2 Lesson 2: Variable, Constants, and Data types in C++

2.1 Variables

A variable is a storage location in memory that is stored by its value. A variable is identified or denoted by a variable name. The variable name is a sequence of one or more letters, digits or underscore.

Rules for defining variable name:

- A variable name can have one or more letters or digits or underscore for example character
- White space, punctuation symbols or other characters are not permitted to denote variable name.
- A variable name must begin with a letter.
- Variable names cannot be keywords or any reserved words of the C++ programming language.
- C++ is a case-sensitive language. Variable names written in capital letters differ from variable names with the same name but written in small letters.

2.2 Data Types

Below is a list of the most commonly used *Data Types* in C++ programming language

short int	short integer.
int	integer.
long int	long integer.
float	floating point
double	double precision floating point number.
long double	double precision floating point number.
char	single character.
bool	boolean value. It can take one of two values True or False

In order for a variable to be used in C++ programming language, the variable must first be declared. The syntax for declaring variable names is

```
Data_type variable_name;
```

The *data_type* can be int or float or any of the data types listed above. A variable name is given based on the rules for defining variable name (refer above rules).

Example:

```
int a;
```

This declares a variable name a of type int.

If there exists more than one variable of the same type, such variables can be represented by separating variable names using comma.

```
int x,y,z ;
```

This declares 3 variables x, y and z all of data type int.

The data type using integers (int, short int, long int) are further assigned a value of **signed** or **unsigned**. Signed integers signify positive and negative number value. Unsigned integers signify only positive numbers or zero.

For example, it is declared as **`unsigned`** `short int a;`

or **`signed`** `int z;`

By default, unspecified integers signify a signed integer.

For example:

```
int a;  // is declared a signed integer
```

It is possible to initialize values to variables:

```
data_type variable_name = value;
```

Example:

```
int a=0;
int b=5;
```

Code 2.1: Variable Declaration

2.3 Constants

Constants have fixed value. Constants, like variables, contain data type. Integer constants are represented as decimal notation, octal notation, and hexadecimal notation. Decimal notation is represented with a number. Octal notation is represented with the number preceded by a zero character. A hexadecimal number is preceded with the characters 0x.

Note:

- Constants are also called literals.
- Constants can be any of the data types.
- It is considered best practice to define constants using only upper-case names.

There are two other different ways to define constants in C++. These are:

- By using `const` keyword
- By using `#define` preprocessor - #define pi 3.142

Code 2.2: Constants

```
#include <iostream>
using namespace std;
#define PI 3.14
int main()
{
  const float pi = 3.142;
  float area1, area2, radius;
  radius=5;
  area1 = pi*radius*radius;
  area2 = PI*radius*radius;
  cout<<"Radius = " << radius <<", Area of the circle= " << area1 << endl;
  cout<<"Radius = " << radius <<", Area of the circle= " << area2 << endl;
  return 0;
}
```

2.4 Referencing variables

The **&** operator is used to reference an object. When using this operator on an object, you are provided with a pointer to that object. This new pointer can be used as a parameter or be assigned to a variable.

Think of a variable name as a label attached to the variable's location in memory. You can then think of a reference as a second label attached to that memory location. Therefore, you can access the contents of the variable through either the original variable name or the reference. For example, suppose we have the following example −

```
int i = 9;
```

We can declare a reference variable for i as follows.

```
int& m = i;
```

Read the **&** in these declarations as reference. Thus, read the first declaration as "m is an integer reference initialized to i".

Code 2.3: Referencing Variables

```
#include <iostream>
using namespace std;
int main()
{
      int i;  // declare a simple variable
      int& m = i; // declare a reference variable
      i = 9;
      cout << "Value of i : " << i << endl;
      cout << "Value at m : " << m << endl;
      return 0;
}
```

2.5 Pointers

A pointer is a variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is −

```
type *var-name;
```

To use a pointer, we begin by defining a pointer variable. Then we assign the address of a variable to a pointer. And finally access the value at the address available in the pointer variable using the unary operator *. If the unary operator * is omitted, what is returned is the address of the variable that the pointer is pointing to.

2.6 Differences between References and Pointers

References are often confused with pointers but there are three major differences between references and pointers;

i. You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.

ii. Once a reference is initialized to an object, it cannot be changed to refer to another object. Pointers can be pointed to another object at any time.

iii. A reference must be initialized when it is created. Pointers can be initialized at any time.

Code 2.4: Pointers and References

```
#include <iostream>
using namespace std;
int main ()
{
      int  x = 20;//  variable x declaration.
      int  *ptr;  // pointer variable ptr
      ptr = &x;   // store address of variable x in pointer variable ptr
      int & r = x;//reference variable r initialized to the address of x
      r = 30;

      cout << "Value of variable x is : " << x << endl;
      cout << "Address stored in ptr variable is : " << ptr << endl;
      cout << "Value at the address pointed by ptr is: " << *ptr << endl;
      cout << "Value at the address referenced by r is: " << r << endl;
      cout << "Value of x " << x << endl;
      return 0;
}
```