Basic Input/Ouput

Stream: In C++ (and other related languages) we use the ‘stream” metaphor to describe the relation between out program and the output destination or the input source. Picture our program surrounded by data streams: output and input streams are used, repectively, to receive data from the program (output stream) or to allow the program to extract data from them.

You can visualize your screen as an output stream waiting for you to insert into it any data output you see fit:

cout << “This is a string, but the next is an int “ << 56;

The stream that represents your screen is called (predefined as) **cout**.
The stream that represents your keyboard (for input purposes) is called **cin.**

Other types of stream are used for disk files, for example. Writing to a file can be viewed as inserting data into a file stream, and retrieving data from a file can be viewed as inserting (“fishing”) data out of an file input stream.

Output: For the time being, we will only be sending output to the monitor, not files. Therefore the general statement:

cout << …. << …. << ……   ;

is all we need.

The ellipsis (…) stands for variables or values.

Input : The keyboard as an input stream is represented by **cin.** Values entered from the keyboard are read into the program in accordance with the type of variable defined to store them. An input operation will keep reading from the input stream until all variables mentioned in the input statement have been filled or the input stream has been exhausted (end of file).

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

int main()
{

  int num;
  char c1, c2, c3;

  cout << "Enter a 3 digit number" << endl;
  cin >> num;
  cout << "The number was " << num << endl;

  cout <<  "Enter again the same 3 digit number" << endl;
  cin >>  c1 >> c2 >> c3;

  cout << c1 << " " << c2  << " "<< c3 << endl;

  return 0;
}
```

Let's say that the wrong type of input is entered, for example, letters, when the program expects an integer value. In this case, for instance:

```
  cin >> num;
```

since *num* was defined as being of type int, if a letter is entered instead. input will stop and will not be accepted again unless the input stream is cleared, using *cin.clear().*

You can test for input error using the condition:

if (cin) .....   if no error do something
if (!cin) ....   If there was an input error, do something else

In the next example, we will run the same program above, but this time we will enter three letters (as in *abc,* no spaces) instead of a three digit number. Of course, when we do this, since out program expects an int, an input error will be raised and no more input from *cin* will be processed.

We can remedy this by including a condition in our program that clears the error in *cin* and allows input to proceed. However, note that the next input will be retrieved from the data not consumed by the input statement that had failed earlier. That is, the next input statement in the example below grabs the input waiting in the buffer that was not used by the previous input operation. Thus, the next input statement loads the characters that the had failed to load because of type incompatibility.

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

int main()
{

  int num;
  char c1, c2, c3;
 cout<< "At the beginning, variable num = " << num;
```

```
  cout << "Now, try to input 3 letters –NO SPACES-- into the num variable" << endl;

  cin >> num;


  cout << "The new value if num is  " << num << endl;

 cout << "As you can see, num hasn't changed" << endl;

  if (!cin) {

 cout << "There was an input error because letters couldn't be loaded into the int
variable" << endl ;

cout << "We now  clear the error using cin.clear()" << endl;

    cin.clear();
}

//Now the same letters not consumed in the step before will be loaded into
// the three char variables c1, c2 and c3

cout << "And we load c1 , c2 and c3" << endl;

  cin >>  c1 >> c2 >> c3;

  cout << c1 << " " << c2 << " "<< c3 << endl;

  return 0;
}
```

If you enter ABC on the first prompt, the first input will fail, the input error will be cleared and the second input will read the characters ABC that were waiting in the buffer.