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.

Why Do We Use Variables in C?

A variable in C is a user-assigned name to a certain location in the computer's memory, which is a collection of a large number of randomly accessible locations capable of holding a single bit. Each location in the memory is identified by a unique address, expressed in binary (or Hexa-decimal for convenience) format.

Since it is extremely cumbersome to store and process the data in the memory by referring to their locations in binary form, high-level languages such as C let the locations be identified by user-defined names or variables.

Instead of identifying a free memory location and assigning it a value, you can find a suitable mnemonic identifier and assign it a value. The C compiler will choose an appropriate location and bind it to the identifier specified by you.

Naming Conventions of C Variables

The name of the variable must start with an alphabet (upper or lowercase) or an underscore (_). It may consist of alphabets (upper or lowercase), digits, and underscore characters. No other characters can be a part of the name of a variable in C.

Variable names in C are case-sensitive. For example, "age" is not the same as "AGE".

The ANSI standard recognizes a length of 31 characters for a variable name. Although you can choose a name with more characters, only the first 31 will be recognized. Using a descriptive name for a variable, that reflects the value it intends to store is considered to be a good practice. Avoid using very short variable names that might confuse you.

C is a statically typed language. Hence, the data type of the variable must be mentioned in the declaration before its name. A variable may be declared inside a function (local variable) or globally. More than one variable of the same type may be declared in a single statement.

Example

Based on the above set of rules and conventions, here are some valid and invalid variable names:

```
int _num = 5;        // valid integer variable
float marks = 55.50; // valid float variable
char choice = '0';   // valid char variable

// invalid variable name
// cannot use "-"
int sub-1 = 35;

//invalid; must have data type
avg = 50;

// invalid; name can be used for
// declaration only once in a function
int choice = 0;

// Valid integer name
int sal_of_employee = 20000;

// Valid because all are of same type
int phy, che, maths;

// error because variables of
// different types in same statement
int sal, float tax;
```

In C, **variables** can store data belonging to any of the types it recognizes. Hence there are as many number of types of variables as the number of data types in C.

Sr.No	Type & Description
1	**char** Typically a single octet(one byte). It is an integer type.
2	**int** The most natural size of integer for the machine.
3	**float** A single-precision floating point value.
4	**double** A double-precision floating point value.
5	**void** Represents the absence of type.

C programming language also allows to define various other types of variables such as Enumeration type, Pointer type, Array type, Structure type, Union type, etc. For this chapter, let us study only basic variable types.

Variable Definition in C

A variable definition tells the compiler where and how much storage to create 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; and **variable_list** may consist of one or more identifier names separated by commas.

Some valid variable declarations are shown here −

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

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

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows −

```
type variable_name = value;
```

Example: Variable Definition and Initialization

Take a look at the following examples:

```
// declaration of d and f
extern int d = 3, f = 5;

// definition and initializing d and f
int d = 3, f = 5;
```

```
// definition and initializes z
byte z = 22;

// the variable x has the value 'x'
char x = 'x';
```

For definition without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables are undefined.

Variable Declaration in C

As per the ANSI C standard, all the variables must be declared in the beginning. Variable declaration after the first processing statement is not allowed. Although the C99 and C11 standard revisions have removed this stipulation, it is still considered a good programming practice. You can declare a variable to be assigned a value later in the code, or you can initialize it at the time of declaration.

Example: Variable Declaration

```
// declaration with initialization
int x = 10;

// declare first and assign later
int y;
y = 20;

// define and initialize two variables
int d = 3, f = 5;

// the variable x has the value 'x'
char x = 'x';
```

Once a variable of a certain type is declared, it cannot be assigned a value of any other type. In such a case the C compiler reports a type mismatch error.

A variable declaration provides assurance to the compiler that there exists a variable with the given type and name so that the compiler can proceed with further compilation without requiring complete detail about the variable. A variable definition has its meaning at the time of compilation only, the compiler needs actual variable definition at the time of linking 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 the program. You will use the keyword "extern" to declare a variable at any place. Though you can declare a variable multiple times in your C program, it can be defined only once in a file, a function, or a block of code.

Example

Try the following example, where variables have been declared at the top, but they have been defined and initialized inside the main function −

</> Open Compiler

```
#include <stdio.h>

// 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;
   printf("value of c : %d \n", c);

   f = 70.0/3.0;
   printf("value of f : %f \n", f);

   return 0;
}
```

Output

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

```
value of c : 30
value of f : 23.333334
```

The 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 in C

There are two kinds of expressions in C:

- lvalue expressions
- rvalue expressions

Lvalue Expressions in C

Expressions that refer to a memory location are called "lvalue" expressions. An lvalue may appear as either the left-hand or right-hand side of an assignment.

Variables in C are lvalues and so they may appear on the left-hand side of an assignment.

Rvalue Expressions in C

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-hand side but not on the left-hand side of an assignment.

Numeric literals are rvalues and so they may not be assigned and cannot appear on the left-hand side.

Take a look at the following valid and invalid statements:

```
// valid statement
int g = 20;

// invalid statement
// it would generate compile-time error
10 = 20;
```

Variables in C can be classified based on the following parameters:

- **Data types** – int, float, char or struct types.
- **Scope** – global or local variables.
- **Storage type** – automatic, static, register or extern.

We shall learn about local and global types and storage types later in this tutorial.