INTRODUCTION TO PROGRAMING

Chapter 6

File Processing

Khoa Công Nghệ Thông Tin
Trường Đại Học Khoa Học Tự Nhiên
ĐHQG-HCM

GV: Thái Hùng Văn

Outlines

- **Introduction**
- **The Data Hierarchy**
- **File Types**
- **Input /Output Streams**
- **Stream Headers, Templates and Classes**
- **File Streams**
- **File Modes**
- **Writing Data from a Text File**
- **Reading Data from a Text File**
- **Example**
- Issues to expand career knowledge

Introduction

- Storage of data
  - Arrays, strings, structs, and **all variables** in C++ are **temporary** (stored in RAM)
  - **Files** are **permanent** (stored in disk, tapes, cards)
- Size of data
  - The total size of the static variables is limited by the size of the stack (very small, most systems don't auto-grow stacks). On Windows, the typical maximum size for a **stack** is **1MB**.
  - The total size of the dynamic variables is limited by the size of the heap. Heap can grow to **all available** (virtual) **memory**, and not too big.
  - The data stored in the **file** is **unlimited** in **size**.
- Data access speed
  - File access speed on HDD or USB disk is much slower than RAM. Not bad with SSD, and will be equivalent if it is RAM disk

Data hierarchy

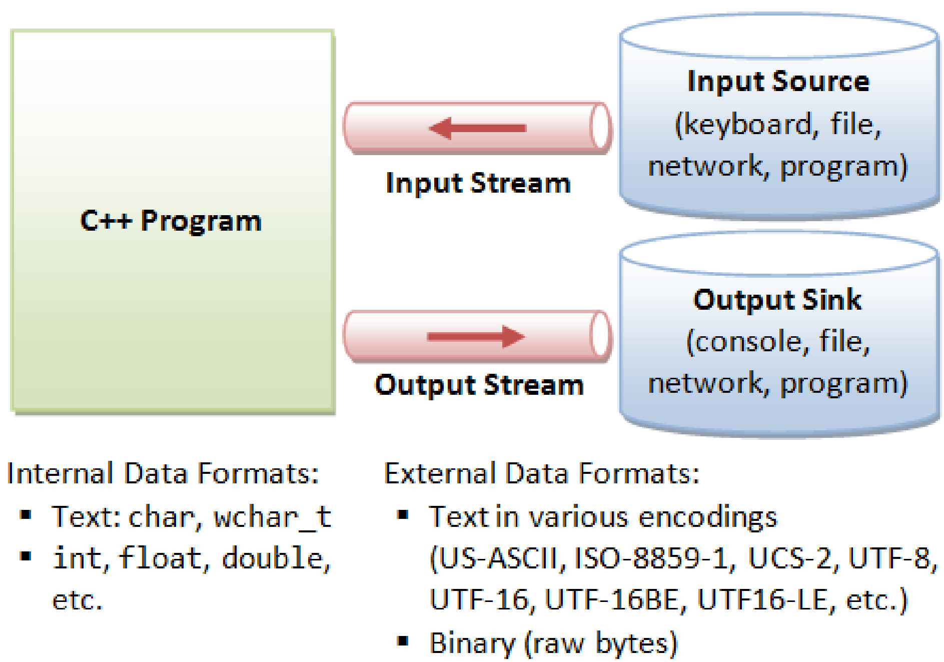
- **Data hierarchy** refers to the systematic organization of data, often in a hierarchical form. The components of the data hierarchy are listed below:
    - A **Data field** holds a single fact or attribute of an entity. Consider a date field, e.g. "15/10/2019". There are a single date field, or 3 sub fields: day of month, month and year.
    - A **Record** is a collection of related fields. An Employee record may contain a name fields address fields, birthdate field, so on.
    - A **File** is a collection of related records. If there are N employees, then each employee would have a record
    - Files are integrated into a **database**. This is done using a *Database Management System*

File Types

- In programming, all files can be categorized into one of two file formats - **binary** or **text**.
- Both binary and text files contain data stored as a series of bytes, and may look the same on the surface, but they encode data differently.
  - The bytes in text files represent characters
  - The bits in binary files represent custom data.
- While text files contain only textual data, binary files may contain both textual and custom binary data.

Input /Output Streams

- C++ IO are based on ***streams***, which are sequence of bytes flowing in and out of the programs
  - In **input** operations, data bytes flow from an ***input source*** *(such as keyboard, file, network,..)* into the program.
  - In **output** operations, data bytes flow from the program to an ***output sink*** (*console, file, network, another program,..*)



Mechanism of performing IO via Stream

- To perform input and output, a C++ program:
    1. Construct a stream object.
    2. Connect (Associate) the stream object to an actual IO device (e.g., keyboard, console, file, ..)
    3. Perform input/output operations on the stream, via the functions defined in the stream's pubic interface in a device independent manner.
    4. Disconnect (Dissociate) the stream to the actual IO device (e.g., close the file).
    5. Free the stream object.

IO stream functions /operations

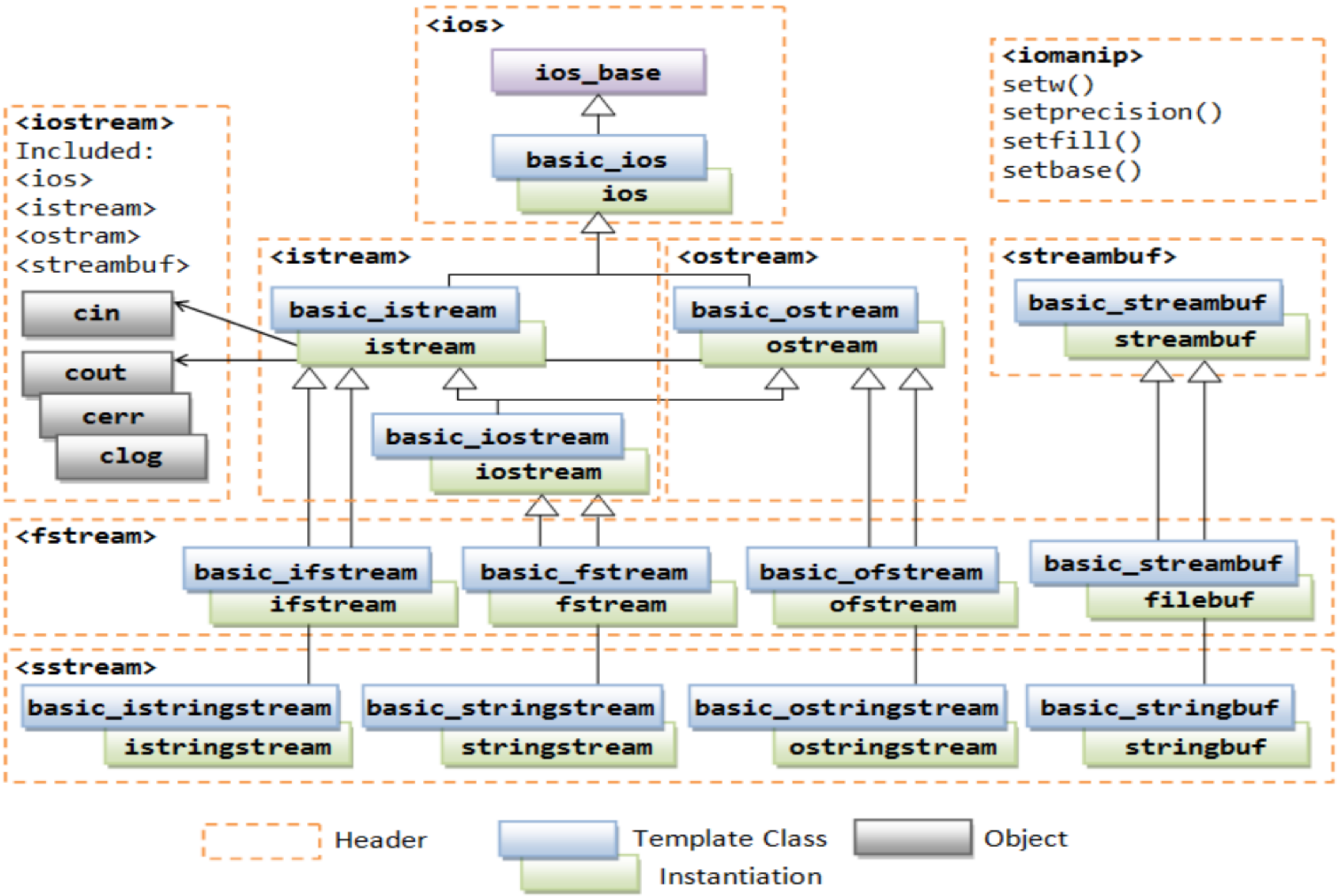
- C++ streams provide both the formatted & unformatted IO functions.
  - In **formatted** or high-level IO, bytes are grouped and converted to types such as **int**, **double**, **string** or **user-defined types**.
  - In **unformatted** or low-level IO, bytes are treated as **raw bytes** and unconverted.
- Formatted IO operations are supported via overloading the stream insertion (<<) and stream extraction (>>) operators, which presents a consistent public IO interface.
- Examples:

```
int a, b;
cin>>a>>b;
cout<<a+b<<endl;
cin.put(a);
```

C++ Stream Headers

- C++ IO stream is provided in some main headers:
  - <**iostream**>: included <ios>, <istream>, <ostream> and <streambuf>; provided basic functions /operations on the standard IO device (keyboard, screen)
  - <**fstream**> : for file IO
  - <**sstream**> : for string IO
  - <**iomanip**> provided manipulators such as setw(), setprecision(), setfill(), setbase(),.. for formatting

Stream Headers, Templates and Classes



https://www.ntu.edu.sg/home/ehchua/programming/cpp/cp10_IO.html

Files Stream

- We have been using the **iostream** standard library, which provides **cin** and **cout** methods for reading /writing from /to standard IO devide respectively.
- For reading /writing from /to Files, we use another standard C++ library called **fstream**, which defines 3 new data types
  - **ifstream** : Stream class represents the input file stream and is used to read information from files.
  - **ofstream** : Stream class represents the output file stream and is used to create files and to write information to files.
  - **fstream** : Stream class has the capabilities of both **ofstream** and **ifstream** ; it can create files, write information to files, and read information from files.
- To perform input /output via these streams, we have to connect them with physical files by **open** method

Openning a File

- A file must be opened before you can read /write from /to it
  - **ifstream** object is used to open a file for reading purpose only.
  - Either **ofstream** or **fstream** may be used to open a file for writing
- Following is the standard syntax for **open**() function, which is a member of **fstream**, **ifstream**, and **ofstream** objects.

void open(const char* filename, ios::openmode mode);

- Here, the **1st argument** specifies the **name** and **location** of the **file** to be opened and the **2nd argument** defines the **mode** in which the file should be opened.
- To perform file processing in C++, the header files **<iostream>** and **<fstream>** must be included

File Modes

- File_Mode is an optional parameter with a combination of the following flags:
  - `ios::in` - open file for input operation
  - `ios::out` - open file for output operation
  - `ios::app` - output appends at the end of the file.
  - `ios::trunc` - truncate the file and discard old contents.
  - `ios::binary` - for raw byte IO operation, instead of character-based.
  - `ios::ate` - position the file pointer "at the end" for input/output.
- You can set multiple flags via bit-OR (|) operator, e.g., ios::out | ios::app to append output at the end of file.
- For output, the default is **ios::out | ios::trunc**. For input, the default is **ios::in**.

Closing a File

- When we are finished with our input and output operations on a file we shall close it *(so that the operating system is notified and its resources become available again)*
- For that, we call the stream's member function **close()**. This function takes flushes the buffers and closes the file:

myfile.close();

- Once *close()* function is called, the stream object can be re-used to open another file, and the file is available again to be opened by other processes.
- In case that an object is destroyed while still associated with an open file, the destructor automatically calls this function.

Writing on Text File

- Text file streams are those where the **ios::binary** flag is not included in their opening mode. They are designed to store text and thus all values that are input or output from/to them can suffer some formatting transformations.
- Writing operations on text files are performed in the same way we operated with **cout**. Example:

```
ofstream myfile;
myfile.open ("D:\\Test\\Example.txt");  // open Text file for output
if (myfile.is_open())   {   // open successful
    myfile << "This is a line.\n";  // write the first line to text file
    myfile << "This is another line.\n";     // write the second line
    myfile.close();
 }else
    cout << "Unable to open file";
```

Reading from Text File

- Similar with writing to text file, reading from a file can also be performed in the same way that we did with **cin**
- The steps are:
  1. Construct an istream object.
  2. Connect it to a file (open file) and set the file mode operation.
  3. Perform output operation via extraction << operator or **read**(), **get**(), **getline**(),.. functions.
  4. Disconnect (close file) and free the istream object.

```
#include <fstream>
 ......
ifstream fin;
fin.open(filename, mode);
......
fin.close();
```

Reading & Writing Text File - Example

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

int main() {
    // Write to file
    ofstream fout ("D:\\Example.txt");  // default mode is ios::out | ios::trunc
    if (!fout)
        return 1;
    fout << "This is a line."<< endl;
    fout << "This is another line."<< endl;
    fout.close();

    // Read from file
    ifstream fin("D:\\Example.txt");  // default mode ios::in
    if (!fin)
        return 1;
    char ch;
    while (fin.get(ch))   // till end-of-file
        cout << ch;
    fin.close();
    return 0;
}
```

Reading & Writing Text File - Example

```
int a = 2019, b = 11;
float f = 2019.11;
char s[80] = "Testing # ";
ofstream fout;
fout.open ("D:\\test\\Example.txt") ;  if (!fout)  return ;
fout << s << endl << "2019  11.2019  11 \n"
          <<  --a << "  " << ++f << "   " <<  b ;
fout.close();
ifstream fin ("D:/test/Example.txt") ;  if (!fin) return ;
fin.getline(s, 80);
fin >> a >> f  >> b;
cout <<s<<a<<'*'<<b<<'*'<<f<<endl;  // => Testing # 2019*11*11.2019
fin >> a >> b  >> s[2] >> f;
cout <<s<<a<<'*'<<b<<'*'<<f<<endl; // => Te.ting # 2018*2020*11
fin.close();
```

Quiz

Investigate

1. Text File in UTF8.
2. Text File in UTF16-LE
3. Text File in UTF16-BE
4. HTML /XML File
5. Binary File
6. geline(), fail(), clear(), eof(), peek(), putback(), read(), write(), seekg(), tellg(),... member functions
7. File Buffer /Cache

End!