FILE INPUT OUPUT

We have seen how to output to a display (cout) and input from the keyboard (cin). C++ treats cout and cin as streams to which we can output (cout) or from which we can input (cin) data. Displaying “hello”, for example, is adding to the stream of data to be displayed by the monitor.

What if we want to send data to a file instead of to the monitor? Or we have to input data from an existing file instead from the keyboard?

Let’s see how to output data to a file and input from a file.

OUTPUT TO A FILE

We have used #include <iostream> at the top of our programs to allow input/output from the keyboard and to the display respectively.
If we want the output to be directed to a file instead, we will need #include <fstream> in our program.

We will also add #include <string> to be able to use variables of type string and use strings in our program. Note that this is independent from the file input/output requirements.

The lines that are needed to allow our programs to output to a file are displayed below in this color.

We have already explained that #include <fstream> is needed to allow file input/output. How can we use this capability?

The line

```
ofstream outfile ("testing.txt");
```

associates a name we choose for the file we want to send the output to with a variable of type ofstream (output to file stream). We can choose the name of that variable. Once we have chosen to and associated this variable (outfile) with the name of our actual file on our disk (i.e. testing.txt), we will use outfile as we use cout in our programs when we want to display values on the monitor.

That is, the output operation will work similarly: the line

```
cout << "Hello" << endl;
```

would send “Hello” on our monitor, represented by cout, while

```
outfile << "Hello" << endl;
```

will send “Hello” to the file represented by outfile,
The name outfile was chosen by us, We could have used a different name

```
ofstream whatever ("testing.txt");
```

and our external file testing.txt would be represented in our program by the variable whatever instead of outfile. Of course, in this case, would would have to replace the output statement with

```
whatever << “Hello” << endl;
```

The string “Hello” would be written to the SAME external file testing.txt, this time represented by the variable whatever.

Finally, once we are done writing to the file, we close it by calling

```
outfile.close();
```

INPUT FROM A FILE

In the example below we will use the same file (testing.txt) we wrote to and we’ll input the two strings we wrote to it into the variable line, of type string.

This time, we associate our file testing.txt with the ifstream variable infile. As we saw before, the name of the variable (infile) is chosen by us to represent the actual file (testing.txt), much like cin represents the keyboard.

In this program, using >> to input the strings on disk into the string variable line could be done much as we did with cin >>.

```
infile >> line;
```

However, if we do it this way, we will get the original strings one word at a time. That is, a string with spaces will have to be loaded by several input commands. If we want to input the whole string, spaces and all, we will have to use the function getline. To use getline, we have to specify the input stream (cin, or, in this case, infile) and the string variable (line) we want the data to be read into.

Thus the lines:

```
 getline (infile,line);
  cout << line << endl;

    getline (infile,line);
  cout << line << endl;
```

As you can see, the same two lines are repeated so we can input the two strings, one at a time, into variable line.

The best way to learn about this is by running te following code.

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

int main () {

  ofstream outfile ("testing.txt");
  string line;


    outfile << "Hello" << endl;
    outfile << "Nice to meet you"<<endl;
    outfile.close();

//now let's use the same file as input

    ifstream infile ("testing.txt");


    getline (infile,line);
   cout << line << endl;
     getline (infile,line);
   cout << line << endl;

infile.close();

return 0;



}
```