Variables and Literals in C++

Till now, you have seen how to print something on the screen and the basics of writing code in C++. Now let's move forward.

Before introducing variables, look at the following example.

```
#include <iostream>
int main()
{
        int n;     // declaring integer variable n
        n = 4;     // assigning 4 to n
        std::cout << n;  //printing the value of n
        return 0;
}
```

Output

In this example, **n** is a **variable**. Variables are used to store certain values. In our case, we stored a value of 4 to the variable **n**. Let's understand the above example.

int n; - This statement declares that variable 'n' can store some integer value. Whenever a variable is declared, it occupies some space in memory.
n = 4; - This assigns a value 4 to the variable 'n'.
std::cout << n; - This statement prints the value of the variable 'n' and hence 4 gets printed.

Note that 'n' is **not** written inside **" "**. 'n' written inside **" "** would have printed simple **n** instead of the value of 'n'. Let's see an example:

```
#include <iostream>
int main()
{
        int n;     // declaring integer variable n
        n = 4;     // assigning 4 to n
        std::cout << "n is " <<  n;  //printing the value of n
        return 0;
}
```

Output

As you have seen, 'n' inside **" "** is simple alphabetical character but the later 'n' is the one used as variable.

Initialization

We can also assign the value to a variable at the time of its declaration. Such assignment is known as **initialization**. Thus, we can assign a value **4** to a variable **n** at the time of its declaration as follows.

```
int n = 4;
```

Here, we initialized a variable 'n' with a value 4.

> An uninitialized variable takes some garbage value.

An **uninitialized** variable is the one to which we have not assigned any value. Unlike some programming languages where a variable which has not been assigned any value takes a value zero, an uninitialized variable in C++ by default takes some garbage value.

Data Types

Variables can be of different types depending on the type of data it can store. In the last example, the variable 'n' was declared 'int' and thus can store an integer value. Therefore, it was of type **int**. Similarly, a variable which stores a character value is of type **char** and so on.

We specify the type of a variable at the time of declaration. For example, a character variable is declared as shown below.

```
char ch;
```

In the above declaration, **ch** is the name of the variable which is of type **char** i.e., it can store character values. Let's see an example to print a character value.

```
#include <iostream>
int main()
{
        char ch;        // declaring character variable
        ch = 'b';       // assigning value 'b' to variable
        std::cout << ch;
        return 0;
}
```

Output

Here **ch** is the name of a variable of type **char** which is given a character value **b**. Note that the character value is assigned within **' '**. Characters are written inside **' '**.

Similarly, there are other data types like float (number having decimal), double (number having decimal) and boolean (true or false). These are listed in the following table along with their sizes.

Type	Keyword	Size (bytes)	Range of Value
Integer	int	4	-2,147,483,648 to 2,147,483,647

Floating point	float	4	1.8E-38 to 3.4E+38
Double floating point	double	8	2.2E-308 to 1.8E+308
Character	char	1	–128 to 127
Boolean	bool	1	false or true

The sizes of variables might change depending on the compiler and the computer you are using.

Thus, we can store any type of data in a variable by declaring its datatype. Let's see an example.

```
#include <iostream>
int main()
{
        using namespace std;
        int i;       //declaring integer variable
        float f;     //declaring float variable
        double d;    //declaring double variable
        char c;      //declaring character variable
        bool b, bl;      //decaring boolean variable;

        //assigning values to these variables

        i = 45;
        f = 34.234;
        d = 34.43242343;
        c = 'g';
        b = true;
        bl = 5 < 4;

        cout << "int : " << i << endl;
        cout << "float : " << f << endl;
        cout << "double : " << d << endl;
        cout << "char : " << c << endl;
        cout << "b : " << b << endl;
        cout << "bl : " << bl << endl;
        return 0;
}
```

Output

Here we declared variables of type int, float, double, char and boolean and stored respective values in these. One thing to note here is that **boolean** gives **1** when **true** and **0** when **false.** We declared two boolean variables (b and bl). Since we assigned true to b, therefore it printed 1 and since the expression which we assigned to bl (5 < 4) is false, hence it printed 0.

Boolean variable returns 1 if the value assigned to it is true, otherwise 0.

Taking input for different data-types

Now let's see an example to input a character in C++.

```
#include <iostream>
int main()
{
        char ch;
        std::cin >> ch;
        std::cout << ch;
        return 0;
}
```

Output

This is the same as what we did in the case of integers, with the only difference in the datatype. Similarly, we can input all types of values from the user like float, double etc.

How is a character value stored?

Whenever a character value is given to a variable of type char, its **ASCII value** (an integer value) gets stored (and not the character value).

We use **int(ch)** to print the **integer value (ASCII value)** of any character 'ch'.

You can download the full ASCII from here.

Let's see an example to print the ASCII value of a character.

```
#include <iostream>
int main()
{
        char ch;
        std::cin >> ch;
        std::cout << int(ch);
        return 0;
}
```

Output

When we gave the value 'A' to the variable 'ch', the ASCII code (65) of 'A' gets stored in ch.
So, **int(ch)** displays its **integer value** i.e. ASCII value.

Type Casting

Suppose, you have a variable whose value is 23.2332 but at some line of your code you want to use its integer value only i.e. 23. The simplest solution is **type casting.**

Type Casting is the conversion of a variable from one data type to another data type. For example, if we want to convert a char value to an int value.

Type Conversions are of two types - implicit and explicit.

Implicit Conversion

Suppose, we are adding two numbers. The first number is of type int and the second number is of type float. Since we cannot add an int and a float, so both the numbers have to be of the same data type i.e., either both are int or both are float. Since float is a larger data type than int, therefore int variable gets converted into float automatically and then both the float variables add up. This automatic conversion is called implicit conversion.

> All the character variables get converted to integers while performing arithmetic operations or in any other such expression.

```
#include <iostream>
int main(){
        int a = 12;
        char ch = 'h';

        //will add ASCII value of ch
        int sum = a + ch;
        std::cout << "sum = " << sum << std::endl;
        return 0;
}
```

Output

In the above code, when 'a' and 'ch' were added, the integer value of 'ch' (ASCII value) i.e. 104 gets added to the integer value of 'a' to produce a sum of 116.

Explicit Conversion

We can also manually convert values from one data type to another as follows:

(data-type) expression ;

Consider an example.

In this example, since x is declared of type float, therefore we are converting **float** to type **int** by writing **(int)x.**

```
#include <iostream>
int main(){
        float x = 2.45;
        std::cout << (int)x << std::endl;
}
```

Output

This is the same thing we did to get the ASCII value of a character. We just converted it into **int**.

l-values and r-values

We know that when we declare a variable, it is given a memory location and its value is stored in that location. Each memory location has an address which becomes the address of the variable which occupies that memory location.

Now let's come to l-value and r-value. An **l-value** is a value which has an address. Thus all variables are l-values since variables have addresses. Since the name l-value stands for left-value, so these are always on the left side in an assignment statement (a statement in which we assign the value from the right side to the left side). Therefore whenever a variable is assigned some value, it is always written on the left side of the assignment operator (=).

In C++, the value on the right side of **=** is assigned to the left side. E.g.- x = 8 will make x equal to 8 but 8 = x will give you error because you can't assign any value to 8, which is not a variable.

An **r-value** is a value which is assigned to an l-value. For example, in the statement **x = 8;**, x is a l-value since it is a variable to which some value is being assigned and 8 is an r-value since it is being assigned to x. Note that when we write **5 = 8;**, there will be a compilation error because 5 is not an l-value (since it is not a variable having any memory space). Let's look at the following example.

```
#include <iostream>
int main()
{
        int x, y;
        x = 4;
        y = x + 2;
        std::cout << "x = " << x << ", y = " << y;
        return 0;
}
```

Output

In the statement **x = 4;**, 4 is an r-value since it is on the right-hand side and is being assigned to 'x' which is an l-value. Similarly, in the statement **y = x + 2;**, 'y' is an l-value since it is on the left side and has a memory space. The expression 'x + 2' is an r-value since its value (4 + 2 = 6) is being assigned to the variable 'y'.

Note that to print the value of 'x', 'x' is written without **" "**.

Literals

Literals are constants which have the same value which cannot be changed in the whole program. For example in **x = 5**, 'x' is a variable and 5 is a literal (constant) and thus the value of 'x' can be changed but not of 5.

Commonly used literals are Integer Numerals, Floating-Point Numerals, Characters, Strings, Boolean and user-defined literals. Let's have a brief look at them one by one.

Integer Literals

These are constants which are integer values. For example, 45, -256 etc.

Floating-Point Literals

These constants are numbers with decimal. These are of type **float** or **double**. Numbers like 23.434, 1.2e-234 and 4.57e152 are examples of floating-point literals.

Characters and Strings

Characters are enclosed within **' '**. These can be letters (like 'a','s','z', etc), escape sequence (like \t) or some universal character.

There are some special **sequence characters** like '\t', '\n' which have some specific meaning. For example,
'\n' is used for changing line,
'\t' is used to give a tab space horizontally,
'\v' is used to give one tab space vertically,
'\"', **'\'"'**, '\\' and **'\?'** are used to print ', ", \ and ? respectively.

Strings are the collection of characters. In simple English, it is a letter, word, sentence or collection of sentences. You will go through a whole topic on string.

Strings are constants which are enclosed within **" "** like "Hello World", "C++".

So, 'a' is a character but "a" is a string.

In the above example, the **string** written within **" "** got printed as it is. **' \t '** gave a one tab space between "Hello" and "World" and '\n' changed the line. '\'', '\\', '\"' printed ', \ and " respectively.

```
#include <iostream>
int main()
{
        std::cout << "Hello" << '\t' << "World" << '\n' << '\\' << '\'' << '\"';
        return 0;
}
```

Output

Boolean

We have two Boolean literals which are **true** and **false.** These Boolean literals are also keywords in C++.