MH1402 Algorithms & Computing II

Lecture 9 File Input/Output

Wu Hongjun

Overview

- **File Output**
  - **ofstream**
- **File Input**
  - **ifstream**
- **File open/close**

Computer Input/Output

- **A computer gets input data from keyboard, mouse, hard drive (including USB flash drive), mic, camera, network interface (wired/wireless) ….**
- **A computer gives output to screen, hard drive, speaker, printer, network interface ….**
- **In this course, we learn the following input/output:**
  - **Input: keyboard (std::cin)**
    read from file
  - **Output: screen (std::cout)**
    write to file

Computer Input/Output

- Files are typically located on reliable storage media (hard drive, USB flash dive, CD/DVD)
- It is convenient to use file to store/organize data on computers
    - Sometimes we need to analyze megabytes (or even gigabytes) of data
    - Typing on the keyboard is impossible for a lot of input data
    - Printing to the screen is inconvenient for a lot of output data
- It is important to learn how to access files (read/write)

Input/Output in C++

- **C++ uses streams to perform input/output operations in sequential media (such as keyboard, screen, file)**
- **cin is the standard C++ input stream   (cin is the stream of what we typed on the keyboard)**
    - **Example :**

```
int x, y;
cin >> x >> y;
```

We type two values in console window (input stream cin now contains two values we typed);

the extraction operator “>>” extracts the first value from cin, assign it to x; the second value in cin is extracted from cin, and assigned to y.

Input/Output in C++

- **cout is the standard C++ output stream (cout is the stream of what we want to print to the screen)**
  - **Exampe :**

```
int x = 3, y = 4;
cout << x << y;
```

  These two values are inserted into the output stream cout using the insertion operator << ; then cout is printed to the screen

- **We can create our own input/output streams to read/write data from/to different places (such as file)**
  - **The use of extraction/insertion operators are the same as cin/cout**

File Input/Output in C++

- **Declaring a file stream is similar to variable declaration**

```
#include <fstream> // must include fstream
int main()
{
    ifstream fin;      //an input file stream with name fin
                       //other name instead of fin can be used

    ofstream fout;     //an output file stream with name fout
                       //other name instead of fout can be used
    /* … */
    return 0;
}
```

File Output in C++

- **A simple program: write “Hello World!” to a file hello.txt**

```
#include <fstream>               // for ofstream (output file stream)
using namespace std;

int main()
{
    ofstream fout;               // Step1: declare an output file stream fout
                                 //        fout can be changed to other name
    fout.open("hello.txt");     // Step2: Open the file hello.txt for writing
                                 //        Note the use of double quotations here
    fout << "Hello World!";     // Step3:  Write the string to the file
    fout.close();               // Step4:  Close the file after writing
    return 0;
}
```

File Output in C++

Running the program given in the previous slide, an output file hello.txt is generated (in the same directory as the executable file)

Note that between the output file stream member functions open() and close(), writing to a file is very similar to using cout to print to screen.

File Output in C++

- **Another program: write 10000 random numbers to a file random.txt**

```
#include <fstream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
   ofstream fout;
   fout.open("random.txt");

   srand(time(0));
   for (int i = 0; i < 10000; i++)
      fout << rand() << endl;

   fout.close();
   return 0;
}
```

File Output in C++

- **By default, output file is located in the same directory as the executable file**
- **We can specify the directory of the output file**
  - **We should ensure that such directory already exists; otherwise, we cannot write to a file in that directory**
  - **Note that we should ensure that we have "write" permission in that directory; otherwise, we cannot open (create) the file for writing.**
  - **We do not have write permission in some directories.**
    For example, the directories of another user on your computer,
    or the root C:\ directory on some computers

File Output in C++

- **We can specify absolute directory:**
  Example: we want to write to file C:\temp\foo.txt

```
    ofstream bar;
    bar.open("C:\\temp\\foo.txt");   // we  use double backslashes \\
```

- **We can specify relative directory:**
  Example: there is a directory qux in the same directory as the executable program, we want to write to the file   qux\foo.txt

```
   ofstream bar;
   bar.open("qux\\foo.txt");            // we use double backslashes \\
```

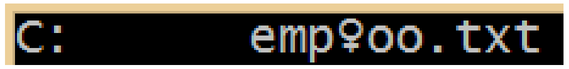
File Output in C++

- **Why double backslashes?**

```
bar.open("C:\\temp\\foo.txt");
bar.open("qux\\foo.txt");
```

- **In a string, a single backslash is a so-called 'escape' character. It is used to include special characters like tab (\t) or a new line (\n). (recall Lab 0)**
  - **If we print out the string with single slash:**

    cout << "C:\temp\foo.txt");

    the output is: 

    ```
    C:        emp♀oo.txt
    ```

    (the backslash characters disappear)
  - **If we want to specify a 'real' backslash in a string, we should use double backslashes**

File Output in C++

- When we use the ofstream member function open() to open a file foo.txt,

  Example :
  ```
  ofstream bar;
  bar.open("foo.txt");
  ```
  - If the file foo.txt does not exist in that directory, a new file foo.txt is created.
  - If the file foo.txt already exists in that directory, the original file foo.txt gets overwritten (i.e., the original data gets lost)! Be careful!

File Output in C++

- If we want to write data into an existing file, but do not want to overwrite it, we use the append mode when we open the file:

Example:

```
ofstream bar;
bar.open("foo.txt", ios::app);
```

File Input in C++

- **Now we read data from files**
- **A simple example (code in next page):**
  - **there is a file sample.txt containing 6 integers, we want to read those integers into an array**

sample.txt

```
13  14  15
17
22  2457
```

```
#include <fstream>              //for ifstream (input file stream)
#include <iostream>
using namespace std;

int main()
{
    int foo[6];
    ifstream fin;                 // Step1: declare an input file stream fin
                                  //          fin can be changed to other name
    fin.open("sample.txt");   // Step2: Open file sample.txt for reading
    for (int i = 0; i < 6; i++)
        fin >> foo[i];            // Step3:  Read data from the file
    fin.close();                  // Step4:  Close the file after reading

    for (int i = 0; i < 6; i++) cout << foo[i] << endl;
    return 0;
}
```

File Input in C++

- **Suppose that we do not know how many data in a file, and we want to read all the data**
  - **We need to use vector to store the data**
  - **If there is no data to read, the extraction operator   >>   returns false**

    We can use this information to stop reading.
- **Example given in the next page.**

```
vector<int> foo;
ifstream fin;

fin.open("sample.txt");
int x;
while (fin >> x)
    foo.push_back(x);

fin.close();

for (int i = 0; i < foo.size(); i++)
    cout << foo[i] << endl;
```

File Input in C++

- How to read each line of a file as a string? We use getline
  (The spaces in the line are also read as part of a string.)
  Example: Read all the lines into a vector of strings

```
string line;
vector<string>  foo;
ifstream fin;

fin.open("sample.txt") ;

while ( getline (fin, line) )     //getline returns false when there is no line to read.
        foo.push_back(line);

 fin.close( );
```

File Input in C++

- **How to read a file if it consists of words, int, double?**
  - **For example: A text file storing products' name, price and score**
    We want to read the name into a vector (string), the price into a vector (double), and the score into a vector (int) for data processing

games.txt

```
Wii             255.7    8
PC               1230     5
MAC              1500     6
PS3             600       3
XBOX             599.9   7
```

```
vector<string> gameName;  vector<double> gamePrice; vector<int> gameScore;
ifstream fin;

fin.open("games.txt");
string name;  double price;  int score;
while ( (fin >> name) &&  (fin >> price ) && (fin >> score ) )
{
     gameName.push_back(name);
     gamePrice.push_back(price);
     gameScore.push_back(score);
}
fin.close();
```

File Input in C++

- **A more complicated example**

  A text file storing the students' exam marks

  Note that a name may consist of one, two, three, or more words.
  We want to read the names into a vector, and the marks into another vector.

marks.txt

```
Lee Seng Yi                        81.1
Lim Jia Ren                        82.3
Richard Goh Xin Hao                83.4
Tan Xin Chee                       84.5
Teo Shannon                        86.7
Wong You Pei                       87.8
```

We have a simple solution if the format is regular, for example, in each line, the marks starts at the 31st character position

```
#include <fstream>
#include <vector>
#include <string>
#include <cstdlib>
……..
vector<string> student_names;
vector<double> student_marks;


ifstream fin;    fin.open("marks.txt");


string line, name, marks_str;
double marks;
```

```
while (getline(fin,line))
{
    name = line.substr(0,30);
    for (int i = name.size()-1; i >= 0; i--)
    {
        if (name[i] != ' ') break;
        else name.erase(name.end()-1);
    }

    marks_str = line.substr(30,4);
    marks = atof(marks_str.c_str());

    student_names.push_back(name);
    student_marks.push_back(marks);
}
```

In the previous slide,

1) String name is extracted from the first 30 characters of line (a string)
   Then the empty space at the end of the name is removed.

```
name = line.substr(0,30);
for (int i = name.size()-1; i >= 0; i--)
{
   if (name[i] != ' ') break;
   else name.erase(name.end()-1);
}
```

2) String marks_str is extracted from the next 4 characters

```
marks_str = line.substr(30,4);
marks = atof(marks_str.c_str());
```

   Then function atof(s.c_str()) converts a string (corresponding to a numerical data) into floating-point (double). To use atof(), #include <cstdlib>

File close( )

- **Why do we close a file after reading/writing?**
  - **In general, closing a file after reading/writing is good for system efficiency The operating system does not need to keep tracking those opened files.**
  - **If we do not close a file opened for reading, it can get blocked for concurrent writing**
  - **If we do not close a file opened for writing, some file data may get lost:**

    The data being written into a file is not written into the file immediately, since some data may be temporarily stored in memory buffer. It is to avoid writing data to the disk too frequently (slow) when each data is small. When we close a file, it ensures that all the data (of that file) that are still in the memory buffer get written into that file.
  - **Close the file also prevents data from being written into that file by accident**

File close( )

- **If we want to ensure that the data gets written into a file (instead of in the memory buffer), but we want to continue writing into the file, we can use the ofstream member function flush( )   (slow)**
  Example:

```
ofstream fout;
fout.open("sample.txt");
for (int i = 0; i < 10; i++)
{
   fout << i << endl;
   fout.flush();  //data gets written into the file
}
fout.close();
```

File open( )

- In previous examples, the file name is given directly in the open function, for example:

```
ofstream fout;
fout.open("sample.txt");
```

- But we cannot open a file in the following way (compilation error):

```
string foo = "sample.txt";
ofstream fout;
fout.open(foo);
```

- Reason: The function open() is old, and it does not recognize C++ string (tricky)
- Solution: Convert the C++ string to C string using the c_str member function of string:

```
string foo = "sample.txt";
ofstream fout;
fout.open(foo.c_str( ));
```

- This solution is important if we need to get file names from user input or to generate new files names in our program

File open( )

- **Can we always open a file for “write”?**
  - **If a file exists already but is protected (read only), cannot open it for “write”**
  - **If there is insufficient storage space, we cannot open a file for “write”**
  - **If the specified directory does not exist, we cannot open a file in that directory for “write” (in C++)**
  - **If a program does not have the “write” permission in a directory, cannot open a file for “write”**
  - **Invalid file name (for example, consisting of strange characters), cannot open a file for “write”**
- **Can we always open a file for “read”?**
  - **If the file does not exist, we are not able to open it**
  - **If we do not have permission to read that file, we are not able to open it.**

- **How to handle if the file “open” failed?**
  - **A program can be modified to detect the failure of file “open” before accessing it. One approach is to use the function is_open(), which returns true to indicate that the stream is currently associated with a file; otherwise returns false. We may use exit(1) to exit the program, or prompt the user to provide new file name.**

```
ifstream fin;
fin.open("foo.txt");
if (fin.is_open())      // is_open() can also be applied to file writing
{
   /*… your code to read data from the file foo.txt*/
}
else
{
   cout << "cannot open the file for reading";
   exit(1);   //exit from the program, 1 means general error
}
fin.close();
```

More on File Write in C++

- If file "open" is successful, can we guarantee that all the data can be written into that file?
  - If there is insufficient storage space, there will be error in the writing.
  - If the data size is too large, then there may be error in the writing
    - FAT file system (most of the USB flash drives), file size limit is 4GB (GigaBytes)
    - NTFS file system (by default in Windows 7), file size limit is at least 16TB (TeraBytes). Large enough for almost all the applications today