At this time, you have already familiar with the standard input and output stream in c++ which are “cin” and “cout”. The standard input / output by default is the screen.

In this chapter we are focusing on “file stream” to complete the basic file I/O jobs. Read from file and write to file(file is normally a .txt file). There are some predefined libraries we can use to do file I/O. But before we start, firstly you need to know the common concept listed below:

* It doesn’t matter you want to read or write, always open the file stream in the beginning and close the stream after all the works done.
* It is allowed to write to a file which dost not exist yet. If you use write to file stream but the file does not exist, the system will create the file with the file name you put in the parameter when you open the writing stream
* You can write to a file which doesn’t exist but you can never read from a file which doesn’t exist !

So base on the rules, let’s see how do we use file stream to read and write. There are coupe ways to do it.

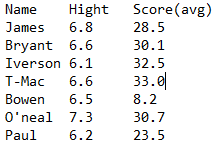
1. Using <fstream> library

fstream is the basic file stream which you can do read or write after you create this stream. You just need to mention it when you open the file stream.

Example: ( read from file line by line into a string )

The input file look like this ( I also put this txt file in the folder which you can use is to practice, or you create your own txt file)

Input file: “NBA\_player.txt”



Sample project:

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main()

{

string s;

fstream f;

f.open("NBA\_player.txt", ios::in);

while (!f.eof())

{

getline(f, s);

cout << s << endl;

}

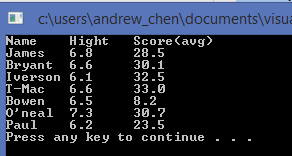
f.close();

system("pause");

return 0;

}

Output:

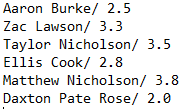


Notice the function ”eof()“ which checks the end of file. Since you open a file input stream, so you can check the where are you reading from the file. In this case we use f.eof() (f is the stream name) to check do we finish reading all the content from the file.

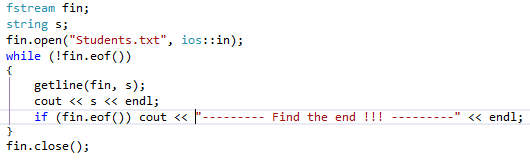
There is a common problem you might meet:

If the eof() function did not detect the end bit at the correct time, you may not able to finish reading at the correct ending point. Here is a sample case:

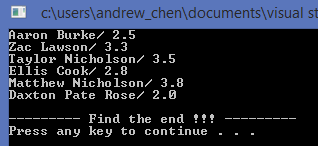
Input file: “Students.txt”



Main function:



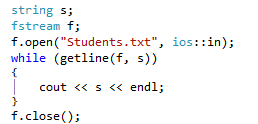
Output:



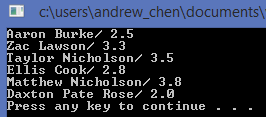
You can see there is an empty line between the last line of file and the “Find the end” message. This empty line actually come from one extra turn of reading from file. It happens because when the system read the 6th line (last line) into string s, it has not detected the end point. The while loop didn’t stop (stop condition is eof() is true). system read again from file and store into string s (but actually read nothing so if you print the s.length(), you get 0). Then after this reading nothing turn, system finally detect the end point to stop the loop.

If you want to avoid this situation, I suggest you change the stop condition of the while loop. Check the example below:

Main function:

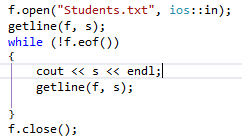


Output:



The different in this while loop is the stop condition. I directly put the reading instruction in the while loop condition. So if there is nothing to read (getline will not consider to read ‘\n’), the loop stop.

Another way to fix this problem:



Just change the order that you read the first line before the while loop start, then you can still use eof() function to complete the job.

Just remember this solution in case you need to spend so much time to debug if

the problem happened ~

I will discuss more example in SI session include other reading method

(e.g. fin.getline , fin.get , fin.ignore() … )