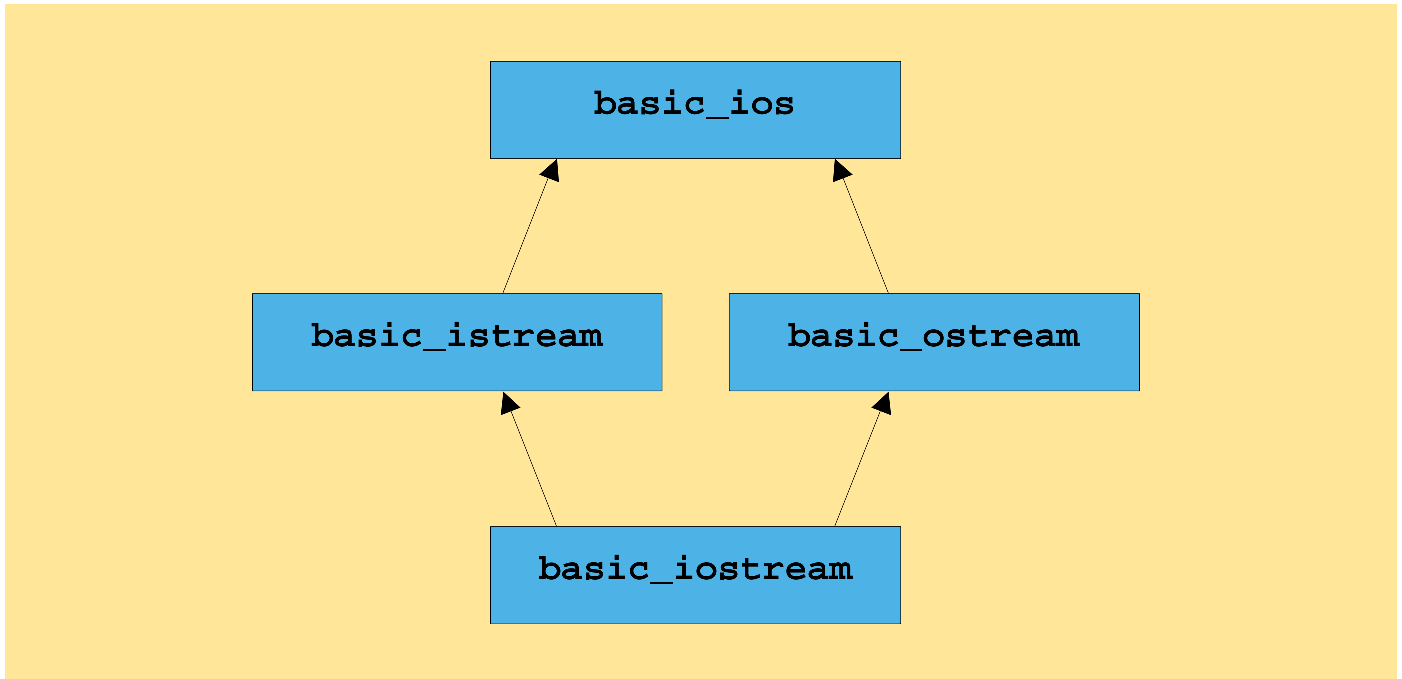
- **iostream** library has class templates for I/O
  - **basic_istream** (stream input)
  - **basic_ostream** (stream output)
  - **basic_iostream** (stream input and output)
- **typedef** declares alias for data type
  - **typedef Card *CardPtr;**
    - **CardPtr** synonym for **Card ***
  - **typedef**s **istream**, **ostream**, **iostream**
    - Allow **char** I/O
    - Use these **typedef**s in chapter

© 2004 Benjamin X, Wang. All rights reserved.

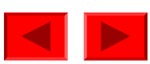


7.2.3 Stream Input/Output Classes and Objects

- Templates "derive" from **basic_ios**



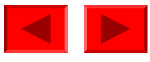
© 2004 Benjamin X, Wang. All rights reserved.



7.2.3 Stream Input/Output Classes and Objects

- `<<` and `>>`
  - Stream insertion and extraction operators
- `cin`
  - `istream` object
  - Connected to standard input (usually keyboard)
  - `cin >> grade;`
    - Compiler determines data type of grade
    - Calls proper overloaded operator
    - No extra type information needed

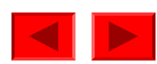
© 2004 Benjamin X, Wang. All rights reserved.



7.2.3 Stream Input/Output Classes and Objects

- **`cout`**
  - **`ostream`** object
  - Standard output (usually display screen)
  - **`cin << grade;`**
    - As with **`cin`**, no type information needed
- **`cerr`**, **`clog`**
  - **`ostream`** objects
  - Connected to standard error device
  - **`cerr`** outputs immediately
  - **`clog`** buffers output
    - Outputs when buffer full or flushed
    - Performance advantage (discussed in OS classes)

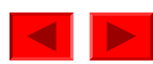
© 2004 Benjamin X, Wang. All rights reserved.



7.2.3 Stream Input/Output Classes and Objects

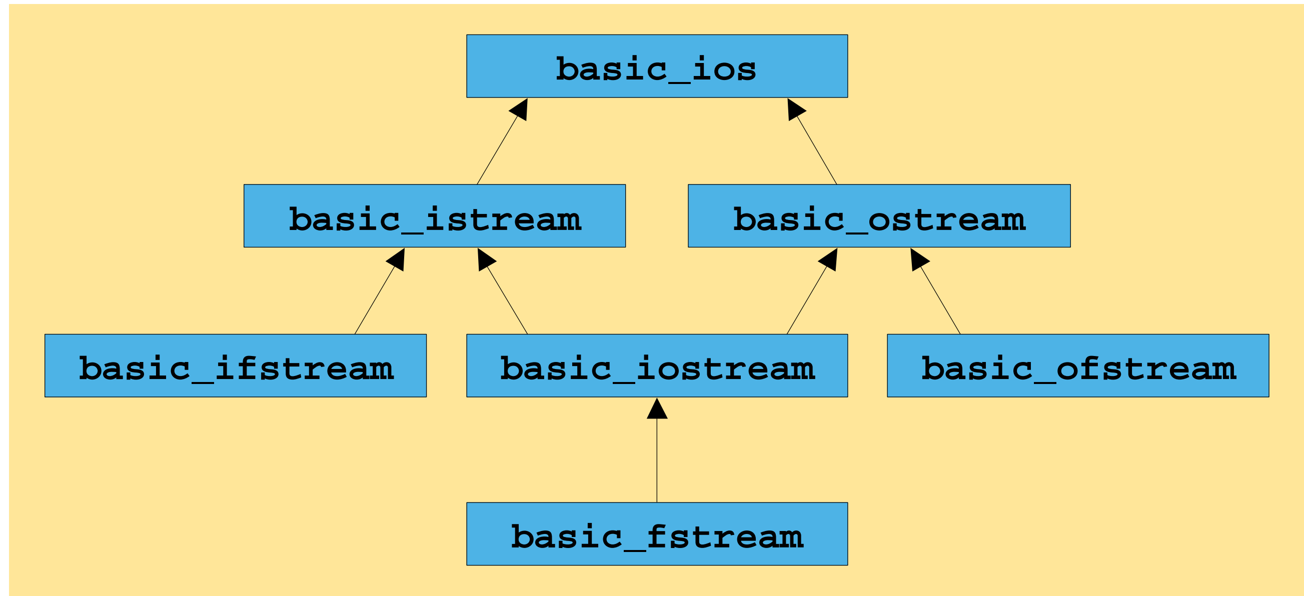
- C++ file processing similar
  - Class templates
    - **basic_ifstream** (file input)
    - **basic_ofstream** (file output)
    - **basic_fstream** (file I/O)
  - Specializations allow for **char** I/O
    - **typedef**s aliases for specializations
    - **ifstream**
    - **ofstream**
    - **fstream**

© 2004 Benjamin X, Wang. All rights reserved.

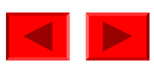


7.2.3 Stream Input/Output Classes and Objects

- Template hierarchy



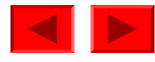
© 2004 Benjamin X, Wang. All rights reserved.



7.3 Stream Output

- Output
  - Use **`ostream`**
  - Formatted and unformatted
  - Standard data types (**`<<`**)
  - Characters (**`put`** function)
  - Integers (decimal, octal, hexadecimal)
  - Floating point numbers
    - Various precision, forced decimal points, scientific notation
  - Justified, padded data
  - Uppercase/lowercase control

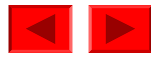
© 2004 Benjamin X, Wang. All rights reserved.



7.3.1 Output of char * Variables

- C++ determines data type automatically
  - Generally an improvement (over C)
  - Try to print value of a **`char *`**
    - Memory address of first character
- Problem
  - << overloaded to print null-terminated string
  - Solution: cast to **`void *`**
    - Use whenever printing value of a pointer
    - Prints as a hex (base 16) number

© 2004 Benjamin X, Wang. All rights reserved.



Outline

fig7_03.cpp (1 of 1)

fig7_03.cpp output (1 of 1)

```
1  // Fig. 7.3: fig7_03.cpp
2  // Printing the address stored in a char * variable.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    char *word = "test";
11
12    // display value of char *, then display value of char
13    // static_cast to void *
14    cout << "Value of word is: " << word << endl
15         << "Value of static_cast< void * >( word ) is: "
16         << static_cast< void * >( word ) << endl;
17
18    return 0;
19
20 } // end main
```

To print the value of the pointer, we must cast to a **void ***. Otherwise, the string is printed.

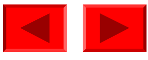
```
Value of word is: test
Value of static_cast< void *>( word ) is: 0046C070
```

© 2004 Benjamin X. Wang.
All rights reserved.

7.3.2 Character Output using Member Function put

- **`put`** function
  - Outputs characters
    - **`cout.put( 'A' );`**
  - May be cascaded
    - **`cout.put( 'A' ).put( '\n' );`**
    - Dot operator (**`.`**) evaluates left-to-right
  - Can use numerical (ASCII) value
    - **`cout.put( 65 );`**
    - Prints **`'A'`**

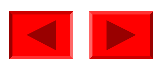
© 2004 Benjamin X, Wang. All rights reserved.



7.4 Stream Input

- Formatted and unformatted input
  - **`istream`**
- **`>>`** operator
  - Normally skips whitespace (blanks, tabs, newlines)
    - Can change this
  - Returns **`0`** when EOF encountered
    - Otherwise, returns reference to object
    - **`cin >> grade`**
  - State bits set if errors occur
    - Discussed in 7.7 and 7.8

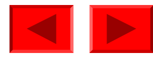
© 2004 Benjamin X, Wang. All rights reserved.

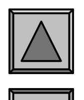


7.4.1 get and getline Member Functions

- **`get`** function
  - **`cin.get()`**
  - Returns one character from stream (even whitespace)
    - Returns **`EOF`** if end-of-file encountered
- End-of-file
  - Indicates end of input
    - *ctrl-z* on IBM-PCs
    - *ctrl-d* on UNIX and Macs
  - **`cin.eof()`**
    - Returns **`1`** (**`true`**) if EOF has occurred

© 2004 Benjamin X, Wang. All rights reserved.





Outline

fig7_04.cpp (1 of 2)

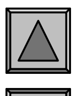
```
1  // Fig. 7.4: fig7_04.cpp
2  // Using member functions get, put and eof.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    int character; // use int, because char cannot represent EOF
12
13    // prompt user to enter line of text
14    cout << "Before input, cin.eof() is "
15         << "Enter a sentence followed by
16
17    // use get to read each character; us
18    while ( ( character = cin.get() ) !=
19       cout.put( character );
20
21    // display end-of-file character
22    cout << "\nEOF in this system is: " << character << endl;
23    cout << "After input, cin.eof() is " << cin.eof() << endl;
24
25    return 0;
```

Function **get** (with no arguments) returns a single character input, unless **EOF** encountered.

© 2004 Benjamin X. Wang.
All rights reserved.

```
26
27 } // end main
```

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

EOF in this system is: -1
After input cin.eof() is 1
```

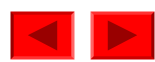


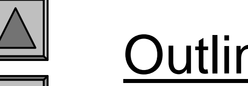
© 2004 Benjamin X. Wang.
All rights reserved.

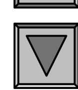
7.4.1 get and getline Member Functions

- **`get(charRef)`**
  - With character reference argument
  - Gets one character, stores in **`charRef`**
    - Returns reference to **`istream`**
    - If EOF, returns **`-1`**
- **`get(charArray, size, delimiter)`**
  - Reads until **`size-1`** characters read, or delimiter encountered
    - Default delimiter **`'\n'`**
    - Delimiter stays in input stream
      - Can remove with **`cin.get()`** or **`cin.ignore()`**
  - Makes array null-terminated

© 2004 Benjamin X, Wang. All rights reserved.





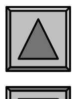



```
1  // Fig. 7.5: fig7_05.cpp
2  // Contrasting input of a string via cin and cin.get.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    // create two char arrays, each with 80 elements
12    const int SIZE = 80;
13    char buffer1[ SIZE ];
14    char buffer2[ SIZE ];
15
16    // use cin to input characters
17    cout << "Enter a sentence:" << endl;
18    cin >> buffer1;
19
20    // display buffer1 contents
21    cout << "\nThe string read with cin was
22         << buffer1 << endl << endl;
23
24    // use cin.get to input characters into buffer2
25    cin.get( buffer2, SIZE );
```

`cin` will only read until the first whitespace.

No delimiter specified, so the default (**`\n`**) is used.

© 2004 Benjamin X. Wang.
All rights reserved.



Outline

```
26
27    // display buffer2 contents
28    cout << "The string read with cin.get was:" << endl
29         << buffer2 << endl;
30
31    return 0;
32
33 } // end main
```

```
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
```

fig7_05.cpp (2 of 2)

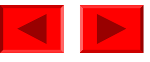
fig7_05.cpp output (1 of 1)

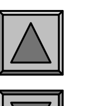
© 2004 Benjamin X. Wang.
All rights reserved.

7.4.1 get and getline Member Functions

- **`getline(array, size, delimiter)`**
  - Like last version of get
  - Reads **`size-1`** characters, or until delimiter found
    - Default **`\n`**
  - Removes delimiter from input stream
  - Puts null character at end of array

© 2004 Benjamin X, Wang. All rights reserved.





Outline

fig7_06.cpp
(1 of 1)

```
1  // Fig. 7.6: fig7_06.cpp
2  // Inputting characters using cin member function getline.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    const int SIZE = 80;
12    char buffer[ SIZE ]; // create array of 80 characters
13
14    // input characters in buffer via cin function getline
15    cout << "Enter a sentence:" << endl;
16    cin.getline( buffer, SIZE );
17
18    // display buffer contents
19    cout << "\nThe sentence entered is:" << endl << buffer << endl;
20
21    return 0;
22
23 } // end main
```

© 2004 Benjamin X. Wang.
All rights reserved.

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

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

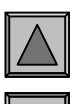


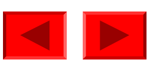
fig7_06.cpp
output (1 of 1)

© 2004 Benjamin X. Wang.
All rights reserved.

7.4.2 istream Member Functions peek, putback and ignore

- **`ignore()`**
  - Discards characters from stream (default 1)
  - Stops discarding once delimiter found
    - Default delimiter **`EOF`**
- **`putback()`**
  - Puts character obtained by **`get()`** back on stream
- **`peek()`**
  - Returns next character in stream, but does not remove

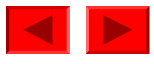
© 2004 Benjamin X, Wang. All rights reserved.



7.4.3 Type-Safe I/O

- **<<** and **>>**
  - Overloaded to accept data of specific types
- If unexpected data processed
  - Error bits set
  - User can test bits to see if I/O failed
    - More in section 7.8

© 2004 Benjamin X, Wang. All rights reserved.



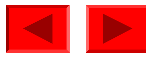
7.5 Unformatted I/O using read, write and gcount

- Unformatted I/O
  - **`read`** (member of **`istream`**)
    - Input raw bytes into character array
    - If not enough characters read, **`failbit`** set
    - **`gcount()`** returns number of characters read by last operation
  - **`write`** (**`ostream`**)
    - Output bytes from character array
      - Stops when null character found

```
char buffer[] = "HAPPY BIRTHDAY";
cout.write( buffer, 10 );
```

  - Outputs first 10 characters

© 2004 Benjamin X, Wang. All rights reserved.



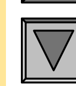


fig7_07.cpp
(1 of 1)

```
1  // Fig. 7.7: fig7_07.cpp
2  // Unformatted I/O using read, gcount and write.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    const int SIZE = 80;
12    char buffer[ SIZE ]; // create array of 80 characters
13
14    // use function read to input characters into buffer
15    cout << "Enter a sentence:" << endl;
16    cin.read( buffer, 20 );
17
18    // use functions write and gcount to display buffer characters
19    cout << endl << "The sentence entered was:" << endl;
20    cout.write( buffer, cin.gcount() );
21    cout << endl;
22
23    return 0;
24
25 } // end main
```

Get 20 characters from input stream. Display the proper number of characters using **`write`** and **`gcount`**.

© 2004 Benjamin X. Wang.
All rights reserved.

```
Enter a sentence:
Using the read, write, and gcount member functions
The sentence entered was:
Using the read, writ
```

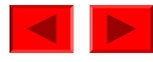

fig7_07.cpp
output (1 of 1)

© 2004 Benjamin X. Wang.
All rights reserved.

7.6 Introduction to Stream Manipulators

- Stream manipulators perform formatting tasks
  - Field widths
  - Precisions
  - Format flags
  - Fill character in fields
  - Flushing streams
  - Inserting newline in output stream
  - Skipping whitespace in input stream

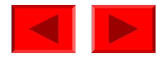
© 2004 Benjamin X, Wang. All rights reserved.



7.6.1 Integral Stream Base: dec, oct, hex and setbase

- Integers normally base 10 (decimal)
  - Stream manipulators to change base
    - **`hex`** (base 16)
    - **`oct`** (base 8)
    - **`dec`** (resets to base 10)
    - **`cout << hex << myInteger`**
  - **`setbase(newBase)`**
    - One of 8, 10, or 16
  - Base remains same until explicitly changed
- Parameterized stream manipulators
  - Use header **`<iomanip>`**
  - Take argument (like **`setbase`**)

© 2004 Benjamin X, Wang. All rights reserved.



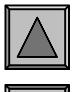
Outline

fig7_08.cpp
(1 of 2)

```
1  // Fig. 7.8: fig7_08.cpp
2  // Using stream manipulators hex, oct, dec and setbase.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8  using std::hex;
9  using std::dec;
10 using std::oct;
11
12 #include <iomanip>
13
14 using std::setbase;
15
16 int main()
17 {
18    int number;
19
20    cout << "Enter a decimal number: ";
21    cin >> number; // input number
22
23    // use hex stream manipulator to show hexadecimal number
24    cout << number << " in hexadecimal is: " << hex
25         << number << endl;
```

Note usage of stream manipulator.

© 2004 Benjamin X. Wang.
All rights reserved.



Outline

fig7_08.cpp (2 of 2)

fig7_08.cpp output (1 of 1)

```
26
27    // use oct stream manipulator to show octal number
28    cout << dec << number << " in octal is: "
29         << oct << number << endl;
30
31    // use setbase stream manipulator to show decimal number
32    cout << setbase( 10 ) << number << " in decimal is: "
33         << number << endl;
34
35    return 0;
36
37 } // end main
```

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

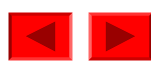
`setbase` is a parameterized stream manipulator (it takes an argument).

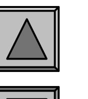
© 2004 Benjamin X. Wang.
All rights reserved.

7.6.2 Floating-Point Precision (precision, setprecision)

- Set precision of floating point numbers
  - Number of digits to right of decimal
  - **`setprecision`** stream manipulator
    - Pass number of decimal points
    - **`cout << setprecision(5)`**
  - **`precision`** member function
    - **`cout.precision(newPrecision)`**
    - With no arguments, returns current precision
  - Settings remain until changed explicitly

© 2004 Benjamin X, Wang. All rights reserved.





Outline

fig7_09.cpp
(1 of 2)

```
1   // Fig. 7.9: fig7_09.cpp
2   // Controlling precision of floating-point values.
3   #include <iostream>
4
5   using std::cout;
6   using std::cin;
7   using std::endl;
8   using std::fixed;
9
10  #include <iomanip>
11
12  using std::setprecision;
13
14  #include <cmath>  // sqrt prototype
15
16  int main()
17  {
18     double root2 = sqrt( 2.0 ); // calculate square root of 2
19     int places;
20
21     cout << "Square root of 2 with
22          << "Precision set by ios_
23          << "precision:" << endl;
24
25     cout << fixed; // use fixed precision
```

Use fixed precision, not scientific notation (more details in 12.7).

© 2004 Benjamin X. Wang.
All rights reserved.

Outline

fig07_09.cpp

```
26
27    // display square root using ios_base function precision
28    for ( places = 0; places <= 9; places++ ) {
29       cout.precision( places );
30       cout << root2 << endl;
31    }
32
33    cout << "\nPrecision set by stream-manipulator "
34         << "setprecision:" << endl;
35
36    // set precision for each digit, then display square root
37    for ( places = 0; places <= 9; places++ )
38       cout << setprecision( places ) << root2 << endl;
39
40    return 0;
41
42 } // end main
```

Note format of function **`precision`** and parameterized stream manipulator **`setprecision`**.

© 2004 Benjamin X. Wang.
All rights reserved.

fig7_09.cpp
output (1 of 1)

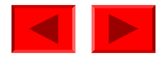
```
Square root of 2 with precisions 0-9.
Precision set by ios_base member-function precision:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562

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

© 2004 Benjamin X. Wang.
All rights reserved.

7.6.3 Field Width (width, setw)

- `width` member function (base class `ios_base`)
  - `cin.width(5)`
  - Sets field width
    - Number of character positions for output
    - Maximum number of characters that should be input
  - Returns previous width
  - Fill characters/Padding
    - Used when output too small for width
    - Large outputs are printed (not truncated)
  - Can also use `setw` stream manipulator
- When reading to `char` arrays
  - Reads 1 less character (leave room for null)

© 2004 Benjamin X, Wang. All rights reserved.



Outline

fig7_10.cpp
(1 of 2)

```
1   // Fig. 7.10: fig7_10.cpp
2   // Demonstrating member function width.
3   #include <iostream>
4
5   using std::cout;
6   using std::cin;
7   using std::endl;
8
9   int main()
10  {
11     int widthValue = 4;
12     char sentence[ 10 ];
13
14     cout << "Enter a sentence:" <<
15     cin.width( 5 ); // input only 5
16
17     // set field width, then display characters based on that width
18     while ( cin >> sentence ) {
19        cout.width( widthValue++ );
20        cout << sentence << endl;
21        cin.width( 5 ); // input 5 more characters from sentence
22     } // end while
23
24     return 0;
```

Reads up to 4 characters, stops when whitespace read.

Increment the output width.

© 2004 Benjamin X. Wang.
All rights reserved.

```
25
26  } // end main
```

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

fig7_10.cpp (2 of 2)

fig7_10.cpp output (1 of 1)

© 2004 Benjamin X. Wang.
All rights reserved.

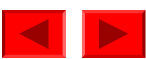
7.6.4 Programmer-Defined Manipulators

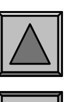
- User-defined stream manipulators
  - Nonparameterized
  - Example

```
ostream& bell( ostream& output )
{
  return output << '\a';  // issue system beep
}
```

  - `\a` - bell
  - `\r` - carriage return
  - `\t` - tab

© 2004 Benjamin X, Wang. All rights reserved.





Outline

```
1  // Fig. 7.11: fig7_11.cpp
2  // Creating and testing programmer-defined, nonparameterized
3  // stream manipulators.
4  #include <iostream>
5
6  using std::ostream;
7  using std::cout;
8  using std::flush;
9
10 // bell manipulator (using escape sequence \a)
11 ostream& bell( ostream& output )
12 {
13    return output << '\a';  // issue system beep
14 }
15
16 // carriageReturn manipulator (using escape sequence \r)
17 ostream& carriageReturn( ostream& output )
18 {
19    return output << '\r';  // issue carriage return
20 }
21
```

fig7_11.cpp
(1 of 3)

© 2004 Benjamin X. Wang.
All rights reserved.

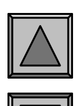
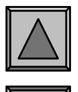


fig7_11.cpp
(2 of 3)

```
22 // tab manipulator (using escape sequence \t)
23 ostream& tab( ostream& output )
24 {
25    return output << '\t';  // issue tab
26 }
27
28 // endLine manipulator (using escape sequence \n and member
29 // function flush)
30 ostream& endLine( ostream& output )
31 {
32    return output << '\n' << flush;  // issue end of line
33 }
34
35 int main()
36 {
37    // use tab and endLine manipulators
38    cout << "Testing the tab manipulator:" << endLine
39         << 'a' << tab << 'b' << tab << 'c' << endLine;
40
41    cout << "Testing the carriageReturn and bell manipulators:"
42         << endLine << "..........";
43
44    cout << bell; // use bell manipulator
45
```

© 2004 Benjamin X. Wang.
All rights reserved.




```
46     // use carriageReturn and endLine manipulators
47     cout << carriageReturn << "-----" << endLine;
48
49     return 0;
50
51  } // end main
```

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

fig7_11.cpp (3 of 3)

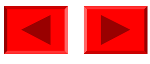
fig7_11.cpp output (1 of 1)

© 2004 Benjamin X. Wang.
All rights reserved.

7.7 Stream Format States and Stream Manipulators

- Many stream manipulators for formatting
  - Coming up next
  - All inherit from **`ios_base`**

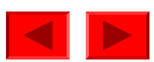
© 2004 Benjamin X, Wang. All rights reserved.

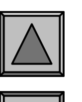


7.7.1 Trailing Zeros and Decimal Points (showpoint)

- **`showpoint`**
  - Forces decimal number to print with trailing zeros
  - For decimal number 79.0
    - 79 without **`showpoint`**
    - 79.000000 with **`showpoint`** (up to level of precision)
  - Reset with **`noshowpoint`**

© 2004 Benjamin X, Wang. All rights reserved.



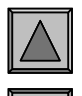


Outline

```
1  // Fig. 7.13: fig7_13.cpp
2  // Using showpoint to control the printing of
3  // trailing zeros and decimal points for doubles.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8  using std::showpoint;
9
10 int main()
11 {
12    // display double values with default stream format
13    cout << "Before using showpoint" << endl
14         << "9.9900 prints as: " << 9.9900 << endl
15         << "9.9000 prints as: " << 9.9000 << endl
16         << "9.0000 prints as: " << 9.0000 << endl << endl;
17
18    // display double value after showpoint
19    cout << showpoint
20         << "After using showpoint" << endl
21         << "9.9900 prints as: " << 9.9900 << endl
22         << "9.9000 prints as: " << 9.9000 << endl
23         << "9.0000 prints as: " << 9.0000 << endl;
24
25    return 0;
```

fig7_13.cpp
(1 of 2)

© 2004 Benjamin X. Wang.
All rights reserved.




```
26
27  } // end main
```

```
Before using showpoint
9.9900 prints as: 9.99
9.9000 prints as: 9.9
9.0000 prints as: 9

After using showpoint
9.9900 prints as: 9.99000
9.9000 prints as: 9.90000
9.0000 prints as: 9.00000
```

Without **`showpoint`**, trailing zeros are removed.

fig7_13.cpp (2 of 2)

fig7_13.cpp output (1 of 1)

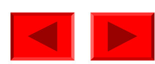
© 2004 Benjamin X. Wang.
All rights reserved.

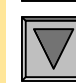
7.7.2 Justification (left, right and internal)

- **`left`** stream manipulator
  - Left-justified, padding to right
  - **`Right`** stream manipulator
    - Right-justified, padding to left
  - Can set padding/fill character
    - Next section
- **`internal`**
  - Number's sign left-justified
  - Number's value right-justified

    `+          123`
  - **`showpos`** forces sign to print
    - Remove with **`noshowpos`**

© 2004 Benjamin X, Wang. All rights reserved.





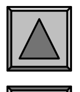
Outline

fig7_14.cpp
(1 of 2)

```
1   // Fig. 7.14: fig7_14.cpp
2   // Demonstrating left justification and right justification.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7   using std::left;
8   using std::right;
9
10  #include <iomanip>
11
12  using std::setw;
13
14  int main()
15  {
16     int x = 12345;
17
18     // display x right justified (default)
19     cout << "Default is right justified:" << endl
20          << setw( 10 ) << x;
21
22     // use left manipulator to display x left justified
23     cout << "\n\nUse std::left to left justify x:\n"
24          << left << setw( 10 ) << x;
25
```

Right- and left-justify **x** (within a width of 10).

© 2004 Benjamin X. Wang.
All rights reserved.



Outline

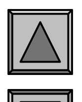
```
26     // use right manipulator to display x right justified
27     cout << "\n\nUse std::right to right justify x:\n"
28          << right << setw( 10 ) << x << endl;
29
30     return 0;
31
32  } // end main
```

```
Default is right justified:
     12345

Use std::left to left justify x:
12345

Use std::right to right justify x:
     12345
```

fig7_14.cpp (2 of 2)

fig7_14.cpp output (1 of 1)

© 2004 Benjamin X. Wang.
All rights reserved.



Outline

```
1   // Fig. 7.15: fig7_15.cpp
2   // Printing an integer with internal spacing and plus sign.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7   using std::internal;
8   using std::showpos;
9
10  #include <iomanip>
11
12  using std::setw;
13
14  int main()
15  {
16     // display value with internal spacing and plus sign
17     cout << internal << showpos << setw( 10 ) << 123 << endl;
18
19     return 0;
20
21  } // end main
```

Note use of **`internal`** and **`showpos`**.

```
+      123
```

fig7_15.cpp (1 of 1)

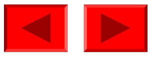
fig7_15.cpp output (1 of 1)

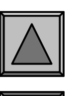
© 2004 Benjamin X. Wang.
All rights reserved.

7.7.3 Padding (fill, setfill)

- Set fill character used in padding
  - **`fill`** member function
    - **`cout.fill('*')`**
  - **`setfill`** stream manipulator
    - **`setfill( '^' )`**

© 2004 Benjamin X, Wang. All rights reserved.





Outline

```
1  // Fig. 7.16: fig7_16.cpp
2  // Using member-function fill and stream-manipulator setfill
3  // to change the padding character for fields larger the
4  // printed value.
5  #include <iostream>
6
7  using std::cout;
8  using std::endl;
9  using std::showbase;
10 using std::left;
11 using std::right;
12 using std::internal;
13 using std::hex;
14 using std::dec;
15
16 #include <iomanip>
17
18 using std::setw;
19 using std::setfill;
20
```

fig7_16.cpp (1 of 3)

© 2004 Benjamin X. Wang.
All rights reserved.

```
21 int main()
22 {
23    int x = 10000;
24
25    // display x
26    cout << x << " printed as int right and left justified\n"
27         << "and as hex with internal justification.\n"
28         << "Using the default pad character (space):" << endl;
29
30    // display x with plus sign
31    cout << showbase << setw( 10 ) << x << endl;
32
33    // display x with left justification
34    cout << left << setw( 10 ) << x << endl;
35
36    // display x as hex with internal justification
37    cout << internal << setw( 10 ) << hex << x << endl << endl;
38
39    cout << "Using various padding c
40
41    // display x using padded charac
42    cout << right;
43    cout.fill( '*' );
44    cout << setw( 10 ) << dec << x << endl;
45
```

Note use of member function **fill**.

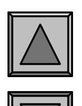
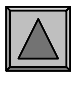


fig7_16.cpp
(2 of 3)

© 2004 Benjamin X. Wang.
All rights reserved.



Outline

```
46     // display x using padded characters (left justification)
47     cout << left << setw( 10 ) << setfill( '%' ) << x << endl;
48
49     // display x using padded characters (internal justification)
50     cout << internal << setw( 10 ) << setfill( '^' ) << hex
51          << x << endl;
52
53     return 0;
54
55  } // end main
```

```
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
```

fig7_16.cpp (3 of 3)

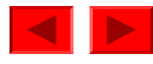
fig7_16.cpp output (1 of 1)

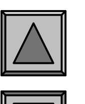
© 2004 Benjamin X. Wang.
All rights reserved.

7.7.4 Integral Stream Base (dec, oct, hex, showbase)

- Print integer in various bases
  - **dec**, **oct**, **hex**
- Stream extraction
  - Decimal numbers default
  - Preceding **0** for octal
  - Preceding **0x** or **0X** for hex
- **showbase**
  - Forces base of number to be shown
  - Remove with **noshowbase**

© 2004 Benjamin X, Wang. All rights reserved.






```
1  // Fig. 7.17: fig7_17.cpp
2  // Using stream-manipulator showbase.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::showbase;
8  using std::oct;
9  using std::hex;
10
11 int main()
12 {
13    int x = 100;
14
15    // use showbase to show numbe
16    cout << "Printing integers pr                                l
17         << showbase;
18
19    cout << x << endl;        // print decimal value
20    cout << oct << x << endl; // print octal value
21    cout << hex << x << endl; // print hexadecimal value
22
23    return 0;
24
25 } // end main
```

Forces numbers to be printed with a preceding **0** (if octal) or **0x** (if hexadecimal).

fig7_17.cpp
(1 of 1)

© 2004 Benjamin X. Wang.
All rights reserved.

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

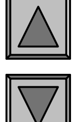


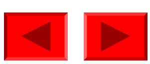
fig7_17.cpp
output (1 of 1)

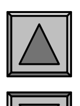
© 2004 Benjamin X. Wang.
All rights reserved.

7.7.5 Floating-Point Numbers; Scientific and Fixed Notation (scientific, fixed)

- Stream manipulator **`scientific`**
  - Forces scientific notation
    - **`1.946000e+009`**
- Stream manipulator **`fixed`**
  - Forces fixed point format
  - Prints number of decimals specified by precision
    - **`1946000000.000000`**
- If no manipulator specified
  - Format of number determines how it appears

© 2004 Benjamin X, Wang. All rights reserved.





Outline

```
1  // Fig. 7.18: fig7_18.cpp
2  // Displaying floating-point values in system default,
3  // scientific and fixed formats.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8  using std::scientific;
9  using std::fixed;
10
11 int main()
12 {
13    double x = 0.001234567;
14    double y = 1.946e9;
15
16    // display x and y in default format
17    cout << "Displayed in default format:" << endl
18         << x << '\t' << y << endl;
19
20    // display x and y in scientific format
21    cout << "\nDisplayed in scientific format:" << endl
22         << scientific << x << '\t' << y << endl;
23
```

Note differing initializations, and use of the **`scientific`** stream manipulator.

fig7_18.cpp (1 of 2)

© 2004 Benjamin X. Wang.
All rights reserved.

```
24     // display x and y in fixed format
25     cout << "\nDisplayed in fixed format:" << endl
26          << fixed << x << '\t' << y << endl;
27
28     return 0;
29
30  } // end main
```

```
Displayed in default format:
0.00123457      1.946e+009

Displayed in scientific format:
1.234567e-003   1.946000e+009

Displayed in fixed format:
0.001235        1946000000.000000
```

Note difference between the default, fixed, and scientific formats.

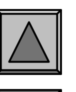


fig7_18.cpp (2 of 2)

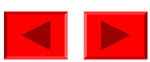
fig7_18.cpp output (1 of 1)

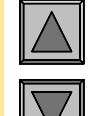
© 2004 Benjamin X. Wang.
All rights reserved.

7.7.6 Uppercase/Lowercase Control (uppercase)

- Stream manipulator **`uppercase`**
  - Uppercase E in scientific notation
    - **`1E10`**
  - Uppercase X in hex notation and uppercase hex letters
    - **`0XABCD`**
  - By default, lowercase
  - Reset with **`nouppercase`**

© 2004 Benjamin X, Wang. All rights reserved.





Outline

fig7_19.cpp
output (1 of 1)

```
1   // Fig. 7.19: fig7_19.cpp
2   // Stream-manipulator uppercase.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7   using std::uppercase;
8   using std::hex;
9
10  int main()
11  {
12     cout << "Printing uppercase letters in scientific" << endl
13          << "notation exponents and hexadecimal values:" << endl;
14
15     // use std:uppercase to display uppercase letters;
16     // use std::hex to display hexadecimal values
17     cout << uppercase << 4.345e10 << endl << hex << 123456789
18          << endl;
19
20     return 0;
21
22  } // end main
```

Force uppercase format.

© 2004 Benjamin X. Wang.
All rights reserved.

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

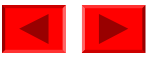
fig7_19.cpp
output (1 of 1)

© 2004 Benjamin X. Wang.
All rights reserved.

7.7.7 Specifying Boolean Format (boolalpha)

- Data type **bool**
  - Values **true** or **false**
  - Outputs **0** (**false**) or **1** (**true**) when used with **<<**
    - Displayed as integers
- Stream manipulator **boolalpha**
  - Display strings **"true"** and **"false"**
  - Reset with **noboolalpha**

© 2004 Benjamin X, Wang. All rights reserved.



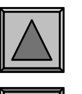
Outline

fig7_20.cpp
(1 of 2)

```
1  // Fig. 7.20: fig7_20.cpp
2  // Demonstrating stream-manipulators boolalpha and noboolalpha.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::cin;
8  using std::boolalpha;
9  using std::noboolalpha;
10
11 int main()
12 {
13    bool booleanValue = true;
14
15    // display default true booleanValue
16    cout << "booleanValue is " << booleanValue << endl;
17
18    // display booleanValue after using boolalpha
19    cout << "booleanValue (after using boolalpha) is "
20         << boolalpha << booleanValue << endl << endl;
21
22    cout << "switch booleanValue and use noboolalpha" << endl;
23    booleanValue = false; // change booleanValue
24    cout << noboolalpha << endl; // use noboolalpha
25
```

`bool` variables can be `false` or `true`. Note use of the `boolalpha` stream manipulator.

© 2004 Benjamin X. Wang.
All rights reserved.



Outline

```
26     // display default false booleanValue after using noboolalpha
27     cout << "booleanValue is " << booleanValue << endl;
28
29     // display booleanValue after using boolalpha again
30     cout << "booleanValue (after using boolalpha) is "
31          << boolalpha << booleanValue << endl;
32
33     return 0;
34
35  } // end main
```

```
booleanValue is 1
booleanValue (after using boolalpha) is true

switch booleanValue and use noboolalpha

booleanValue is 0
booleanValue (after using boolalpha) is false
```

fig7_20.cpp (2 of 2)

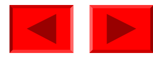
fig7_20.cpp output (1 of 1)

© 2004 Benjamin X. Wang.
All rights reserved.

7.7.8 Setting and Resetting the Format State via Member-Function flags

- Can save/restore format states
  - After apply many changes, may want to restore original
- Member function **`flags`**
  - **`cout.flags()`**
  - With no argument
    - Returns current state as **`fmtflags`** object
      - Namespace **`ios_base`**
    - Represents format state
  - With **`fmtflags`** argument
    - Sets state
    - Returns previous state

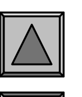
© 2004 Benjamin X, Wang. All rights reserved.



```
1  // Fig. 7.21: fig7_21.cpp
2  // Demonstrating the flags member function.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::oct;
8  using std::scientific;
9  using std::showbase;
10 using std::ios_base;
11
12 int main()
13 {
14    int integerValue = 1000;
15    double doubleValue = 0.0947628;
16
17    // display flags value, int and double values (original format)
18    cout << "The value of the flags variable is: " << cout.flags()
19         << "\nPrint int and double in original format:\n"
20         << integerValue << '\t' << doubleValue << endl << endl;
21
22    // use cout flags function to save original format
23    ios_base::fmtflags originalFormat = cout.flags();
24    cout << showbase << oct << scientific; // change format
25
```

Save original format using function **flags**.

© 2004 Benjamin X. Wang.
All rights reserved.



Outline

fig7_21.cpp
(2 of 2)

```
26     // display flags value, int and double values (new format)
27     cout << "The value of the flags variable is: " << cout.flags()
28          << "\nPrint int and double in a new format:\n"
29          << integerValue << '\t' << doubleValue << endl << endl;
30
31     cout.flags( originalFormat ); // restore format
32
33     // display flags value, int and double values (original format)
34     cout << "The restored value of the flags variable is: "
35          << cout.flags()
36          << "\nPrint values in original format again:\n"
37          << integerValue << '\t' << doubleValue << endl;
38
39     return 0;
40
41  } // end main
```

Restore original format.

© 2004 Benjamin X. Wang.
All rights reserved.

```
The value of the flags variable is: 513
Print int and double in original format:
1000    0.0947628

The value of the flags variable is: 012011
Print int and double in a new format:
01750   9.476280e-002

The restored value of the flags variable is: 513
Print values in original format again:
1000    0.0947628
```

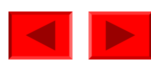
fig7_21.cpp
output (1 of 1)

© 2004 Benjamin X. Wang.
All rights reserved.

7.8 Stream Error States

- Test state of stream using bits
  - **`eofbit`** set when EOF encountered
    - Function **`eof`** returns **`true`** if **`eofbit`** set
    - **`cin.eof()`**
  - **`failbit`** set when error occurs in stream
    - Data not lost, error recoverable
    - Function **`fail`** returns **`true`** if set
  - **`badbit`** set when data lost
    - Usually nonrecoverable
    - Function **`bad`**
  - **`goodbit`** set when **`badbit`**, **`failbit`** and **`eofbit`** off
    - Function **`good`**

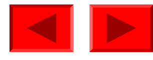
© 2004 Benjamin X, Wang. All rights reserved.



7.8 Stream Error States

- Member functions
  - **`rdstate()`**
    - Returns error state of stream
    - Can test for **`goodbit`**, **`badbit`**, etc.
    - Better to test using **`good()`**, **`bad()`**
  - **`clear()`**
    - Default argument **`goodbit`**
    - Sets stream to "good" state, so I/O can continue
    - Can pass other values
      - **`cin.clear( ios::failbit )`**
      - Sets **`failbit`**
      - Name "clear" seems strange, but correct

© 2004 Benjamin X, Wang. All rights reserved.



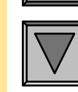
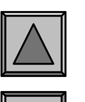


fig7_22.cpp
(1 of 2)

```
1  // Fig. 7.22: fig7_22.cpp
2  // Testing error states.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::cin;
8
9  int main()
10 {
11    int integerValue;
12
13    // display results of cin functions
14    cout << "Before a bad input operation:"
15         << "\ncin.rdstate(): " << cin.rdstate()
16         << "\n    cin.eof(): " << cin.eof()
17         << "\n   cin.fail(): " << cin.fail()
18         << "\n    cin.bad(): " << cin.bad()
19         << "\n   cin.good(): " << cin.good()
20         << "\n\nExpects an integer, but enter a character: ";
21
22    cin >> integerValue; // enter character value
23    cout << endl;
24
```

Output the original states using the member functions.

© 2004 Benjamin X. Wang.
All rights reserved.



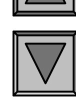
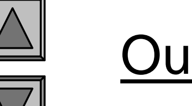
Outline

fig7_22.cpp (2 of 2)

```
25     // display results of cin functions after bad input
26     cout << "After a bad input operation:"
27          << "\ncin.rdstate(): " << cin.rdstate()
28          << "\n    cin.eof(): " << cin.eof()
29          << "\n   cin.fail(): " << cin.fail()
30          << "\n    cin.bad(): " << cin.bad()
31          << "\n   cin.good(): " << cin.good() << endl << endl;
32
33     cin.clear(); // clear stream
34
35     // display results of cin functions after clearing cin
36     cout << "After cin.clear()"
37          << "\ncin.fail(): " << cin.fail()
38          << "\ncin.good(): " << cin.good() << endl;
39
40     return 0;
41
42  } // end main
```

Note the use of **clear**.

© 2004 Benjamin X. Wang.
All rights reserved.


```
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(): 1
    cin.bad(): 0
   cin.good(): 0

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

fig7_22.cpp
output (1 of 1)

© 2004 Benjamin X. Wang.
All rights reserved.

7.9 Tying an Output Stream to an Input Stream

- Problem with buffered output
  - Interactive program (prompt user, he/she responds)
  - Prompt needs to appear before input proceeds
    - Buffered outputs only appear when buffer fills or flushed
- Member function **`tie`**
  - Synchronizes streams
  - Outputs appear before subsequent inputs
  - Automatically done for **`cin`** and **`cout`**, but could write
    - **`cin.tie( &cout )`**
  - Need to explicitly tie other I/O pairs
  - To untie
    - **`inputStream.tie( 0 )`**

© 2004 Benjamin X, Wang. All rights reserved.