Complementos de Programação de Computadores – Aula 5b
Input e Output com Streams

Mestrado Integrado em Electrónica Industrial e Computadores

Luís Paulo Reis

lpreis@dsi.uminho.pt

Professor Associado do Departamento de Sistemas de Informação, Escola de Engenharia, Universidade do Minho, Portugal

(Slides Baseados em P.Deitel e H.Deitel 2010)

Outline (I)

Outline (II)

- **21.6 Stream Manipulators**
  - 21.6.1 Integral Stream Base: `dec`, `oct`, `hex` and `setbase`
  - 21.6.2 Floating-Point Precision (`precision`, `setprecision`)
  - 21.6.3 Field Width (`setw`, `width`)
  - 21.6.4 User-Defined Manipulators
- **21.7 Stream Format States**
  - 21.7.1 Format State Flags
  - 21.7.2 Trailing Zeros and Decimal Points (`ios::showpoint`)
  - 21.7.3 Justification (`ios::left`, `ios::right`, `ios::internal`)
  - 21.7.4 Padding (`fill`, `setfill`)
  - 21.7.5 Integral Stream Base (`ios::dec`, `ios::oct`, `ios::hex`, `ios::showbase`)
  - 21.7.6 Floating-Point Numbers; Scientific Notation (`ios::scientific`, `ios::fixed`)
  - 21.7.7 Uppercase/Lowercase Control (`ios::uppercase`)
  - 21.7.8 Setting and Resetting the Format Flags (`flags`, `setiosflags`, `resetiosflags`)
- **21.8 Stream Error States**
- **21.9 Tying an Output Stream to an Input Stream**

Objectives

- To understand how to use C++ object-oriented stream input/output
- To be able to format inputs and outputs
- To understand the stream I/O class hierarchy
- To understand how to input/output objects of user-defined types
- To be able to create user-defined stream manipulators
- To be able to determine the success or failure of input/output operations
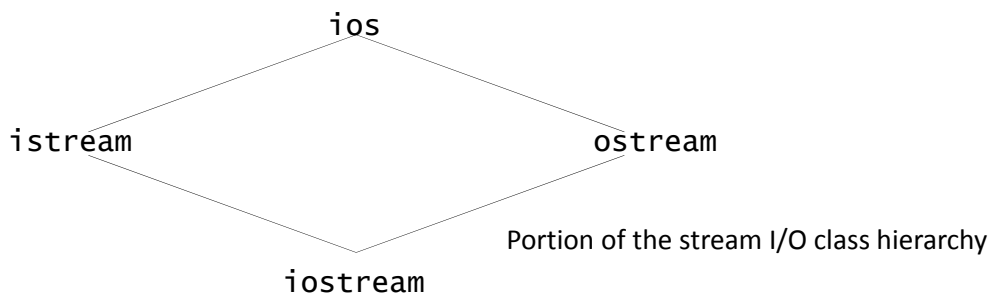- To be able to tie output streams to input streams

21.1 Introduction

- ## Many C++ I/O features are object-oriented
  - Use references, function overloading and operator overloading
- ## C++ uses type safe I/O
  - Each I/O operation is automatically performed in a manner sensitive to the data type
- ## Extensibility
  - Users may specify I/O of user-defined types as well as standard types

---

21.2 Streams

- **Stream:** **A transfer of information in the form of a sequence of bytes**
- **I/O Operations:**
  - Input: A stream that flows from an input device (i.e.: keyboard, disk drive, network connection) to main memory
  - Output: A stream that flows from main memory to an output device (i.e.: screen, printer, disk drive, network connection)
- **I/O operations are a bottleneck**
  - The time for a stream to flow is many times larger than the time it takes the CPU to process the data in the stream
- **Low-level I/O**
  - Unformatted, Individual byte unit of interest
  - High speed, high volume, but inconvenient for people
- **High-level I/O**
  - Formatted, Bytes grouped into meaningful units: integers, characters, etc.
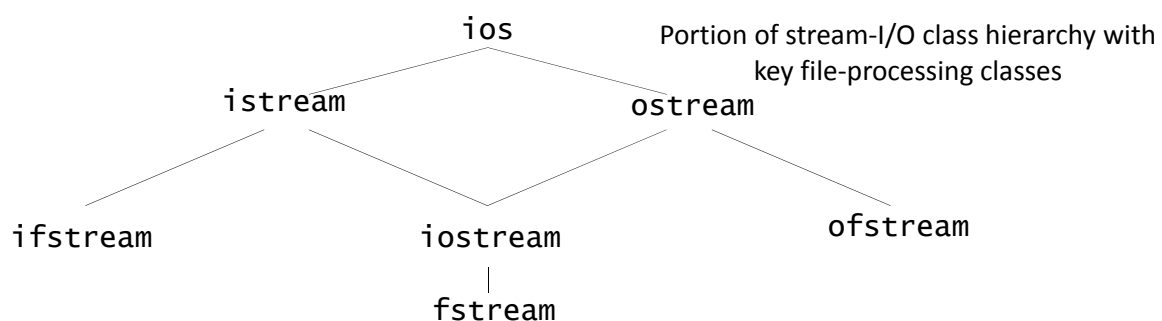 - Good for all I/O except high-volume file processing

21.2 Stream Input/Output Classes and Objects

- **`iostream` library:**
  - `<iostream.h>`: Contains `cin`, `cout`, `cerr` and `clog` objects
  - `<iomanip.h>`: Contains *parameterized stream manipulators*

- **`ios`: `istream` and `ostream` inherit from `ios`**
  - `iostream` inherits from `istream` and `ostream`.

- **<< (left-shift operator)**
  - Overloaded as *stream insertion operator*

- **>> (right-shift operator)**
  - Overloaded as *stream extraction operator*
  - Both operators used with `cin`, `cout`, `cerr`, `clog`, and with user-defined stream objects

```
                        ios

  istream                              ostream


                     iostream
```

Portion of the stream I/O class hierarchy

21.2 Stream Input/Output Classes and Objects

- **`istream`: input streams**
  - `cin >> grade;`
    - `cin` knows what type of data is to be assigned to `grade` (based on the type of `grade`)

- **`ostream`: output streams**
  - `cout << grade;`   − `cout` knows the type of data to output
  - `cerr << errorMessage;`   − Unbuffered - prints `errorMessage` immediately
  - `clog << errorMessage;`   − Buffered - prints `errorMessage` as soon as output buffer is full or flushed

```
                        ios              Portion of stream-I/O class hierarchy with
          istream                                key file-processing classes
                              ostream

  ifstream           iostream                         ofstream

                     fstream
```

21.3 Stream Output

- **`ostream`: performs formatted and unformatted output**
  - Uses `put` for characters and `write` for unformatted output
  - Output of integers in decimal, octal and hexadecimal
  - Varying precision for floating points
  - Formatted text outputs

Stream-Insertion Operator

- **`<<` is overloaded to output built-in types**
  - Can also be used to output user-defined types
  - `cout << '\n';`
    - Prints newline character
  - `cout << endl;`
    - `endl` is a stream manipulator that issues a newline character and flushes the output buffer
  - `cout << flush;`
    - `flush` flushes the output buffer

```cpp
1   // Fig. 21.3: fig21_03.cpp
2   // Outputting a string using stream insertion.
3   #include <iostream>
4
5   using std::cout;
6
7   int main()
8   {
9      cout << "Welcome to C++!\n";
10
11     return 0;
12  } // end function main
```

fig21_03.cpp

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

Program Output

```cpp
1  // Fig. 21.5: fig21_05.cpp
2  // Using the endl stream manipulator.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10     cout << "Welcome to ";
11     cout << "C++!";
12     cout << endl;    // end line stream manipulator
13
14     return 0;
15 } // end function main
```

fig21_05.cpp

Program Output

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

```cpp
1  // Fig. 21.6: fig21_06.cpp
2  // Outputting expression values.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10     cout << "47 plus 53 is ";
11
12     // parentheses not needed; used for clarity
13     cout << ( 47 + 53 );       // expression
14     cout << endl;
15
16     return 0;
17 } // end function main
```

fig21_06.cpp

Program Output

```
47 plus 53 is 100
```

21.3.2 Cascading Stream-Insertion/Extraction Operators

- **<< : Associates from left to right, and returns a reference to its left-operand object (i.e. cout).**
  - This enables cascading:
    ```
    cout << "How" << " are" << " you?";
    ```

  Make sure to use parenthesis:
  ```
  cout << "1 + 2 = " << (1 + 2);
    NOT
  cout << "1 + 2 = " << 1 + 2;
  ```

```cpp
1  // Fig. 21.7: fig21_07.cpp
2  // Cascading the overloaded << operator.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    cout << "47 plus 53 is " << ( 47 + 53 ) << endl;
11
12    return 0;
13 } // end function main
```

fig21_07.cpp

Program Output

```
47 plus 53 is 100
```

21.3.3 Output of `char` * Variables

- << will output a variable of type `char` * as a string

- To output the address of the first character of that string, cast the variable as type `void` *

---

fig21_08.cpp

```cpp
1  // Fig. 21.8: fig21_08.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    const char *string = "test";
11
12    cout << "Value of string is: " << string
13        << "\nValue of static_cast< void * >( string ) is: "
14        << static_cast< void * >( string ) << endl;
15    return 0;
16  } // end function main
```

Program Output

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

21.3.4 Character Output with Member Function `put`; Cascading `put`s

- **`put` member function**
  - Outputs one character to specified stream
    ```
    cout.put( 'A');
    ```
  - Returns a reference to the object that called it, so may be cascaded
    ```
    cout.put( 'A' ).put( '\n' );
    ```
  - May be called with an ASCII-valued expression
    ```
    cout.put( 65 );
    ```
    - Outputs A

21.4 Stream Input

- **>> (stream-extraction)**
  - Used to perform stream input
  - Normally ignores whitespaces (spaces, tabs, newlines)
  - Returns zero (`false`) when EOF is encountered, otherwise returns reference to the object from which it was invoked (i.e. `cin`)

- **>> controls the state bits of the stream**
  - `failbit` set if wrong type of data input
  - `badbit` set if the operation fails

21.4.1 Stream-Extraction Operator

- **>> and << have relatively high precedence**
  - Conditional and arithmetic expressions must be contained in parentheses

- **Popular way to perform loops**

  ```
  while (cin >> grade)
  ```

  - Extraction returns 0 (`false`) when EOF encountered, and loop ends

```cpp
1   // Fig. 21.9: fig21_09.cpp
2   // Calculating the sum of two integers input from the keyboard
3   // with cin and the stream-extraction operator.
4   #include <iostream>
5
6   using std::cout;
7   using std::cin;
8   using std::endl;
9
10  int main()
11  {
12     int x, y;
13
14     cout << "Enter two integers: ";
15     cin >> x >> y;
16     cout << "Sum of " << x << " and " << y << " is: "
17        << ( x + y ) << endl;
18
19     return 0;
20  } // end function main
```

fig21_09.cpp

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

Program Output

fig21_10.cpp

```cpp
1  // Fig. 21.10: fig21_10.cpp
2  // Avoiding a precedence problem between the stream-insertion
3  // operator and the conditional operator.
4  // Need parentheses around the conditional expression.
5  #include <iostream>
6
7  using std::cout;
8  using std::cin;
9  using std::endl;
10
11 int main()
12 {
13    int x, y;
14
15    cout << "Enter two integers: ";
16    cin >> x >> y;
17    cout << x << ( x == y ? " is" : " is not" )
18         << " equal to " << y << endl;
19
20    return 0;
21 } // end function main
```

Program Output

```
Enter two integers: 7 5
7 is not equal to 5
Enter two integers: 8 8
8 is equal to 8
```

fig21_11.cpp

```cpp
1  // Fig. 21.11: fig21_11.cpp
2  // Stream-extraction operator returning false on end-of-file.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    int grade, highestGrade = -1;
12
13    cout << "Enter grade (enter end-of-file to end): ";
14    while ( cin >> grade ) {
15       if ( grade > highestGrade )
16          highestGrade = grade;
17
18       cout << "Enter grade (enter end-of-file to end): ";
19    } // end while
20
21    cout << "\n\nHighest grade is: " << highestGrade << endl;
22    return 0;
23 } // end function main
```

Program Output

```
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
```

21.4.2 `get` and `getline` Member Functions

- `cin.eof()`: returns `true` if end-of-file has occurred on `cin`

- `cin.get()`: inputs a character from stream (even white spaces) and returns it

- `cin.get( c )`: inputs a character from stream and stores it in `c`

fig21_12.cpp

```
1  // Fig. 21.12: fig21_12.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    char c;
12
13    cout << "Before input, cin.eof() is " << cin.eof()
14         << "\nEnter a sentence followed by end-of-file:\n";
15
16    while ( ( c = cin.get() ) != EOF )
17       cout.put( c );
18
19    cout << "\nEOF in this system is: " << c;
20    cout << "\nAfter input, cin.eof() is " << cin.eof() << endl;
21    return 0;
22 } // end function main
```

Program Output

```
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
```

21.4.2 `get` and `getline` Member Functions

- ## `cin.get(array, size):`
  - Accepts 3 arguments: array of characters, the size limit, and a delimiter (default of '\n').
  - Uses the array as a buffer
  - When the delimiter is encountered, it remains in the input stream
  - Null character is inserted in the array
  - Unless delimiter flushed from stream, it will stay there

- ## `cin.getline(array, size)`
  - Operates like `cin.get(buffer, size)` but it discards the delimiter from the stream and does not store it in array
  - Null character inserted into array

```
1   // Fig. 21.13: fig21_13.cpp
2   // Contrasting input of a string with 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     const int SIZE = 80;
12     char buffer1[ SIZE ], buffer2[ SIZE ];
13
14     cout << "Enter a sentence:\n";
15     cin >> buffer1;
16     cout << "\nThe string read with cin was:\n"
17          << buffer1 << "\n\n";
18
19     cin.get( buffer2, SIZE );
20     cout << "The string read with cin.get was:\n"
21          << buffer2 << endl;
22
23     return 0;
24  } // end function main
```

fig21_13.cpp

Program Output

```
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
```

```
1   // Fig. 21.14: fig21_14.cpp
2   // Character input with 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 SIZE = 80;
12      char buffer[ SIZE ];
13
14      cout << "Enter a sentence:\n";
15      cin.getline( buffer, SIZE );
16
17      cout << "\nThe sentence entered is:\n" << buffer << endl;
18      return 0;
19  } // end function main
```

fig21_14.cpp

Program Output

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

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

21.4.3 istream Member Functions peek, putback and ignore

- **ignore member function**
  - Skips over a designated number of characters (default of one)
  - Terminates upon encountering a designated delimiter (default is EOF, skips to the end of the file)

- **putback member function**
  - Places the previous character obtained by get back in to the stream

- **peek**
  - Returns the next character from the stream without removing it

21.4.4 Type-Safe I/O

- **<< and >> operators**
  - Overloaded to accept data of different types
  - When unexpected data encountered, error flags set
  - Program stays in control

21.5 Unformatted I/O with `read`, `gcount` and `write`

- **`read` and `write` member functions**
  - Unformatted I/O
  - Input/output raw bytes to or from a character array in memory
  - Since the data is unformatted, the functions will not terminate at a `newline` character for example
    - Instead, like `getline`, they continue to process a designated number of characters
  - If fewer than the designated number of characters are read, then the failbit is set

- **`gcount:`**
  - Returns the total number of characters read in the last input operation

```
1   // Fig. 21.15: fig21_15.cpp
2   // Unformatted I/O with 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 ];
13
14      cout << "Enter a sentence:\n";
15      cin.read( buffer, 20 );
16      cout << "\nThe sentence entered was:\n";
17      cout.write( buffer, cin.gcount() );
18      cout << endl;
19      return 0;
20  } // end function main
```

fig21_15.cpp

Program Output

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

21.6 Stream Manipulators

Escola de Engenharia
Universidade do Minho

- **Stream manipulator capabilities**
  - Setting field widths
  - Setting precisions
  - Setting and unsetting format flags
  - Setting the fill character in fields
  - Flushing streams
  - Inserting a newline in the output stream and flushing the stream
  - Inserting a null character in the output stream and skipping whitespace in the input stream

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

- ## oct, hex or dec:
  - Change base of which integers are interpreted from the stream.

    Example:

    ```cpp
    int n = 15;
     cout << hex << n;
    ```
  - Prints "F"

- ## setbase:
  - Changes base of integer output
  - Load `<iomanip>`
  - Accepts an integer argument (10, 8, or 16)

    ```cpp
    cout << setbase(16) << n;
    ```
  - Parameterized stream manipulator - takes an argument

```cpp
1  // Fig. 21.16: fig21_16.cpp
2  // Using hex, oct, dec and setbase stream manipulators.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::hex;
12 using std::dec;
13 using std::oct;
14 using std::setbase;
15
16 int main()
17 {
18    int n;
19
20    cout << "Enter a decimal number: ";
21    cin >> n;
22
```

fig21_16.cpp
(Part 1 of 2)

```cpp
23    cout << n << " in hexadecimal is: "
24        << hex << n << '\n'
25        << dec << n << " in octal is: "
26        << oct << n << '\n'
27        << setbase( 10 ) << n << " in decimal is: "
28        << n << endl;
29
30    return 0;
31 } // end function main
```

fig21_16.cpp
(Part 2 of 2)

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

Program Output

21.6.2   Floating-Point Precision (`precision`, `setprecision`)

- ## precision
  - Member function
  - Sets number of digits to the right of decimal point
    `cout.precision(2);`
  - `cout.precision()` returns current precision setting

- ## setprecision
  - Parameterized stream manipulator
  - Like all parameterized stream manipulators, `<iomanip>` required
  - Specify precision:
    `cout << setprecision(2) << x;`

- **For both methods, changes last until a different value is set**

```cpp
1   // Fig. 21.17: fig21_17.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
9   #include <iomanip>
10
11  using std::ios;
12  using std::setiosflags;
13  using std::setprecision;
14
15  #include <cmath>
16
17  int main()
18  {
19     double root2 = sqrt( 2.0 );
20     int places;
21
22     cout << setiosflags( ios::fixed )
23          << "Square root of 2 with precisions 0-9.\n"
24          << "Precision set by the "
25          << "precision member function:" << endl;
26
```

fig21_17.cpp
(Part 1 of 2)

```cpp
27     for ( places = 0; places <= 9; places++ ) {
28        cout.precision( places );
29        cout << root2 << '\n';
30     } // end for
31
32     cout << "\nPrecision set by the "
33          << "setprecision manipulator:\n";
34
35     for ( places = 0; places <= 9; places++ )
36        cout << setprecision( places ) << root2 << '\n';
37
38     return 0;
39  } // end function main
```

fig21_17.cpp (Part 2
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
```

21.6.3   Field Width(`setw`, `width`)

- **`ios width` member function**
  - Sets field width (number of character positions a value should be output or number of characters that should be input)
  - Returns previous width
  - If values processed are smaller than width, fill characters inserted as padding
  - Values are not truncated - full number printed
  - `cin.width(5);`

- **`setw` stream manipulator**
  `cin >> setw(5) >> string;`

- **Remember to reserve one space for the null character**

```
1   // fig21_18.cpp
2   // Demonstrating the width member function
3   #include <iostream>
4
5   using std::cout;
6   using std::cin;
7   using std::endl;
8
9   int main()
10  {
11     int w = 4;
12     char string[ 10 ];
13
14     cout << "Enter a sentence:\n";
15     cin.width( 5 );
16
17     while ( cin >> string ) {
18        cout.width( w++ );
19        cout << string << endl;
20        cin.width( 5 );
21     } // end while
22
23     return 0;
24  } // end function main
```

fig21_18.cpp

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

Program Output

21.6.4 User-Defined Manipulators

- ## We can create our own stream manipulators
  - `bell`
  - `ret` (carriage return)
  - `tab`
  - `endLine`

- ## Parameterized stream manipulators
  - Consult installation manuals

```cpp
1  // Fig. 21.19: fig21_19.cpp
2  // Creating and testing user-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 ) { return output << '\a'; }
12
13 // ret manipulator (using escape sequence \r)
14 ostream& ret( ostream& output ) { return output << '\r'; }
15
16 // tab manipulator (using escape sequence \t)
17 ostream& tab( ostream& output ) { return output << '\t'; }
18
19 // endLine manipulator (using escape sequence \n
20 // and the flush member function)
21 ostream& endLine( ostream& output )
22 {
23    return output << '\n' << flush;
24 } // end function endLine
25
```

**fig21_19.cpp
(Part 1 of 2)**

```
26  int main()
27  {
28     cout << "Testing the tab manipulator:" << endLine
29         << 'a' << tab << 'b' << tab << 'c' << endLine
30         << "Testing the ret and bell manipulators:"
31         << endLine << "..........";
32     cout << bell;
33     cout << ret << "-----" << endLine;
34     return 0;
35  } // end function main
```

Program Output

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

21.7 Stream Format States

- ## Format flags
  - Specify formatting to be performed during stream I/O operations
- ## `setf`, `unsetf` and `flags`
  - Member functions that control the flag settings
- ## Format State Flags
  - Defined as an enumeration in class `ios`
  - Can be controlled by member functions
  - `flags` - specifies a value representing the settings of all the flags
    - Returns `long` value containing prior options
  - `setf` - one argument, "ors" flags with existing flags
  - `unsetf` - unsets flags
  - `setiosflags` - parameterized stream manipulator used to set flags
  - `resetiosflags` - parameterized stream manipulator, has same functions as `unsetf`

- ## Flags can be combined using bitwise OR (|)

21.7.1 Format State Flags

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 is 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 letters (i.e., X and A through F) should be used in the 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. 21.20 Format state flags.	

21.7.2 Trailing Zeros and Decimal Points (ios::showpoint)

- ios::showpoint
  - Forces a float with an integer value to be printed with its decimal point and trailing zeros

    ```
    cout.setf(ios::showpoint)
    cout << 79;
    ```
    79 will print as 79.00000
    - Number of zeros determined by precision settings

```
1   // Fig. 21.21: fig21_21.cpp
2   // Controlling the printing of trailing zeros and decimal
3   // points for floating-point values.
4   #include <iostream>
5
6   using std::cout;
7   using std::endl;
8
9   #include <iomanip>
10
11  using std::ios;
12
13  #include <cmath>
14
15  int main()
16  {
17     cout << "Before setting the ios::showpoint flag\n"
18          << "9.9900 prints as: " << 9.9900
19          << "\n9.9000 prints as: " << 9.9000
20          << "\n9.0000 prints as: " << 9.0000
21          << "\n\nAfter setting the ios::showpoint flag\n";
22     cout.setf( ios::showpoint );
23     cout << "9.9900 prints as: " << 9.9900
24          << "\n9.9000 prints as: " << 9.9000
25          << "\n9.0000 prints as: " << 9.0000 << endl;
26     return 0;
27  } // end function main
```

fig21_21.cpp

```
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
```

Program Output

21.7.3 Justification (`ios::left`, `ios::right`, `ios::internal`)

- **`ios::left`** – Fields to left-justified with padding characters to the right

- **`ios::right`** - Default setting - Fields right-justified with padding characters to the left

- Character used for padding (Default character is space) set by
  - `fill` member function
  - `setfill` parameterized stream manipulator

- **`internal` flag**
  - Number's sign left-justified, Number's magnitude right-justified
  - Intervening spaces padded with the fill character

- **`static` data member `ios::adjustfield`**
  - Contains `left`, `right` and `internal` flags
  - `ios::adjustfield` must be the second argument to `setf` when setting the `left`, `right` or `internal` justification flags
    ```
    cout.setf( ios::left, ios::adjustfield);
    ```

```cpp
1   // Fig. 21.22: fig21_22.cpp
2   // Left-justification and right-justification.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   #include <iomanip>
9
10  using std::ios;
11  using std::setw;
12  using std::setiosflags;
13  using std::resetiosflags;
14
15  int main()
16  {
17     int x = 12345;
18
19     cout << "Default is right justified:\n"
20          << setw(10) << x << "\n\nUSING MEMBER FUNCTIONS"
21          << "\nUse setf to set ios::left:\n" << setw(10);
22
```

fig21_22.cpp
(Part 1 of 2)

```
23     cout.setf( ios::left, ios::adjustfield );
24     cout << x << "\nUse unsetf to restore default:\n";
25     cout.unsetf( ios::left );
26     cout << setw( 10 ) << x
27          << "\n\nUSING PARAMETERIZED STREAM MANIPULATORS"
28          << "\nUse setiosflags to set ios::left:\n"
29          << setw( 10 ) << setiosflags( ios::left ) << x
30          << "\nUse resetiosflags to restore default:\n"
31          << setw( 10 ) << resetiosflags( ios::left )
32          << x << endl;
33     return 0;
34 } // end function main
```

Program Output

```
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
```

```
1  // Fig. 21.23: fig21_23.cpp
2  // Printing an integer with internal spacing and
3  // forcing the plus sign.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::ios;
12 using std::setiosflags;
13 using std::setw;
14
15 int main()
16 {
17    cout << setiosflags( ios::internal | ios::showpos )
18         << setw( 10 ) << 123 << endl;
19    return 0;
20 } // end function main
```

fig21_23.cpp

Program Output

```
+       123
```

21.7.4 Padding (`fill`, `setfill`)

- ## `fill` member function
  - Specifies the fill character
  - Space is default
  - Returns the prior padding character

    ```
    cout.fill('*');
    ```

- ## `setfill` manipulator
  - Also sets fill character

    ```
    cout << setfill ('*');
    ```

```cpp
1   // Fig. 21.24: fig21_24.cpp
2   // Using the fill member function and the setfill
3   // manipulator to change the padding character for
4   // fields larger than the values being printed.
5   #include <iostream>
6
7   using std::cout;
8   using std::endl;
9
10  #include <iomanip>
11
12  using std::ios;
13  using std::setw;
14  using std::hex;
15  using std::dec;
16  using std::setfill;
17
18  int main()
19  {
20     int x = 10000;
21
```

**fig21_24.cpp
(Part 1 of 2)**

```
22    cout << x << " printed as int right and left justified\n"
23         << "and as hex with internal justification.\n"
24         << "Using the default pad character (space):\n";
25    cout.setf( ios::showbase );
26    cout << setw( 10 ) << x << '\n';
27    cout.setf( ios::left, ios::adjustfield );
28    cout << setw( 10 ) << x << '\n';
29    cout.setf( ios::internal, ios::adjustfield );
30    cout << setw( 10 ) << hex << x;
31
32    cout << "\n\nUsing various padding characters:\n";
33    cout.setf( ios::right, ios::adjustfield );
34    cout.fill( '*' );
35    cout << setw( 10 ) << dec << x << '\n';
36    cout.setf( ios::left, ios::adjustfield );
37    cout << setw( 10 ) << setfill( '%' ) << x << '\n';
38    cout.setf( ios::internal, ios::adjustfield );
39    cout << setw( 10 ) << setfill( '^' ) << hex << x << endl;
40    return 0;
41 } // end function main
```

fig21_24.cpp
(Part 1 of 2)

```
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
```

Program Output

21.7.5- Integral Stream Base (`ios::dec,` `ios::oct,` `ios::hex,` `ios::showbase`)

- ## `ios::basefield` static member
  - Used similarly to `ios::adjustfield` with `setf`
  - Includes the `ios::oct,` `ios::hex` and `ios::dec` flag bits
  - Specify that integers are to be treated as octal, hexadecimal and decimal values
  - Default is decimal
  - Default for stream extractions depends on form inputted
    - Integers starting with `0` are treated as octal
    - Integers starting with `0x` or `0X` are treated as hexadecimal
  - Once a base specified, settings stay until changed

```
1   // Fig. 21.25: fig21_25.cpp
2   // Using the ios::showbase flag
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   #include <iomanip>
9
10  using std::ios;
11  using std::setiosflags;
12  using std::oct;
13  using std::hex;
14
15  int main()
16  {
17     int x = 100;
18
19     cout << setiosflags( ios::showbase )
20          << "Printing integers preceded by their base:\n"
21          << x << '\n'
22          << oct << x << '\n'
23          << hex << x << endl;
24     return 0;
25  } // end function main
```

fig21_25.cpp

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

21.7.6 Floating-Point Numbers; Scientific Notation (`ios::scientific`, `ios::fixed`)

- `ios::scientific`
  - Forces output of a floating point number in scientific notation:
    - 1.946000e+009

- `ios::fixed`
  - Forces floating point numbers to display a specific number of digits to the right of the decimal (specified with `precision`)

- `static` data member `ios::floatfield`
  - Contains `ios::scientific` and `ios::fixed`
  - Used similarly to `ios::adjustfield` and `ios::basefield` in `setf`
    - `cout.setf(ios::scientific, ios::floatfield);`
  - `cout.setf(0, ios::floatfield)` restores default format for outputting floating-point numbers

```
1  // Fig. 21.26: fig21_26.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::ios;
9
10 int main()
11 {
12    double x = .001234567, y = 1.946e9;
13
14    cout << "Displayed in default format:\n"
15        << x << '\t' << y << '\n';
16    cout.setf( ios::scientific, ios::floatfield );
17    cout << "Displayed in scientific format:\n"
18        << x << '\t' << y << '\n';
19    cout.unsetf( ios::scientific );
20    cout << "Displayed in default format after unsetf:\n"
21        << x << '\t' << y << '\n';
22    cout.setf( ios::fixed, ios::floatfield )
23    cout << "Displayed in fixed format:\n"
24        << x << '\t' << y << endl;
25    return 0;
26 } // end function main
```

fig21_26.cpp

Program Output

```
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
```

21.7.7 Uppercase/Lowercase Control (`ios::uppercase`)

- `ios::uppercase`
  - Forces uppercase `E` to be output with scientific notation
    `4.32E+010`
  - Forces uppercase `X` to be output with hexadecimal numbers, and causes all letters to be uppercase
    `75BDE`

```
1   // Fig. 21.27: fig21_27.cpp
2   // Using the ios::uppercase flag
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   #include <iomanip>
9
10  using std::setiosflags;
11  using std::ios;
12  using std::hex;
13
14  int main()
15  {
16     cout << setiosflags( ios::uppercase )
17          << "Printing uppercase letters in scientific\n"
18          << "notation exponents and hexadecimal values:\n"
19          << 4.345e10 << '\n' << hex << 123456789 << endl;
20     return 0;
21  } // end function main
```

fig21_27.cpp

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

Program Output

21.7.8 Setting and Resetting the Format Flags (`flags`, `setiosflags`, `resetiosflags`)

- **`flags` member function**
  - Without argument, returns the current settings of the format flags (as a `long`)
  - With a `long` argument, sets the format flags as specified:
    - Returns prior settings

- **`setf` member function**
  - Sets the format flags provided in its argument
  - Returns the previous flag settings as a `long` value
  - Unset the format using `unsetf` member function

  `long previousFlagSettings = cout.setf( ios::showpoint | ios::showpos );`

- **`setf` with two `long` arguments**

  `cout.setf( ios::left, ios::adjustfield );`

  clears the bits of `ios::adjustfield` then sets `ios::left`
  - This version of setf can be used with
  - `ios::basefield (ios::dec, ios::oct, ios::hex)`
  - `ios::floatfield (ios::scientific, ios::fixed)`
  - `ios::adjustfield (ios::left, ios::right, ios::internal )`

- **`Unsetf` –** **Resets specified flags, Returns previous settings**

```cpp
1   // Fig. 21.28: fig21_28.cpp
2   // Demonstrating the flags member function.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7   using std::ios;
8
9
10  int main()
11  {
12     int i = 1000;
13     double d = 0.0947628;
14
15     cout << "The value of the flags variable is: "
16          << cout.flags()
17          << "\nPrint int and double in original format:\n"
18          << i << '\t' << d << "\n\n";
19     long originalFormat =
20          cout.flags( ios::oct | ios::scientific );
21     cout << "The value of the flags variable is: "
22          << cout.flags()
23          << "\nPrint int and double in a new format\n"
24          << "specified using the flags member function:\n"
25          << i << '\t' << d << "\n\n";
```

fig21_28.cpp
(Part 1 of 2)

```cpp
26     cout.flags( originalFormat );
27     cout << "The value of the flags variable is: "
28          << cout.flags()
29          << "\nPrint values in original format again:\n"
30          << i << '\t' << d << endl;
31     return 0;
32  } // end function main
```

fig21_28.cpp
(Part 2 of 2)

Program Output

```
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: 12000
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: 513
Print values in original format again:
1000     0.0947628
```

21.8 Stream Error States

- ## eofbit - **Set for an input stream after end-of-file encountered**
  - `cin.eof()` returns `true` if end-of-file has been encountered on cin
- ## failbit - **Set for a stream when a format error occurs**
  - `cin.fail()` - returns true if a stream operation has failed
  - Normally possible to recover from these errors
- ## badbit - **Set when an error occurs that results in data loss**
  - `cin.bad()` returns `true` if stream operation failed
  - normally nonrecoverable
- ## Goodbit - **Set for a stream if neither `eofbit`, `failbit` or `badbit` are set**
  - `cin.good()` returns `true` if the `bad`, `fail` and `eof` functions would all return false.
  - I/O operations should only be performed on "good" streams

---

21.8 Stream Error States

- ## rdstate – **Returns the state of the stream**
  - Stream can be tested with a `switch` statement that examines all of the state bits
  - Easier to use `eof`, `bad`, `fail`, and `good` to determine state
- ## clear - **Used to restore a stream's state to "good"**
  - `cin.clear()` clears `cin` and sets `goodbit` for the stream
  - `cin.clear( ios::failbit )` actually sets the `failbit`
    - Might do this when encountering a problem with a user-defined type
- ## Other operators
  - `operator!` - **Returns `true` if `badbit` or `failbit` set**
  - `operator void*` – **Returns `false` if `badbit` or `failbit` set**
  - Useful for file processing

```cpp
1   // Fig. 21.29: fig21_29.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 x;
12      cout << "Before a bad input operation:"
13          << "\ncin.rdstate(): " << cin.rdstate()
14          << "\n     cin.eof(): " << cin.eof()
15          << "\n    cin.fail(): " << cin.fail()
16          << "\n     cin.bad(): " << cin.bad()
17          << "\n    cin.good(): " << cin.good()
18          << "\n\nExpects an integer, but enter a character: ";
19      cin >> x;
20
21      cout << "\nAfter a bad input operation:"
22          << "\ncin.rdstate(): " << cin.rdstate()
23          << "\n     cin.eof(): " << cin.eof()
24          << "\n    cin.fail(): " << cin.fail()
25          << "\n     cin.bad(): " << cin.bad()
26          << "\n    cin.good(): " << cin.good() << "\n\n";
27
```

**fig21_29.cpp
(Part 1 of 2)**

```cpp
28      cin.clear();
29
30      cout << "After cin.clear()"
31          << "\ncin.fail(): " << cin.fail()
32          << "\ncin.good(): " << cin.good() << endl;
33      return 0;
34  } // end function main
```

```
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
```

**fig21_29.cpp
(Part 2 of 2)**

Program Output

- `tie` member function
  - Synchronize operation of an `istream` and an `ostream`
  - Outputs appear before subsequent inputs
  - Automatically done for `cin` and `cout`

- `inputStream.tie( &outputStream );`
  - Ties `inputStream` to `outputStream`
  - `cin.tie( &cout)` done automatically

- `inputStream.tie( 0 );`
  - Unties `inputStream` from an output stream

Complementos de Programação de Computadores – Aula 5b
Input e Output com Streams

Mestrado Integrado em Electrónica Industrial e Computadores
Luís Paulo Reis

lpreis1970@gmail.com / lpreis@dsi.uminho.pt
Professor Associado do Departamento de Sistemas de Informação, Escola de Engenharia, Universidade do Minho, Portugal

(Slides Baseados em P.Deitel e H.Deitel 2010)