Input/ Output Technique

Until now, the example programs of previous sections provide very little interaction with the user, if any at all. Using the standard input and output library, we will be able to interact with the user by printing messages on the screen and getting the user's input from the keyboard.

C++ uses a convenient abstraction called streams to perform input and output operations in sequential media such as the screen or the keyboard. A stream is an object where a program can either insert or extract characters to/from it. We do not really need to care about many specifications about the physical media associated with the stream – we only need to know it will accept or provide characters sequentially.

The standard C++ library include the header file iostream, where the standard input and output stream object are declared.

Standard output (cout)

By default, the standard output of a program is the screen, and the C++ stream object defined to access it is cout.

cout is used in conjunction with the insertion operator, which is written as << (two "less than" signs).

```
cout << "output sentence";          //prints output sentence on screen

cout << 120;                        // print number 120 on screen

cout << x;                          // prints the content of x on screen
```

The << operator inserts the data that follows it into the stream preceding it. In the example above it inserted the constant string output sentence, the numerical constant 120 and variable x into the standard output stream cout. Notice that the sentence in the first instruction is enclosed between double quotes (") because it is a constant string of characters. Whenever we want to use constant strings of characters we must enclose them between quotes (") so that they can be clearly distinguished from variable names. For example, these two sentences have very different results:

```
cout << "Hello";            // prints Hello

cout << Hello;              // prints the content of Hello variable
```

The insertion operator (<<) may be used more than once in a single statement:

```
cout << " Hello," << "I am " << "a C++ statement";
```

This last statement would print the message Hello, I am a C++ statement on the screen. The utility of repeating the insertion operator (<<) is demonstrated when we want to print out a combination of variables and constants or more than one variable:

```
cout << "Hello, Iam "<< age<< " years old and my zipcode is "<< zipcode;
```

If we assume the age variable to contain the value 24 and the zipcode variable to contain 90064 the output of the previous statement would be:

```
Hello, I am 24 years old and my zipcode is 90064
```

It is important to notice that cout does not add a line break after its output unless we explicitly indicate it, therefore, the following statements:

```
cout << "This is a sentence."

cout << "This is another sentence.";
```

Will be shown on the screen one following the other without any line break between them:

```
This is a sentence. This is another sentence.
```

Even though we had written them in two different insertions into cout. In order to perform a line break on the output we must explicitly insert a new-line character into cout. In C++ a new-line character can be specified as \n (backslash ,n):

```
cout << "First sentence.\n";

cout << "Second sentence.\nThird sentence.";
```

This produces the following output:

```
First sentence.

Second sentence.

Third sentence.
```

Additionally, to add a new-line, you may also use the endl manipulator. For example:

```
cout << "First sentence."<<endl;

cout << "Second sentence."<<endl;
```

Would print out:

```
First sentence.

Second sentence.
```

The endl manipulator produces a newline character, exactly as the insertion of '\n' does, but it also has an additional behavior when it is used with buffered streams: the buffer is flushed. Anyway, cout will be an unbuffered stream in most cases, so you can generally use both the \n escape character and the endl manipulator in order to specify a new line without any difference in its behavior.

Standard Input (cin)

The standard input device is usually the keyboard. Handing the standard input in C++ is done by applying the overloaded operator of extraction (>>) on the cin stream. The operator must be followed by the variable that will store the data that is going to be extracted from the stream. For example:

```
int age;
cin >> age;
```

The first statement declares a variable of type int called age, and the second one waits for an input from cim (the keyboard) in order to store it in this integer variable.

cin can only process the input from the keyboard once the return key has been pressed. Therefore, even if you request a single character, the extraction from cin will not process the input until the user presses return after the character has been introduced.

You must always consider the type of the variable that you are using as container with cin extractions. If you request an integer you will get an integer, if you request a character you will get a character and if you request a string of character you will get a string of character.

```
// I/O example

#include <iostream.h>

int main()

{

    int i;

    cout << "Please enter an integer value: ";

    cin >> i;

    cout << "The value you entered is " << i;

    cout << " and its double is " << i*2 << ".\n";

    return 0;

}
```

This produces the following output:

```
Please enter an integer value: 702

The value you entered is 702 and its double is 1404.
```

The user of a program may be one of the factors that generate errors even in the simplest programs that use cin (like the one we have just seen). Since if you request an integer value and the user introduce a name (which generally is a string of characters), the result may cause your program to misoperate since it is not what we were expecting from the user. So when

you use the data input provided by cin extractions you will have to trust that the user of your program will be cooperative and that he/she will not introduce his/her name or something similar when an integer value is requested. A little ahead, when we see the string stream class we will see a possible solution for the errors that can be caused by this type of user input.

You can also use cin to request more than one datum input from the user:

```
cin >> a >> b;
```

Is equivalent to:

```
cin >> a;
cin >> b;
```

```
Q) write a program to calculate the area of circle
#include <iostream.h>
#include <conio.h>
#define pi 3.1415
int main()
{
 float r,area;
 clrscr();
 cout << "radius=";
 cin >> r;
 area=r*r*pi;
 cout << "Area of circle="<<area;
 getch();
}
```

Q) exchange two numbers

```
#include <iostream.h>
#include <conio.h>
int main()
{
 int x, y,z;
 clrscr();
 cout <<"x=";
 cin >>x;
 cout <<"y=";
 cin >>y;
 z=x;
 x=y;
 y=z;
 cout <<"New x="<<x;
 cout <<"\nNew y="<<y;
 getch();
}
```