Input and Output in C++

June 2017

Pre-requisites

- Variables in C++
- Primitive Data types
- Arrays and basic loops

1 Introduction

The TV is programmed, so is the washing machine. The microwave is programmed, and our brains too! But how do we know that these things are programmed in some manner? The answer is quite simple, we see that all these machines take some **input**, do some process and produce some **output**. That is the basic structure of any program we execute. However, how we do make this possible? We need some construct that does this interaction(input and output) for us. Well, don't worry, our programming language comes to our aid. The language we use has features that enable us to take input and produce output. Input can be taken by the user(typed from the keyboard) or it can be read from a file. Similarly, output can be printed to the monitor or can be written to a file. Now, let us see how this happens in our language of concern - C++.

2 Streams

As discussed, there can be different sources of input and output. However, the process is the same. While taking input, you just need to read the characters that appear until a *delimiter* is reached. But what is a delimiter? It is nothing but a character that marks the end of input, e.g. it can be a space. Whenever, a space is encountered the input ends. Consider the following line: "Good to see you learn input output"? If I say that the delimiter is the space character, how many different inputs are there? We have 7 - "Good", "to", "see", "you", "learn", "input", "output". But if I say that the delimiter is the newline character(when you press enter), then there is only one input - "Good to see you learn input output".

So coming back to the input process - you just need to read character by character until the delimiter is reached, whether the input is from a file or provided by the user. This is done in C++ using something called **streams**. Streams are nothing but an interface where the data is read/written onto in a sequential manner regardless of their source. So, what C++ does is, it takes the input from any source, creates a stream out of it. And then you can read the input. This is done by the header file - **iostream** - which stands for input output stream. So you need to include this header file before doing any sort of input or output. Now, let us take a look at the operators that allow input and output.

3 Input using cin

cin is an object that is used to take input. It stands for console input. It is delimeted by a white space character(space, tab or new line character) i.e a white space character marks the end of a input. cin requires

a variable where the data can be stored. The proper syntax is

```
#include <iostream>

using namespace std;

int main()
{
        int x;  //the variable where we will store our data.
        cin >> x;
        return 0;
}
```

In the above program, we first include the header file **iostream** to enable us to perform input and output. In our main function, we declare an integer type variable so that it can store integer type data. Next, we take input using the statement:
cin >> x;
Note that, whenever you need to take input you need to use ">>" operator. This program will take one integer as an input. That means when the user types an integer from the keyboard and presses enter(pressing enter is equivalent to a white space/newline character), that data will be stored in the variable x.

This was easy right, there is simply one command to accept input! All you need to do is declare the variable where you need to store the input and then use the cin object to accept input. You can use multiple ">>" operators to accept multiple input.

Consider the following program:

```
#include <iostream>

using namespace std;

int main()
{
        int x, y, z;     //the variable where we will store our data.
        cin >> x >> y >> z;
        return 0;
}
```

In the above program we accept three integers - the integers can be space separated, in different lines or even tab separated.
Example of space separated integers input:
1 2 3
Example of integers in different lines input:
1
2
3
Example of tab separated integers input:
1 2 3
The above code will work if the input is formatted in any of these ways.

Now let us see what we do when we need to take n inputs from the user.

Consider the following input format:

The first line will contain a single integer n.
The second line contains n space separated integers.

How do we store n inputs together? We use an array! Take a look at the following program :

```
#include <iostream>

using namespace std;

int main()
{
        int n;
        cin >> n;
        int array[n];    //the variable where we will store our data.
        for(int i = 0; i < n; i++)
        {
                cin >> array[i];
        }
        return 0;
}
```

The above program first takes an integer input n. Then it creates an array of size n. Then it takes n inputs from the user and stores them in the array declared.

For example, if the input is the following:

6
1 3 4 20 2 9

Then $n = 6$ and the array becomes $= \{1, 3, 4, 20, 2, 9\}$

Easy enough? Cool! This was when the delimeter could be any white space character. However, what if you need to take in spaces as input as well?

4 Input using getline

C++ also allows you to delimit using any character of your choice. This is achieved using the function **getline()**. We need to pass three parameters to this function where the third parameter is optional. The first paramter is the stream object we need to use. Since we mostly are concerned with cin stream, we take the first parameter as cin. The second parameter is the string where you need to store the input. Notice that since we are making the delimiter of our choice, we can only store the data in string format. Later, we can convert the data according to our needs but we will initially store it as a string. The third optional parameter is the delimiter. We can specify the character by which we want to delimit the input. If we do not supply any third parameter, the new line character is taken as the delimiter by default. Let us move onto an example:

```
#include <iostream>

using namespace std;

int main()
{
        string line;
        getline(cin, line);
        //We do not give any delimiter, so the delimiter is the new line character.
        return 0;
}
```

In the above program we are taking a string with spaces as an input. So, if the user provides the input "My name is Arjun." The variable **line** will have the value "My name is Arjun.". Notice that this is not possible using *cin*.

Exercise:
Can you tell what would string line store if instead of writing:
getline(cin, line);
we wrote:
cin>>line;
Hint: What is the default delimiter when we use cin? That character will mark the end of input.
Answer: The default delimiter is white space character when we use cin. The first occurence of a white space character is after the word 'My'. This would mark the end of the input and string line will end up storing only "My" instead of "My name is Arjun."

Now I think we are comfortable with taking input in C++. Let us now move onto output.

5 Output using cout

Just like *cin*, **cout** is a also a stream object. It is used to write onto the output stream. The proper syntax for its use is:

```
#include <iostream>

using namespace std;

int main()
{
        int x;
        cin >> x;
        cout << x;
        return 0;
}
```

The above program first takes an integer input and stores it in the variable x. Next, it prints the value of x on the output screen using the statment:

cout << x;

This statement simply uses << operator to print the value onto the screen. The value we provide can be a variable or a constant. And just like *cin*, we can use multiple << for multiple outputs.
Now consider the following program.

```
#include <iostream>

using namespace std;

int main()
{
        int x;
        cin >> x;
        cout << "The answer is " << x << "\n";
        return 0;
}
```

In the above program we used three << operators to print three different outputs. First the output is the string constant : "The answer is ", the second output is the integer x and the third output is the string constant : "\n" (the new line character).

Pretty simple, right? That is all we need to learn about input and output.