C++ VARIABLE TYPES

A variable provides us with named storage that our programs can manipulate. Each variable in C++ has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C++ is case-sensitive:

There are following basic types of variable in C++ as explained in last chapter:

Type	Description
bool	Stores either value true or false.
char	Typically a single octet*onebyte*. This is an integer type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.
wchar_t	A wide character type.

C++ also allows to define various other types of variables, which we will cover in subsequent chapters like **Enumeration, Pointer, Array, Reference, Data structures,** and **Classes**.

Following section will cover how to define, declare and use various types of variables.

Variable Definition in C++:

A variable definition means to tell the compiler where and how much to create the storage for the variable. A variable definition specifies a data type, and contains a list of one or more variables of that type as follows:

```
type variable_list;
```

Here, **type** must be a valid C++ data type including char, w_char, int, float, double, bool or any user-defined object, etc., and **variable_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here:

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

The line **int i, j, k;** both declares and defines the variables i, j and k; which instructs the compiler to create variables named i, j and k of type int.

Variables can be initialized *assignedaninitialvalue* in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

```
type variable_name = value;
```

Some examples are:

```
extern int d = 3, f = 5;    // declaration of d and f.
int d = 3, f = 5;           // definition and initializing d and f.
byte z = 22;                // definition and initializes z.
char x = 'x';               // the variable x has the value 'x'.
```

For definition without an initializer: variables with static storage duration are implicitly initialized with NULL *allbyteshavethevalue*0; the initial value of all other variables is undefined.

Variable Declaration in C++:

A variable declaration provides assurance to the compiler that there is one variable existing with the given type and name so that compiler proceed for further compilation without needing complete detail about the variable. A variable declaration has its meaning at the time of compilation only, compiler needs actual variable declaration at the time of linking of the program.

A variable declaration is useful when you are using multiple files and you define your variable in one of the files which will be available at the time of linking of the program. You will use **extern** keyword to declare a variable at any place. Though you can declare a variable multiple times in your C++ program, but it can be defined only once in a file, a function or a block of code.

Example

Try the following example where a variable has been declared at the top, but it has been defined inside the main function:

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

// Variable declaration:
extern int a, b;
extern int c;
extern float f;

int main ()
{
  // Variable definition:
  int a, b;
  int c;
  float f;

  // actual initialization
  a = 10;
  b = 20;
  c = a + b;

  cout << c << endl ;

  f = 70.0/3.0;
  cout << f << endl ;

  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
30
23.3333
```

Same concept applies on function declaration where you provide a function name at the time of its declaration and its actual definition can be given anywhere else. For example:

```
// function declaration
int func();
```

```
int main()
{
    // function call
    int i = func();
}

// function definition
int func()
{
    return 0;
}
```

Lvalues and Rvalues:

There are two kinds of expressions in C++:

- **lvalue :** Expressions that refer to a memory location is called "lvalue" expression. An lvalue may appear as either the left-hand or right-hand side of an assignment.
- **rvalue :** The term rvalue refers to a data value that is stored at some address in memory. An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right- but not left-hand side of an assignment.

Variables are lvalues and so may appear on the left-hand side of an assignment. Numeric literals are rvalues and so may not be assigned and can not appear on the left-hand side. Following is a valid statement:

```
int g = 20;
```

But following is not a valid statement and would generate compile-time error:

```
10 = 20;
```