STRING LAB : VALIDATING USER INPUT

" 'Begin at the beginning,' the King said, very gravely, 'and go on till you come to the end: then stop.' "– Lewis Carroll, Alice in Wonderland, [1]

0.1. **Lab objectives.**

In this lab you will learn to handle input and output in your program, with emphasis on string as a tool. In the first part of the lab you will learn about input and output through console. In the second part, you will learn to handle input and output by reading from and writing to files. Through these exercises that require user input manipulation, you will become aware of the importance of input validation.

1. Input/output on the console window

1.1. **Introduction**.

The vast expanse of black screen greets you as you open your eyes. You must have fallen asleep on your keyboard and without knowing initiated the console window. Suddenly, white letters appear at the top of the screen.

> Hey, are you there?

The cursor is blinking as you try to figure out what had just happened. You type and enter: yes. The reply is instant.

> Want to find out how I'm talking to you? Say yes, and you'll find out.
> Say no, you'll go back to sleep and forever wonder if this encounter
> was a dream.

This doesn't leave you with much of a choice. You are about to learn about input and output through the console window. When you write a program that prints to and reads from the console, you need to include the iostream library.

1.2. iostream **library: input and output using objects cout and cin.**

The iostream library provides input and output functionality using streams.[2] A stream is an abstraction which represents, in this case, the console. The objects cin and cout belong to iostream library. Insertion operator (<<) and extraction operator (>>) are required to perform input/output operations. You have already seen the insertion operator[1] from the earlier parts of

---

[1]As a refresher, here is an example printing different variables with cout :

```
int numApples = 3;
int numOranges = 4;
string container = "box";
cout << "There are " << numApples << " apples and " << numOranges << " oranges in the " <<
    container << endl;
```

This will print to the console the message : There are 3 apples and 4 oranges in the box

the course. The “Hello World! ” program uses cout and <<. cout << inserts messages to the console.
To recreate the conversation in the introduction you may write

```
cout <<  "Hey, are you there?" << endl;
```

endl adds a newline character to the end of the message. It is equivalent to writing a newline character ‘\n’ within the quotation marks:

```
cout << "Hey, are you there?\n";
```

Both expressions print to the console the message:

```
Hey, are you there?
```

Now it is time for you to write a program that will accept user input, and respond accordingly. First, you need to know how cin and extraction operator >> work. Put together, cin>> extract formatted data from user input. There are two major sources of error in using cin>>: when the user inputs the wrong type of data and when the user inputs more data than necessary for the specific operation. When the user inputs the wrong type of data, the stream enters the fail state and can no longer get user input from the console. This state can be checked with cin.fail(): after using cin>>, calling cin.fail() will return true if the user input an incorrect type and false if the user input the correct type. Consider the following code snippet:

```
int x;
cin >> x;
if (cin.fail()) cout << "fail" << endl;
else cout << "success" << endl;
```

Here, if the user inputs “66u”, cin.fail() will return true and “fail” will be printed to the console. If the user inputs “66”, cin.fail() will return false and “success” will be printed to the console.

The less obvious error is due to extra characters in the buffer. cin>> consumes all blank spaces and stops reading after extracting a single input. The error arises from the fact that cin>> keeps the rest of user input after extraction of formatted data. The remaining characters can be manipulated[2] further or mess up other input if unexpected. Correct use of the operation is given in the following example. If the user inputs 7 8 9 in console for this line of code:

```
cin >> numApples >> numOranges >> numGrapes;
```

---

[2] As a refresher, here is an example printing different variables with cout :

```
int numApples = 3;
int numOranges = 4;
string container = “box”;
cout << “There are ” << numApples << “ apples and ” << numOranges << “ oranges in the ” << container << endl; 1
```

This will print to the console the message : There are 3 apples and 4 oranges in the box

numApples will have 7, numOranges 8, and numGrapes 9 assigned. However, remaining characters in the buffer can be a cause of great vulnerability. Let's examine code[3] for manipulating ATM transactions :

```
#include <iostream>

using namespace std;

void transferMoney(int value);

int main(){
  int pin;
  int value;
  cout << "Please enter your PIN: " << endl;
  cin >> pin;
  if (!cin.fail() && pin == 12345) {
    cout << "Please enter the amount to be ";
    cout << "transferred (in hundreds of dollars): " << endl;
    cin >> value;
  }
  if (value > 0) {
    transferMoney(100*value);
    cout << "You have transferred $" << 100*value
        << "." << endl;
    cout << "Thank you for your transaction. " << endl;
  }
  return 0;
}

void transferMoney(int value) {
  // this can be blank for the sake of this example
}
```

Run the program. When asked to type in the PIN, you write by accident the PIN and an extra digit, separated by a space. For example: 12345 6. What is printed to the console? Why does this happen?

[3] The idea of the example comes from Keith Schwarz's CS106L coursereader. [5]

Another standard function you can use is getline( )[4]. It reads user input into a string until the newline character[5] and does not leave any extra characters in the buffer. Here is an example of getting a name input:

```
string name;
getline(cin, name);
```

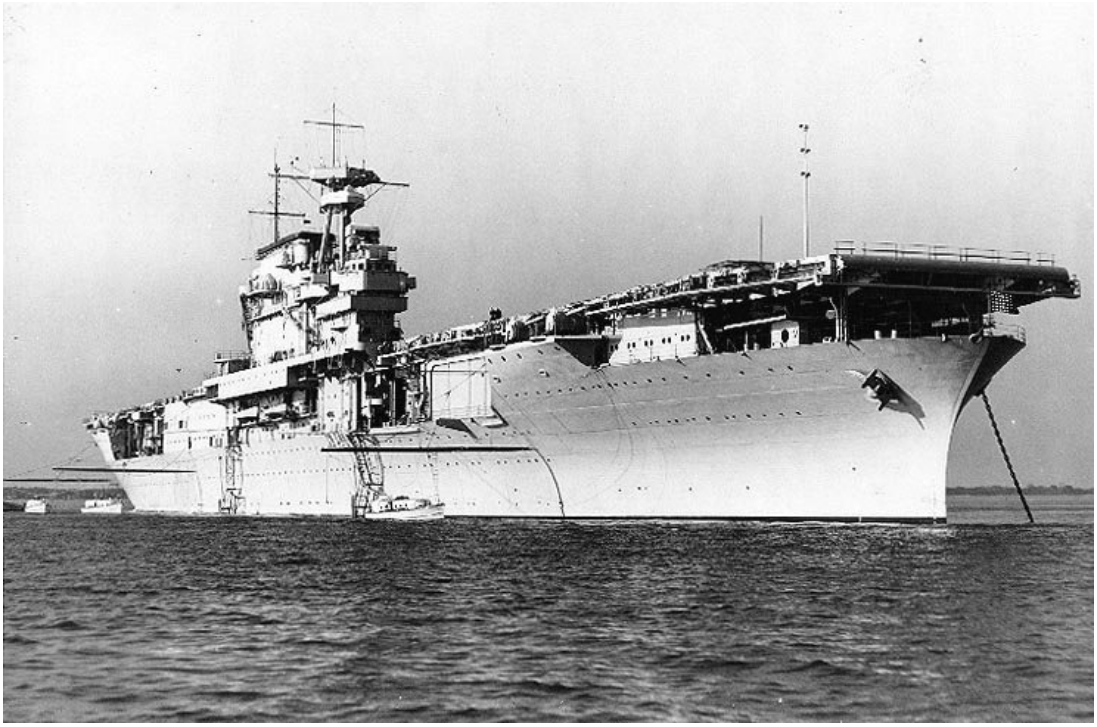
Here, when the user types a name, the entire line is stored in the string "name", including white spaces.

[4] Safer version of getline with checks on fail state. From github.com [3]:

```
string GetLine() {
        string returnString;
        while (true) {
                getline(cin,returnString);
                if (!cin.fail()) break;
                else {
                        returnString="";
                        cin.clear();
                }
        }
        return returnString;
}
```

[5] If you want to use delimiters other than the newline character you can specify by using a variant of getline. For example, if ',' were a delimiter you write : getline(cin, str, ',') However, with any other delimiter than the newline character, you will face the same problem of leftover input as cin>>.

USS Yorktown Software Disaster



Source: http://ww2db.com/image.php?image_id=6865

In November 1998 the USS Yorktown, a guided-missile cruiser that was the first to be outfitted with "Smart Ship technology", suffered a widespread system failure. "After a crew member mistakenly entered a zero into the data field of an application, the computer system proceeded to divide another quantity by that zero. The operation caused a buffer overflow, in which data leaked from a temporary storage space in the memory, and the error eventually brought down the ship's propulsion system. The result: the *Yorktown* was dead in the water for more than two hours." [6]

Writing a safer integer input function: stringstream

`stringstream` provides an interface to manipulate strings as if they were input/output streams[4]. The extraction operator >> and insertion operator << function the same way with `stringstream` as with `iostream`. `Stringstream` objects also have a `.fail()` function that works in the same way as with the `cin` object. For example, consider the following code snippet:

```
stringstream converter;            //declaring sstream object
int x;
string result;
converter << getline(cin, result);
converter >> x;   //works similarly to cin
```

Let us examine some pseudocode for an implementation of a safer integer input function, using stringstream.

```
int GetInteger(){
/* 1. Declare stringstream object */
/* 2. Get a line of user input as a string, then write into the object.*/
/* 3. Extract an integer from the string*/
/* 4. Do error checking : 1) fail state 2) extra characters*/
}
```

Note that on an invalid input, GetInteger() should rebuke the input and re-prompt the user to reenter until a valid integer is written. To check if there are extra characters, you may find the .eof() function of the stringstream object useful.

Although many functions are commonly used, it is always good practice to place checks for errors, and write your own modified versions if necessary. Using the knowledge gained from previous sections of this lab, implement the GetInteger()[6] function.

```
int GetInteger(){




}
```

How does your implementation protect against extra unnecessary characters in the input?

How does your implementation correctly ensure that the input is a valid int?

[6] Implementation from Keith Schwarz's CS106L course reader. [5] Only some comments were changed.

Exercises.
1) Spot the errors in the following piece of code:

```
int number;
string str;
cin >> number;
getline(cin, str);
```

(Hint: `getline` reads until it finds a newline character.)

3) Fixing ugly code: Improve the ATM example by inserting the safer input function in the right places (you only have to reimplement the `main` method).

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;
void TransferMoney(int value);

int main() {









}
```

2. File input/output

Introduction.
The fstream library lets you read from and write to files with two classes: ifstream (short for input file stream) and ofstream (short for output file stream). As you may have guessed, ifstream is used to read from files and ofstream is used to write to them.

Let us write code that will take in the file name and open the file. Table 1 below lists the functions in fstream associated with opening a file.

Table 1

file.open(filename)	This method attempts to open the file named by filename and attach it to the stream file. Note that the filename parameter is a C-style string, not a C++ string object. You can convert a C++ string to its C-style equivalent with the string method c_str. You can check whether open fails by calling the fail() method.
file.close()	This method closes the file attached to the stream file.
file.fail()	This method returns a bool indicating the error state of the stream file. A result of true means that a previous operation on this stream failed for some reason. Once an error occurs, the error state must be cleared before any further operations on the file will succeed.

To open the file, you need to know the filename. If you already know the file name, you can incorporate it in the program itself. However, to be more generic, you will need to get the user's input (using getline()).

If the user enters the wrong file name, all following manipulation with file data will be rendered useless. To ensure that the file exists and opens correctly, you need to place a check (also, you need to keep asking the user until they enter a valid file name). Using the functions from table 1, write a function that prompts the user for a file name, opens the file if it exists, attaching it to the provided ifstream, and keeps prompting the user if it does not.

```
void getFile(ifstream &file) {









}
```

You have successfully completed opening the file! Now you need to read the characters in the file using `ifstream` and write to a separate file using `ofstream`. Opening an existing file is simple, as seen before, but what if you want to open a new file to write to? To instantiate an `ofstream` object with a new file, use the following notation:

```
ofstream outfile(filename);
```

If the file with the given name does not exist, then this will create a new file for you. However, if a file with the same name already exists, then whatever you write to the `ofstream` will not just append to the end of the existing file, but it will overwrite it (so beware!).

Say we had an `ofstream` object called `outfile` and an `ifstream` object called `infile`. Table 2 below summarizes the functions needed to read from an `ifstream` and write to an `ofstream`.

Table 2

infile.get()	This method reads and returns the next character in the input file stream. If there are no more characters, get() returns the constant EOF. If some other error occurs that puts the ifstream object into a fail state, this can be checked by calling ifstream.fail(). Note that the return value of get() is an int, not char.
outfile.put(char ch)	This will write a single character to an output file stream. (Don't forget to call outfile.close() when you're done!)

Using these functions, write a method that takes a file name from user input and copies the contents of the file character by character into a new file named "stringLab.txt". You may use the getFile() function that you've implemented above to read the user's input and open a file.

```
const char *outfileName = "stringLab.txt";

void copyFile() {









}
```

Does your implementation take care not to overwrite any previously existing files? If not, change your implementation to do so and then describe your solution here.

REFERENCES

[1] Carroll, Lewis. Alice in Wonderland. Tribeca Books, 2011. Print.
[2] ”IOstream Library.” cplusplus.com. N.p., n.d. Web. 12 Sep 2011. <http://www.cplusplus.com/reference/iostream/>.
[3] b33tr00t,.”cs106lib/simpio.cpp.” github.com. N.p.,n.d. Web. 12 Sep 2011. <https://github.com/b33tr00t/cs106lib/blob/master/simpio.cpp>.
[4] ”stringstream.”cplusplus.com.N.p.,n.d.Web.12Sep2011.<http://www.cplusplus.com/reference/iostream/stringstream/>.
[5] Schwarz, Keith. ”CS106L Standard C++ Programming Laboratory Course Reader.” www.keithschwarz.com. Stanford University, 2009. Web. 12 Sep 2011. <http://www.keithschwarz.com/coursereader.pdf>.
[6] ”ostream::put.”cplusplus.com.N.p.,n.d.Web.12Sep2011.<http://www.cplusplus.com/reference/iostream/ostream/put/>.
[7] Roberts, Eric, and Zelenski Julie. ”Programming Abstractions in C++.” Stanford University, Sept-12-2011. Web. 12 Sep 2011. <http://www.stanford.edu/class/cs106b/materials/CS106BX-Reader.pdf>.