Use of the Variables in C

Variables are the storage areas in a code that the program can easily manipulate. Every variable in C language has some specific type- that determines the layout and the size of the memory of the variable, the range of values that the memory can hold, and the set of operations that one can perform on that variable.

The name of a variable can be a composition of digits, letters, and also underscore characters. The name of the character must begin with either an underscore or a letter. In the case of C, the lowercase and uppercase letters are distinct. It is because C is case-sensitive in nature. Let us look at some more ways in which we name a variable.

Rules for Naming a Variable in C

We give a variable a meaningful name when we create it. Here are the rules that we must follow when naming it:

1. The name of the variable must not begin with a digit.

2. A variable name can consist of digits, alphabets, and even special symbols such as an underscore (_).

3. A variable name must not have any keywords, for instance, float, int, etc.

4. There must be no spaces or blanks in the variable name.

5. The C language treats lowercase and uppercase very differently, as it is case sensitive. Usually, we keep the name of the variable in the lower case.

Let us look at some of the examples,

int var1; // it is correct

int 1var; // it is incorrect – the name of the variable should not start using a number

int my_var1; // it is correct

int my$var // it is incorrect – no special characters should be in the name of the variable

char else; // there must be no keywords in the name of the variable

int my var; // it is incorrect – there must be no spaces in the name of the variable

int COUNT; // it is a new variable

int Count; // it is a new variable

int count; // it is a valid variable name

Data Type of the Variable

We must assign a data type to all the variables that are present in the C language. These define the type of data that we can store in any variable. If we do not provide the variable with a data type, the C compiler will ultimately generate a syntax error or a compile-time error.

The data Types present in the C language are float, int, double, char, long int, short int, etc., along with other modifiers.

Types of Primary/ Primitive Data Types in C Language

The variables can be of the following basic types, based on the name and the type of the variable:

Type of Variable	Name	Description	Uses
char	Character	It is a type of integer. It is typically one byte (single octet).	We use them in the form of single alphabets, such as X, r, B, f, k, etc., or for the ASCII character sets.
int	Integer	It is the most natural size of an integer used in the machine.	We use this for storing the whole numbers, such as 4, 300, 8000, etc.
float	Floating-Point	It is a floating-point value that is single precision.	We use these for indicating the real number values or decimal points, such as 20.8, 18.56, etc.
double	Double	It is a floating-point value that is double precision.	These are very large-sized numeric values that aren't really allowed in any data type that is a floating-point or an integer.
void	Void	It represents that there is an absence of type.	We use it to represent the absence of value. Thus, the use of this data type is to define various functions.

Let us look at a few examples,

```
// int type variable in C
int marks = 45;
// char type variable in C
char status = 'G';
// double type variable in C
double long = 28.338612;
// float type variable in C
float percentage = 82.5;
```

If we try to assign a variable with an incorrect value of datatype, then the compiler will (most probably) generate an error- the compile-time error. Or else, the compiler will convert this value automatically into the intended datatype of the available variable.

Let us look at an example,

```
#include <stdio.h>
int main() {
// assignment of the incorrect value to the variable
int a = 20.397;
printf("Value is %d", a);
```

```
return 0;
}
```

The output generated here will be:

20

As you can already look at this output- the compiler of C will remove the part that is present after the decimal. It is because the data types are capable of storing only the whole numbers.

We Cannot Change The Data Type

Once a user defines a given variable with any data type, then they will not be able to change the data type of that given variable in any case.

Let us look at an example,

```
// the int variable in C
int marks = 20;
float marks; // it generates an error
```

Variable Definition in C

The variable definition in C language tells the compiler about how much storage it should be creating for the given variable and where it should create the storage. Basically, the variable definition helps in specifying the data type. It contains a list of one variable or multiple ones as follows:

type variable_list;

In the given syntax, *type* must be a valid data type of C that includes w_char, char, float, int, bool, double, or any object that is user-defined. The *variable_list*, on the other hand, may contain one or more names of identifiers that are separated by commas. Here we have shown some of the valid declarations:

```
char c, ch;
int p, q, r;
double d;
float f, salary;
```

Here, the line int p, q, r; defines as well as declares the variables p, q, and r. It instructs the compiler to create three variables- named p, q, and r- of the type int.

We can initialize the variables in their declaration (assigned an initial value). The initializer of a variable may contain an equal sign- that gets followed by a constant expression. It goes like this:

type variable_name = value;

A few examples are −

```
extern int p = 3, q = 5; // for the declaration of p and q.
int p = 3, q = 5; // for the definition and initialization of p and q.
byte x = 22; // for the definition and initialization of x.
char a = ‘a’; // the variable x contains the value ‘a’.
```

In case of definition without the initializer: The variables with a static duration of storage are initialized implicitly with NULL (here, all bytes have a 0 value), while the initial values of all the other variables are not defined.

Declaration of Variable in C

Declaring a variable provides the compiler with an assurance that there is a variable that exists with that very given name. This way, the compiler will get a signal to proceed with the further compilation without needing the complete details regarding that variable.

The variable definition only has a meaning of its own during the time of compilation. The compiler would require an actual variable definition during the time of program linking.

The declaration of variables is useful when we use multiple numbers of files and we define the variables in one of the files that might be available during the time of program linking. We use the *extern* keyword for declaring a variable at any given place. Though we can declare one variable various times in a C program, we can only define it once in a function, a file, or any block of code.

Example

Let us look at the given example where we have declared the variable at the top and initialized and defined them inside the main function:

```
#include <stdio.h>

// Declaration of Variable

extern int p, q;

extern int c;

extern float f;

int main () {

/* variable definition: */

int p, q;

int r;

float i;

/* actual initialization */

p = 10;

q = 20;

r = p + q;

printf("the value of r : %d \n", r);

i = 70.0/3.0;

printf("the value of i : %f \n", i);

return 0;

}
```

The compilation and execution of the code mentioned above will produce the result as follows:

```
the value of r : 30
```

the value of i : 23.333334

Classification of Variables in C

The variables can be of the following basic types, based on the name and the type of the variable:

- **Global Variable**: A variable that gets declared outside a block or a function is known as a global variable. Any function in a program is capable of changing the value of a global variable. It means that the global variable will be available to all the functions in the code. Because the global variable in c is available to all the functions, we have to declare it at the beginning of a block. Explore, Global Variable in C to know more.
  Example,
  int value=30; // a global variable
  void function1(){
  int a=20; // a local variable
  }
- **Local Variable:** A local variable is a type of variable that we declare inside a block or a function, unlike the global variable. Thus, we also have to declare a local variable in c at the beginning of a given block.
  Example,
  void function1(){
  int x=10; // a local variable
  }
  A user also has to initialize this local variable in a code before we use it in the program.
- **Automatic Variable:** Every variable that gets declared inside a block (in the C language) is by default automatic in nature. One can declare any given automatic variable explicitly using the keyword *auto*.
  Example,
  void main(){
  int a=80; // a local variable (it is also automatic variable)
  auto int b=50; // an automatic variable
  }
- **Static Variable:** The static variable in c is a variable that a user declares using the *static* keyword. This variable retains the given value between various function calls.
  Example, void function1(){

  int a=10; // A local variable

  static int b=10; // A static variable

  a=a+1;

  b=b+1;

  printf("%d,%d",a,b);

  }

  If we call this given function multiple times, then the local variable will print this very same value for every function call. For example, 11, 11, 11, and so on after this. The static variable, on the other hand, will print the value that is incremented in each and every function call. For example, 11, 12, 13, and so on.

- **External Variable:** A user will be capable of sharing a variable in multiple numbers of source files in C if they use an external variable. If we want to declare an external variable, then we need to use the keyword *extern*.
  Syntax, extern int a=10;// external variable (also a global variable)

Rvalues and Lvalues in C

There are two types of expressions in the C language:

rvalue − This term refers to the data value that gets stored at some type of memory address. The rvalue is basically an expression that has no value assigned to it. It means that an rvalue may appear on the RHS (right-hand side), but it cannot appear on the LHS (left-hand side) of any given assignment.

lvalue − lvalue expressions are the expressions that refer to any given memory location. The lvalue can appear as both- the RHS as well as the LHS of any assignment.

A variable is an lvalue. Thus, it may appear on the LHS of an assignment as well, along with the RHS. The numeric literals are rvalues. Thus, these might not be assigned. Thus, these can't appear on the LHS. Here are two statements- one valid and invalid:

Example,

int g = 10; // a valid statement

30 = 60; // an invalid statement; will generate a compile-time error

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

We will, in this chapter, look into the way each operator works.

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10

*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then −

Show Examples

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Assignment Operators

The following table lists the assignment operators supported by the C language −

Show Examples

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A

-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Misc Operators ↦ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

Show Examples

Operator	Description	Example

sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Show Examples

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right

Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right