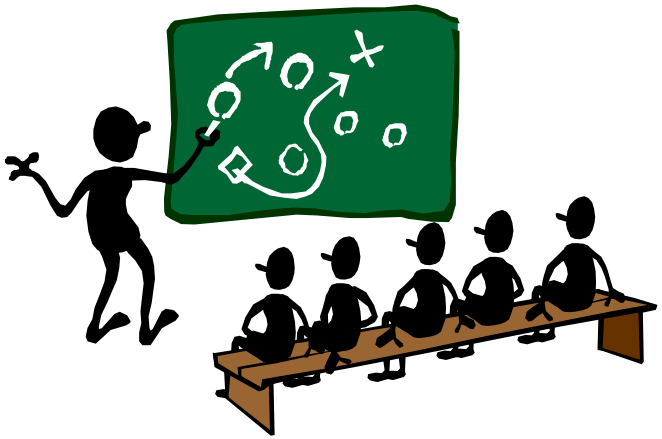
C++ Programming Language
Chapter 12
Streams and File I/O



Juinn-Dar Huang

Associate Professor

jdhuang@mail.nctu.edu.tw

April 2011

Learning Objectives

- I/O stream class hierarchy
- Various I/O formatting options
- I/O manipulators
- File streams
- String streams

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Introduction

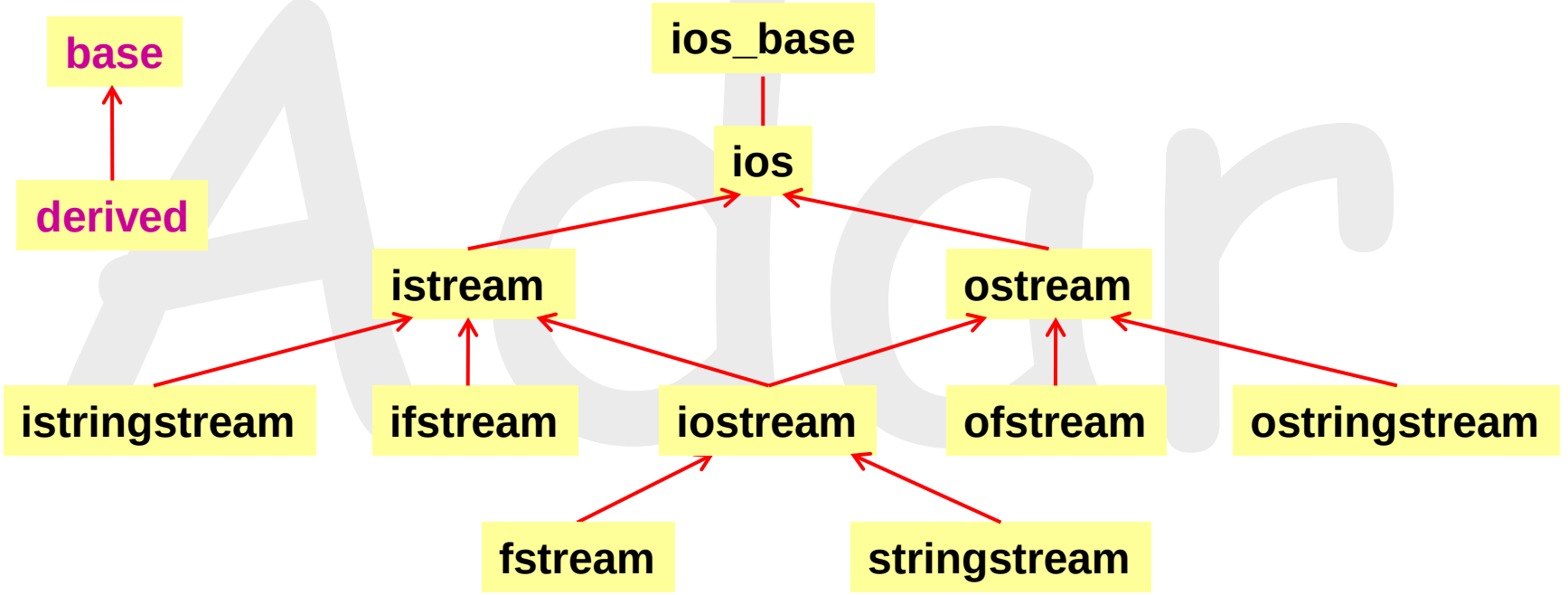
- I/O streams
  - class istream, ostream, (and iostream)
  - special objects for program input and output
  - I/O formatting
- File streams
  - inherited from istream, ostream, (and iostream)
    ➔ ifstream, ofstream, and fstream
- String streams
  - inherited from istream, ostream, (and iostream)
    ➔ istringstream, ostringstream, and stringstream

- Inheritance will be discussed in Chapter 14

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Class Hierarchy

- Details of inheritance will be discussed in Chapter 14
- At this moment, you only need to know



1. Public inheritance ➔ derived IS A base

2. A derived class inherits capability from its base class

P22

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Streams

- A flow of characters
- Input streams
  - flow into program
    - from keyboard/file/string
- Output streams
  - flow out of program
    - go to screen/file/string
- I/O streams
  - both input and output directions

Streams Usage

- We've used streams already
  - cin
    - input stream (istream) object connected to stdin (keyboard by default)
  - cout
    - output stream (ostream) object connected to stdout (screen by default)
  - cerr
    - output stream (ostream) object connected to stderr (screen by default)
- You can define other streams
  - to or from files ; to or from strings (you will see later)
  - used **similarly** as cin and cout

istream and ostream for Built-In Types

- We've used cin/cout and operators >> and << for I/O of built-in types

```
int num = 5;
char ch;
cout << num;
cin >> ch;
```

- We've discussed some other useful member functions of istream and ostream classes for char I/O in Chapter 9
  - cin.get(), cin.unget(), cin.putback(), cin.getline(), …
  - cout.put(), …

I/O for User-Defined Types

- We've already discussed how to overload operator >> & << for I/O of user-defined types
- Examples in Chapter 8

```
ostream& operator<<(ostream& os, const complex& rhs) {
    os << rhs.real() << '+' << rhs.image() << 'i' ;
    return os;
}
```

```
ostream& operator<<(ostream&, const double&)
ostream& operator<<(ostream&, const char&)
```

```
istream& operator>>(istream& is, complex& rhs) {
    is >> rhs.re >> rhs.im ;
    return is;
}
```

```
istream& operator>>(istream&, double&)
```

I/O Formatting

```
#include <iostream>
using namespace std;

int main( ) {
    double d1 = 12.34, d2 = 123456.789;
    int num = 255;
    cout << "1st number is " << d1 << endl;
    cout << "2nd number is " << d2 << endl;
    cout << "3rd number is " << num << endl;
    return 0;
}
```

Output:

```
====================
1st number is 12.34
2nd number is 123457
3rd number is 255
```

What if
I want those decimal points aligned?
I want higher precision?
I want exactly two digits after the decimal point?
I want to output integer values in hex format?

Format Control in ios_base (1/2)

- Class ios_base defines a set of flag bits for I/O formatting
- Besides, it also defines a set of static constants of type fmtflags
  - dec // integer base: base 10 output (decimal)
  - hex // base 16 output (hexadecimal)
  - scientific // floating-point notation: scientific notation
  - fixed // fixed-point notation
  - left // field adjustment: pad after value
  - right // pad before value
  - boolalpha // output true/false instead of 1/0
  - showbase // prefix hex 0x
  - showpoint // print trailing 0s after decimal point
  - showpos // explicit '+' for positive ints
  - … (more)

Format Control in ios_base (2/2)

- Class ios_base also defines a set of member functions to set/clear those flags
  - fmtflags flags() const; // read current flags
  - fmtflags flags(fmtflags f); // set new flag; return previous flags
  - ftmflags setf(fmtflags f) { return flags(flags() | f) ; } // add flag
  - ftmflags setf(fmtflags f, fmtflags mask) // clear and set flags in mask
    { return flags((flags() & ~mask) | (f & mask)); }
  - void unsetf(fmtflags mask) { flags(flags() & ~mask) ; }
    // clear flags in mask

Format Integer Outputs

- The default output format is decimal
- You can change that setting by calling
  - cout.setf(ios_base::dec, ios_base::basefield); // in dec
  - cout.setf(ios_base::hex, ios_base::basefield); // in hex
  - cout.setf(ios_base::oct, ios_base::basefield); // in oct
- Once set, it takes effects until reset

```
cout << 1234 << ' ';
cout.setf(ios_base::hex, ios_base::basefield);
cout << 1234 << ' ';
cout.setf(ios_base::showbase);
cout << 1234 << ' ';
cout.unsetf(ios_base::showbase);
cout << 1234 << ' ';
cout.setf(ios_base::dec, ios_base::basefield);
cout << 1234 << ' ';
```

Output:

==========

1234 4d2 0x4d2 4d2 1234

Format Floating-Point Outputs (1/3)

- Floating-point outputs are controlled by format and precision
- Format
  - general : precision specifies the max # of digits
  - scientific : precision specifies the # of digits after the decimal point
  - fixed: precision specifies the # of digits after the decimal point
- You can change settings by calling
  - cout.setf(ios_base::scientific, ios_base::floatfield); // scientific
  - cout.setf(ios_base::fixed, ios_base::floatfield); // fixed
  - cout.setf(ios_base::fmtflags(0), ios_base::floatfield);
    // reset to general
- Class ios_base defines a set of member functions to show and set precision
  - streamsize precision() const; // get precision
  - streamsize precision(streamsize n); // set precision and return old

Format Floating-Point Outputs (2/3)

- The default format is general
- The default precision is 6
- Floating-point values are rounded, not truncated

```
double num = 1234.56789;
cout << "general:\t" << num << endl;
cout.setf(ios_base::scientific, ios_base::floatfield);
cout << "scientific:\t" << num << endl;
cout.setf(ios_base::fixed, ios_base::floatfield);
cout << "fixed:\t\t" << num << endl;
cout.setf(ios_base::fmtflags(0), ios_base::floatfield);
cout << "general:\t" << num << endl;
```

Output:

```
============================
general:        1234.57
scientific:     1.234568e+003
fixed:          1234.567890
general:        1234.57
```

Format Floating-Point Outputs (3/3)

- If putting cout.precision(8) / cout.precision(4) ahead

```
Output:
==========================
general:        1234.5679
scientific:     1.23456789e+003
fixed:          1234.56789000
general:        1234.5679
```

```
Output:
==========================
general:        1235
scientific:     1.2346e+003
fixed:          1234.5679
general:        1235
```

- If putting cout.precision(3) ahead

```
Output:
==========================
general:        1.23e+003
scientific:     1.235e+003
fixed:          1234.568
general:        1.23e+003
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Field Width (1/2)

- Class ios_base defines functions to show & set field width
  - streamsize width() const; // get field width
  - streamsize width(streamsize w); // set field size
- Class ios defines functions to specify a character if padding is needed
  - char fill() const; // get filler character
  - char fill(char ch); // set filler character
- width() specifies the min # of characters for the **NEXT <<** output operation **ONLY**
- The default field size is 0 (i.e., as many as needed)
- The default filler is the space character

Field Width (2/2)

```
cout.width(4);
cout << 12 << endl;
cout.width(4); cout.fill('#');
cout << "ab" << endl;
cout.width(4);
cout << "abcdef" << endl;
cout.width(4);
cout << 12 << ':' << 34 << endl;
cout.width(0);
cout << "ab" << endl;
```

Output:

```
============================
  12
##ab
abcdef
##12:34
ab
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Field Alignment

- You can change field alignment by calling
  - cout.setf(ios_base::left, ios_base::adjustfield); // left alignment
  - cout.setf(ios_base::right, ios_base::adjustfield); // right alignment
- The default alignment is right

```
cout.width(6); cout.fill('#');
cout << "ab" << endl;
cout.width(6); cout.setf(ios_base::left, ios_base::adjustfield);
cout << "ab" << endl;
```

Output:

```
==========================
####ab
ab####
```

Format Boolean Outputs

```
bool b = false;
cout << b << endl;
cout.setf(ios_base::boolalpha);
cout << b << endl;
cout.unsetf(ios_base::boolalpha);
cout << b << endl;
```

Output:

```
==========================
0
false
0
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Trailing 0s and Positive Signs

```
double num = 12.0;
cout << num << endl;
cout.setf(ios_base::showpoint);
cout << num << endl;
cout.setf(ios_base::showpos);
cout << num << endl;
```

Output:

```
==========================
12
12.0000
+12.0000
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

I/O Manipulators (1/2)

- To make your life easier, C++ provides a set of functions for manipulating aforementioned flags
  - called **manipulators**
- Actually, you've already used one of them

  cout << 1234 << endl;
- There are lots of more…

cout << boolalpha << noboolalpha << showbase << noshowbase
<< showpoint << noshowpoint << showpos << noshowpos
<< left << right << dec << hex << fixed << scientific
<< setfill('#') << setprecision(4) << setw(8) << …

- Include **<iomanip>** for setfill(), setprecision(), setw()
- Include **<iostream>** for others

I/P Manipulator (2/2)

```
#include <iostream>
#include <iomanip>

int main() {

    cout << 1234 << ", " << showbase << hex << 1234 << endl;
    cout << dec << noshowbase;
    cout << '(' <<  left << setw(4) << setfill('@') << 12 << "), (" << 12 << ")\n";

    return 0;
}
```

Output:

```
============================
1234, 0x4d2
(12@@), (12)
```

Class Hierarchy for Streams

Remind you again that

- If class D is derived from class B (public inheritance) ➔
  - an object of class D **IS AN** object of type B
  - derived class
    - can inherit public members from its base class
    - can add its own new features
- Example:
  - class ifstream (input file stream) is derived from class istream
    - ifstream inherits public members of istream/ios/ios_base
    - it then adds its own member functions like open, close, …
  - an ifstream object **IS** also an istream (or ios, ios_base) object
    - public inheritance ➔ is-a relationship

P3

File Copy Using File Streams (1/2)

```
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

void error(const char *p1, const char* p2="") {
    cerr << p1 << p2 << endl;
    exit(1);
}

int main(int argc, char*argv[]) {

    if(argc != 3) error("Wrong number of arguments!");
```

d:\> copy a.txt b.txt

File Copy Using File Streams (2/2)

```
    ifstream ifs(argv[1]); // open a file, named by argv[1], as an input file stream
    if(!ifs) // operator!() inherited from ios, test if file open fails
        error("Cannot open input file: ", argv[1]);

    ofstream ofs(argv[2]); // open a file, named by argv[2], as an output file stream
    if(!ofs) // operator!() inherited from ios, test if file open fails
        error("Cannot open output file: ", argv[2]);

    char ch;
    while(ifs.get(ch))  // get() is inherited from istream
        if (!ofs.put(ch)) break; // put() is inherited from ostream

    if(!ifs.eof() || !ofs)  // eof() is inherited from ios, test if end-of-file is reached
        error("Unexpected failure !");

    return 0;
}
```

File Open and Close

- Two ways for file open
  - Method 1

```
ifstream ifs("pathname\input_file_name");
ofstream ofs("pathname\output_file_name");
```

  - Method 2

```
ifstream ifs; ifs.open("pathname\input_file_name");
ofstream ofs; ofs.open("pathname\output_file_name");
```

- When file streams are no longer used ➔ close them

```
ifs.close();
ofs.close();
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Streams Usage

- Once input/output file streams are successfully opened
  ➔ **You can use them just like input/output streams**

```
int i; double d, char str[100];
ifstream ifs("ifilename");
ifs >> i >> d; ifs.getline(str, 100);        // not from keyboard but input file

ofstream ofs("ofilename");
ofs << setw(10) << setfill('#') << i << endl
<< setprecision(8) << fixed << d; ofs.put('\n');
// not to screen but output file

complex c(1.0, 2.0);
ofs << c << endl;        // to output file
// no need to define ofstream& operator<<(ofstream& ofs, const complex&)
// ostream& operator<<(ostream& ofs, const complex&) works just fine!
```

Simple File Input/Output (1/2)

Display 12.1 Simple File Input/Output

```
1  //Reads three numbers from the file infile.txt, sums the numbers,
2  //and writes the sum to the file outfile.txt.
3  #include <fstream>
4  using std::ifstream;
5  using std::ofstream;
6  using std::endl;

7  int main( )
8  {
9      ifstream inStream;
10     ofstream outStream;

11     inStream.open("infile.txt");
12     outStream.open("outfile.txt");

13     int first, second, third;
14     inStream >> first >> second >> third;
15     outStream << "The sum of the first 3\n"
16               << "numbers in infile.txt\n"
17               << "is " << (first + second + third)
18               << endl;
```

A better version of this program is given in Display 12.3.

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Simple File Input/Output (2/2)

```
19        inStream.close( );
20        outStream.close( );

21        return 0;
22    }
```

SAMPLE DIALOGUE

There is no output to the screen and no input from the keyboard.

infile.txt
(Not changed by program)

```
1
2
3
4
```

outfile.txt
(After program is run)

```
The sum of the first 3
numbers in infile.txt
is 6
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Append to a File

- Previously, we create an output file stream using
  ofstream ofs1("a.txt");
  - if a.txt doesn't exist ➔ it will be created as an empty file
  - if a.txt exists ➔ the original content will be cleared out! (empty file)

- It is possible to append new content to an existing file
  ofstream ofs2("a.txt", ios_base::app);
  - if a.txt doesn't exist ➔ create it first and append to the end
  - if a.txt exists ➔ append to the end

Check File Open Success

- File open could fail
  - if input file doesn't exist
  - no write permissions to output file
  - unexpected results
- Use member functions fail() or operator!() to check whether a stream operation, e.g., file open, succeeds
  - both are inherited from class ios

```
ifstream ifs("in.txt");
if (ifs.fail()) {
  cout << "Input file open failed.\n";
  exit(1); }

ofstream ofs("out.txt");
if (!ofs) {
  cout << "Output file open failed.\n";
  exit(1); }
```

Check End of File

- Use loop to process file until end
  - a typical way to traverse the whole file
- Test for end-of-file
  - member function eof(), inherited from class ios
  - eof() returns true if end-of-file is reached

```
char next;
ifs.get(next);
while (! ifs.eof()) {
    cout << next;
    ifs.get(next);
}
```

String Streams (1/2)

- You can get/put data from/to a file through ifstream/ofstream
- Similarly, you can get/put data from/to a string through istringstream/ostringstream

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

string repeater(const string& str, int n) {
    ostringstream oss;
    for (int i = n; i > 0; --i) oss << str << ' ';
    return oss.str();  // return the resultant string
}

int main( ) {
    cout << repeater("Surprise!", 3) << endl;
    return  0;
}
```

Output:

==========

Surprise! Surprise! Surprise!

String Streams (2/2)

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

void word_per_line(const string& s) {
    istringstream iss(s);
    string w;
    while(iss >> w) cout << w << endl;
}

int main( ) {
    string str("If you think C++ is difficult, try English!");
    word_per_line(str);

    return 0;
}
```

Output:

```
============================
If
you
think
C++
is
difficult,
try
English!
```


Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Summary

- Be familiar with I/O stream class hierarchy (Page 3)
- Be familiar with istream (cin) and ostream (cout) usage
- Various I/O formatting options
  - integer base
  - floating-point notation
  - field width and alignment; filler character
  - …
- I/O manipulators
  - e.g., cout << hex << setw(10) << setprecision(5) << …
- File streams (ifstream, ofstream, fstream)
  - open/close; read/write/append
- String streams (istringstream, ostringstream, stringstream)