Computational Intelligence on Automation Lab @ NCTU

UEE1303(1070) S'12 Object-Oriented Programming in C++

Lecture 06:

Streams and File Input/Output

Learning Objectives

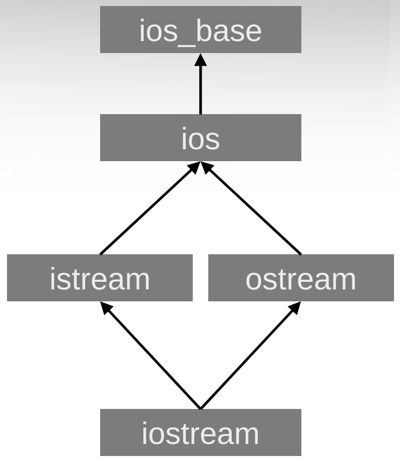
- I/O stream
  - istream and ostream member functions
- File stream
  - ifstream and ofstream member functions
  - access binary files
- String stream
  - istringstream and ostringstream member functions

I/O Stream

- A stream is a sequence of bytes used in an input or output operation
- C++ provides both low-level and high-level input/output (I/O) capabilities
- Low-level I/O is *unformatted*
  - bytes are transferred into and out from memory without regard to the *type of data*
  - for high-volume, high-speed processing
- High-level I/O is formatted
  - bytes are grouped into *meaningful* units such as integers, doubles, and class object types

Stream Class

- The istream class
  - includes a definition of the *extraction* operator >>
- The ostream class
  - includes a definition of the *insertion* operator <<
- The iostream is short for input and output stream



Understand `cin` and `cout`

- When you include `iostream` at the top of your program files, you gain access to these functions `cin` and `cout`
  - `cin` and `cout` are objects and members of the class
  - can use operators such as << and >>
  - *polymorphism*: can generate different machine instructions when placed with different types of variables

Member Functions of `istream` (1/2)

- In C++, the easiest way to read a character is to use `cin` with the extraction operator
  - Extraction operator is an overloaded function named `operator>>()`
  - `cin >> someVariable;`
- Besides the overloaded extraction operator, including `iostream` provides other input functions
  - most compilers support other `istream` member functions, such as `eof()`, `bad()`, and `good()`

Member Functions of `istream` (2/2)

- As an object, `cin` contains member functions
  - use those member functions with the dot operator and the function name

istream function	prototype	purpose

get()	`istream& get(char &);`
	`int get();`
	`istream& get(char *str, int len, char c='\n');`
		extract ***unformatted*** data from a stream

getline()	`istream& getline(char *str, int len, char c = '\n');`	get a line of data from an input stream
ignore()	`istream& ignore(int length = 1, char c = '\n');`	extract and discard characters

Use `get()` Function (1/3)

- The `get()` function takes a character argument and returns a reference to the object that invoked it
  - prototype:

```
istream& get(char &);
```

  - multiple `get()` function calls can be chained

```
char first, middle, last;
cin.get(first);
cin.get(middle).get(last); //why?
```

Use `get()` Function (2/3)

- `cin.get()` can retrieve any character, including letters, numbers, punctuation, and white space such as the character generated by pressing the **`Enter`** key
- Most compilers overload `get()` so that it can also take *no argument*
  - Prototype:

```
int get();
```

  - Example:

```
cout << "Press any key to continue";
c=cin.get(); //c: casted to an integer
```

Use `get()` Function (3/3)

- The `istream` class `get()` function is overloaded so that it can take two or three arguments

```
istream& get(char *str, int len,
                char c = '\n');
```

  - allow you to input a string of characters
  - *1st* argument is a pointer that holds the address of the string (necessary)
  - *2nd* argument is the number of characters that will be stored (necessary)
  - *3rd* argument is the character that terminates the entry, often called the *delimiter* character
- See lec6-1.cpp

Use `getline()` Function

- Prototype:

```
istream& getline(char *str,
                 int len, char c='\n');
```

- The `getline()` function reads a line of text at the address represented by `str`
- It reads until it reaches either the length or the character used as the third argument
- See lec6-1.cpp

Use `ignore()` Function

- using the `ignore()` function to *ignore* or *skip* additional characters left in input stream
  - Prototype

```
istream& ignore(int length = 1,
                     char c = '\n');
```

  - where `length` is the maximum number of characters to ignore, and c is the consumed character that stops the `ignore()` function
  - The *delim* character itself is also discarded
  - See lec6-1.cpp

Member Functions of `ostream` (1/2)

- The concepts you have learned while studying the `cin` object apply to the `cout` object as well
- It is a member of the `ostream` class, which supports member functions and overloaded operators
  - Example: the `operator<<()` function
- Besides member functions, the `cout` object also has data members, or states

Member Functions of `ostream` (2/2)

- Other `ostream` member functions include `put()`, `flush()`, `eof()`, `bad()`, and these functions shown in table

istream function	prototype	purpose
`setf()`	`fmtflags setf(fmtflags);`	tasks arguments that set the bits of `cout`.
`unsetf()`	`fmtflags unsetf(fmtflags);`	tasks arguments that unset the bits of `cout`.
`precision()`	`int precision(int);`	set precision
`width()`	`int width(int);`	set field width

`setf()` and `unsetf()` Functions (1/3)

- Many of the states of `cout` are contained in a long integer field
  - each bit represents some condition of the object
  - example: bit that represents show positive sign might be turned on
- The arguments that determine the state of the `cout` object are called format flags or state flags
  - all begin with `ios::`

`setf()` and `unsetf()` Functions (2/3)

- `ios::left` - left-justifies output within the field size, which may be set by the `width()` function
- `ios::right` - right-justifies output within the field size
- `ios::dec` - formats numbers in decimal (base 10)
- `ios::showpos` - inserts + before positive numbers
- `ios::showpoint` - displays the decimal point and six significant digits for all floating-point numbers

`setf()` and `unsetf()` Functions (3/3)

- `setf()`, a member of `ios` class, takes arguments that set the bits of `cout`
  - `cout.setf(ios::showpos);` turns on the bit that forces the display of + with positive numbers

```
cout.setf(ios::showpos |
          ios::dec | ios::showpoint);
cout << 4.0;
```

```
+4.00000
```

  - `unsetf()` deselects the bit(s)

```
cout.unsetf(ios::showpos);
```

Use `width()` Function (1/2)

- `width()` changes the output field width
  - by default, the argument is right-aligned
  - use `ios::left` to change the default alignment
- The following statements produce three blanks followed by 13, for a total output of five characters:

```
cout.width(5);
cout << 13;
// or cout << setw(5) << 13;
```

```
   13
```

Use `width()` Function (2/2)

- If value provided for `width()` is not large enough for the displayed value, it is disregarded

```
cout.width(2);
cout << 5555;
```

```
5555
```

- Applies only to the *first* subsequent field to be output
  - call it each time you want a width specification to take effect

Use `precision()` Function

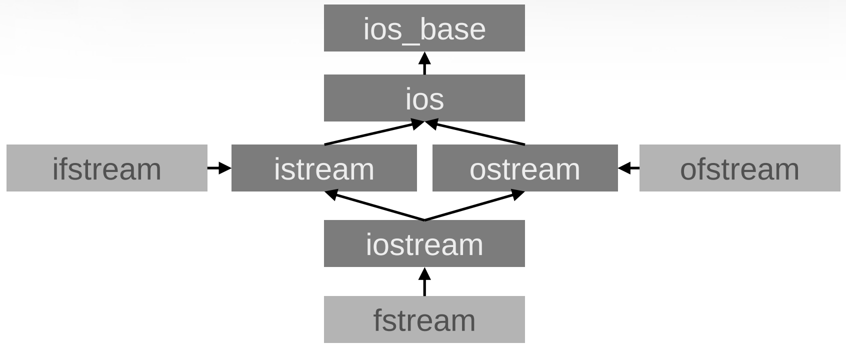
- Use to control the number of significant digits in the output
- For example, the constant 12.98765 contains seven digits; you can display four digits as follows:

```
cout.precision(4);
cout << 12.98765; //displays 12.99
```

- Unlike `width()`, `precision()` applies to *all subsequent output fields* until the precision is reset by another call to it

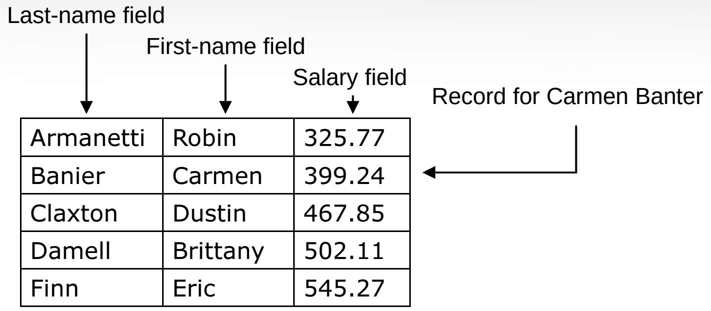
Stream Class

- `fstream` contains `typedefs` that provide aliases for the template specializations in the class
  - `ifstream`, `ofstream`, and `fstream`



Understand Computer Files

- Data file: contains records that are logically related
  - A *delimiter* character separates data fields



Armanetti	Robin	325.77
Banier	Carmen	399.24
Claxton	Dustin	467.85
Damell	Brittany	502.11
Finn	Eric	545.27

- Group of related files is often stored in a database

Simple File Output (1/5)

- To perform file processing, `<iostream>` and `<fstream>` must be included in your program file
- To perform file output, instantiate your own member of the `fstream` or `ofstream` class

```
ofstream outFile("Data.txt");
```

  - if the file does not exist, it is created
  - if the file already exists, it is overwritten

Simple File Output (2/5)

- Use two backslashes in a pathname
  - example:

```
"c:\\DataFolder\\Data.txt"
```

```
"/user/home/9913052/Data.txt"
```

- If you do not provide a filename when you instantiate an `fstream` or `ofstream` object, no file is opened
  - explicitly open the file later with `open()`:

```
ofstream outFile;
outFile.open("Data.txt");
```

Simple File Output (3/5)

- The `fstream` and `ofstream` constructors and the `open()` function are overloaded to accept a second argument to indicate a file mode
- File mode tells C++ how to open the file:
  - `ios::in` open the file for input
  - `ios::out` (default) open the file for output
  - `ios::app` append the file (rather than re-create it)
  - `ios::ate` open an existing file (either input or output) and seeks the end.
  - `ios::binary` open the file for binary I/O

Simple File Output (4/5)

- An `open()` operation (whether explicit or through the constructor) can fail for several reasons
  - attempt to open an existing file using *the wrong path*
  - attempt to create a file on a *storage device that is full*
  - attempt to open a file using an *invalid* mode
- To check whether an `open()` has failed, you can try several methods, using the `out` object itself or its `good()` or `bad()` functions

Simple File Output (5/5)

- Once the file is open, you can write to it as follows:

```
myFile << "This is going to the disk";
```

- After you have written to a file, you do not need to explicitly execute any sort of “close file”
  - when an `fstream` or `ofstream` object goes out of scope, a destructor is called that closes the file
  - To close a file before the file object goes out of scope, use `close()`:

```
myFile.close();
```

An Example for File Output

lec6-3.cpp

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

int main() {
    fstream myFile;
    myFile.open(“test.dat”, ios::in);
    if (myFile.good())
        cout << “File opened!” << endl;
    else
        cout << “Cannot open file!” << endl;
    return 0;
}
```

Simple File Input (1/2)

- To read a file from within a program, create an instantiation of the `ifstream` class
  - Example:

```
ifstream someData("Data.txt");
```

- Name of instantiated object can be any legal identifier
- The file is identified by the filename used as a parameter to the `ifstream` object's constructor

Simple File Input (2/2)

- `open()` and `close()` work with `ifstream` objects
- Appropriate file modes can be used as arguments to `ifstream` constructor or to `open()` statements
- When an `ifstream` object goes out of scope, the file is closed automatically
- You can call `close()` to free resources

An Example for File Input

```
20 50.9 30.7 46.6
89 90.8 24   50   50
```

lec6-3.cpp

```
#include <fstream>
#include <iostream>
using namespace std;
int main() {
    double sum=0, t; int count=0;
    ifstream in(“data.txt”, ios::in);
    if (!in)
        cout << “Cannot open file!” << endl;
    while (in >> t) {
        sum += t;
        count++;
    }
    cout << “avg=” << sum/count << endl;
}
```

Write/Read Objects (1/2)

```
class CStu {
    int sid; string name; double gpa;
friend ostream& operator<<(ostream&, CStu);
friend istream& operator>>(istream&, CStu&);
}

ostream& operator<<(ostream& out, CStu s) {
    out << s.sid << “ ” << s.name << “ ”
        << s.gpa << endl;
    return out;
}
istream& operator>>(istream& in, CStu& s) {
    in >> s.sid >> s.name >> s.gpa;
    return in;
}
```

lec6-4.cpp

Write/Read Objects (2/2)

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

int main() {
    CStu a;
    ofstream out; out.open(“out6-4.txt”);
    cout << “Enter id, name, and gpa\n”;
    while (cin >> a) {
        out << a << endl;
        cout << “Enter id, name, and gpa\n”;
    }
    out.close();
    return 0;
}
```

lec6-4.cpp

Write/Read Binary Files

- Reading or writing a binary file need to use `read()` and `write()` in `<fstream>`
  - read *n* chars from the binary file to a buffer

```
istream& istream::read(char *t, int n);
```

  - read *n* chars from the buffer and write them to the binary file

```
ostream& ostream::write(const char *t,
                        int n);
```

```
write(p1,50); //write 50 chars to disk
read(p2,30);  //read 30 chars
```

Example for Binary I/O (1/2)

lec6-5.cpp

```
#include <fstream>
#include <iostream>
using namespace std;
int main() {
    ofstream out(“dat6-4.txt", ios::out |
                               ios::binary);
    if (!out) {
        cout << “dat6-4.txt is not opened";
        return -1;
    }
    for (int i=1; i<200; i+=2) {
        out.write((char *)&i, sizeof(i));
    } // write 100 odd numbers
    out.close();
```

Example for Binary I/O (2/2)

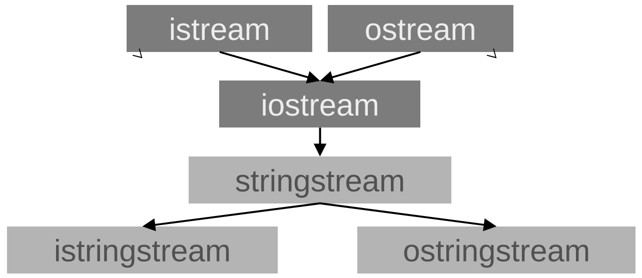
```
    ifstream inp(“dat6-4.txt", ios::in |
                               ios::binary);
    inp.seekg(15*sizeof(int));
    //move to 16th odd integers
    int x;
    for (int i=1; i<50 && !inp.eof(); ++i) {
        inp.read((char *)&x, sizeof(i));
        //read  each odd number
        cout << x << " ";
        if (!(i%=10)) cout << endl;
    }
    cout << endl;
    inp.close();

}
```

lec6-5.cpp

Stream Class

- `stringstream` contains `typedefs` that provide aliases for the template specializations in the class
  - `ifstream`, `ofstream`, and `fstream`
  - Input and output is string object



Examples for `istringstream`

- An example for `istringstream`

lec6-6.cpp

```
string buffer;
getline(cin, buffer);
istringstream inStr(buffer);
long value = 0;
double data = 0.0;
inStr >> value >> data;
```

```
...
istringstream inStr; ...
getline(cin, buffer);
inStr.clear(); inStr.str(buffer);
...
```

Examples for `ostringstream`

- Examples

lec6-6.cpp

```
ostringstream outStr;
double number = 2.5;
outStr << “number = ” << (3 * number/2);
string output = outStr.str();
```

```
stringstream ss;
int data=200;
int result;
ss<<data;     //push data into ss
ss>>result;   //pop result from ss
```

Summary

- `cout` and `cin` are members of a class
  - `>>` is overloaded to input all built-in types
- The ostream class provides useful output functions: `setf()`, `unsetf()`, `width()`, and `precision()`
- To perform file output, you must instantiate your own member of the `fstream` or `ofstream` class
- `stringstream` provides convenient conversion between `stream` and `string`