It’s almost the end of this semester and I believe you guys are doing well. Before the thanksgiving break, professor go through the file stream( fstream ). There are actually some functions you need to know about fstream.

The first thing to know is:

There are actually two pointers( or cursor ) working with the file stream,

One is for read, it point to the current location you are reading. ( to “get” the content )

* You can use “tellg()” function to get the location, this function will return a streampos number which tell you the location count in bytes
* If you want to change the location to read, “seekg()” function can do the job for you.

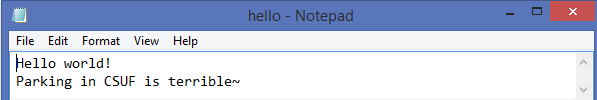
There are two ways to setting the reading pointer (“seekg” is an overloading function with different parameters)

1. seekg ( position )

You can directly set the distance (characters) you want to move.

Example:

There is a file “hello.txt”



Now you want to ignore the first 6 bytes(characters) and start reading from “world!”

You need to set the reading pointer 6 bytes move to the right hand side. Use positive number to complete (The same thing use negative to move to left hand side)

Let’s see the code:

int main()

{

fstream file;

string s;

file.open("hello.txt", ios::in | ios::out);

file.seekg(6);

file >> s;

cout << s << endl;

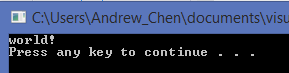
file.close();

system("pause");

return 0;

}

I just move the pointer to the right side (6 bytes distance) to ignore the “Hello” and “white space”. Then I read the world! In a string s to output it.



1. seekg( offset, direction)

There are three predefined direction which are beg(begin), cur(current), and end

In the parameter you need to do like this:

* ios::beg
* ios::cur
* ios::end

Example:

If you want to know the true size (count in bytes) of the file you are reading, just move the pointer to the end of the file (ios::end) and then use tellg to return the current location you are reading. Since you set the pointer to the end by seekg, the current pointer for reading is at the end of file. So the tellg will return the ending location to you ( which means how many bytes in front of the end point)

Let’s see the read code:

int main()

{

fstream file;

string s;

file.open("hello.txt", ios::in | ios::out);

file.seekg(0, ios::end);

cout << "file size is " << file.tellg() << " bytes" << endl;

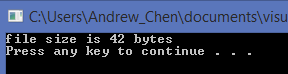
file.close();

system("pause");

return 0;

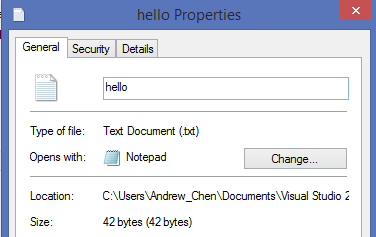
}

Output:



I set the reading pointer to the end (with 0 offset moving). The just use tellg to return the current location (end of file), notice the change line in the file will be transfer to “\n” which is also one byte ( or two bytes … “\” and “n” )

You can check your file properties to confirm the result.



The other thing “write to file” is almost the same concept. Just remember if you want to read and write by the same file stream, always set the pointer before you write (use seekp to set the writing pointer and tellg to return the location of writing pointer).

\*Remember the read pointer and writer pointer are different. If you change the reading pointer by seekg() function, the write pointer will not be change. You still need to use seekp() to change the writing pointer. And if you are writing things into the location which exist any data. The data will be replaced by the new content you just written in the file.

Let’s see the example:

int main()

{

fstream file;

string s;

file.open("hello.txt", ios::in | ios::out);

file >> s;

cout << "The current reading pointer loaction is: " << file.tellg() << endl;

file.seekp(file.tellg());

file.seekp(1, ios::cur);

file << "^(o)^";

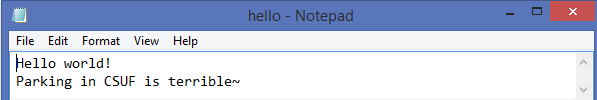
file.close();

system("pause");

return 0;

}

In this case, I still use the same file (hello.txt) like this:



I firstly read “Hello” into a string variable s, then I call tellg() to see where the reading pointer is. It return 5 because “Hello” use 5 characters (5 bytes). Now I want to write something to the file to replace the “world”. So I need to set my writing pointer to the same location and then move 1 more byte to ignore the white space.

file.seekp(file.tellg()); //Set writing pointer to the same place as reading pointer

file.seekp(1, ios::cur); //Move writing pointer 1 byte to right hand side

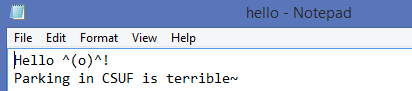
\*Notice\*

If you use the one parameter seekp function already, then you need to use the two parameter seekp function if move it again, or the system will reset the pointer to the beg before the second move

Now I am ready to write 5 bytes things into the file to replace the “world”

file << "^(o)^";

The Result:



The “world” has been replaced by “^(o)^”

Another writing mode is app (append). App means you want to write things after the end of the file, so nothing will be replaced by anything you write into the file.

It’s very simple, so we can see the example:

int main()

{

fstream file;

file.open("hello.txt", ios::app);

file << "\nfrom here is new content append!";

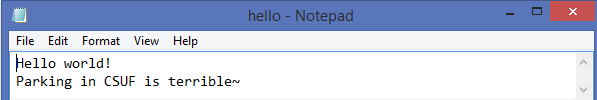
file.close();

system("pause");

return 0;

}

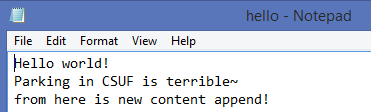
The same, the source file is hello.txt



I just want to add one more sentence after the end of the file.

file << "\nfrom here is new content append!"; // ”\n” for writing in a new line

Let’s see the result in the file:



\*So app is the same you use ios::out but set the write pointer to the end of the file

Like this:

int main()

{

fstream file;

file.open("hello.txt", ios::in | ios::out);

file.seekp(0, ios::end);

cout << file.tellp() << endl;

file.seekp(1, ios::cur);

cout << file.tellp() << endl;

file << "\nfrom here is new content append!";

file.close();

system("pause");

return 0;

}

Notice this line: file.open("hello.txt", ios::in | ios::out);

You must add an ios::in or ios::app with ios::out, so you can successfully use file.seekp(0, ios::end). If you just use it alone: file.open("hello.txt", ios::out), then file.seekp(0, ios::end) will fail. You will start from the beginning of the file. I’m not sure the real reason for that and still trying to find the answer ~

Hope this will help you

Again, we still have the SI session every Mo/We! Feel free to come anytime !