C# - VARIABLES

A variable is nothing but a name given to a storage area 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 basic value types provided in C# can be categorized as:

Type	Example
Integral types	sbyte, byte, short, ushort, int, uint, long, ulong, and char
Floating point types	float and double
Decimal types	decimal
Boolean types	true or false values, as assigned
Nullable types	Nullable data types

C# also allows defining other value types of variable such as **enum** and reference types of variables such as **class**, which we will cover in subsequent chapters.

Defining Variables

Syntax for variable definition in C# is:

```
<data_type> <variable_list>;
```

Here, data_type must be a valid C# data type including char, int, float, double, or any user-defined data type, and variable_list may consist of one or more identifier names separated by commas.

Some valid variable definitions are shown here:

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

You can initialize a variable at the time of definition as:

```
int i = 100;
```

Initializing Variables

Variables are initialized *assignedavalue* with an equal sign followed by a constant expression. The general form of initialization is:

```
variable_name = value;
```

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

```
<data_type> <variable_name> = value;
```

Some examples are:

```
int d = 3, f = 5;    /* initializing d and f. */
byte z = 22;         /* initializes z. */
double pi = 3.14159; /* declares an approximation of pi. */
char x = 'x';        /* the variable x has the value 'x'. */
```

It is a good programming practice to initialize variables properly, otherwise sometimes program may produce unexpected result.

The following example uses various types of variables:

```
using System;
namespace VariableDefinition
{
   class Program
   {
      static void Main(string[] args)
      {
         short a;
         int b ;
         double c;

         /* actual initialization */
         a = 10;
         b = 20;
         c = a + b;
         Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c);
         Console.ReadLine();
      }
   }
}
```

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

```
a = 10, b = 20, c = 30
```

Accepting Values from User

The **Console** class in the **System** namespace provides a function **ReadLine** for accepting input from the user and store it into a variable.

For example,

```
int num;
num = Convert.ToInt32(Console.ReadLine());
```

The function **Convert.ToInt32** converts the data entered by the user to int data type, because **Console.ReadLine** accepts the data in string format.

Lvalue and Rvalue Expressions in C#:

There are two kinds of expressions in C#:

- **lvalue:** An expression that is an lvalue may appear as either the left-hand or right-hand side of an assignment.
- **rvalue:** An expression that is an rvalue may appear on the right- but not left-hand side of an assignment.

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

```
int g = 20;
```

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

```
10 = 20;
```