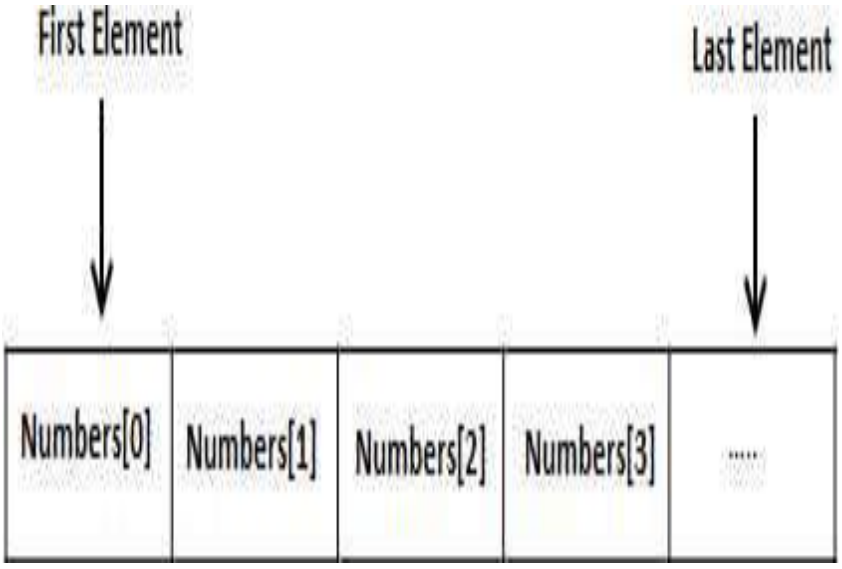
Arrays

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows -

type arrayName [arraySize];

This is called a single-dimensional array. The arraySize must be an integer constant greater than zero and type can be any valid C data type. For

example, to declare a 10-element array called balance of type double, use this statement -

double balance[10];

Here balance is a variable array which is sufficient to hold up to 10 double numbers.

Initializing Arrays:-

You can initialize an array in C either one by one or using a single statement as follows -

double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

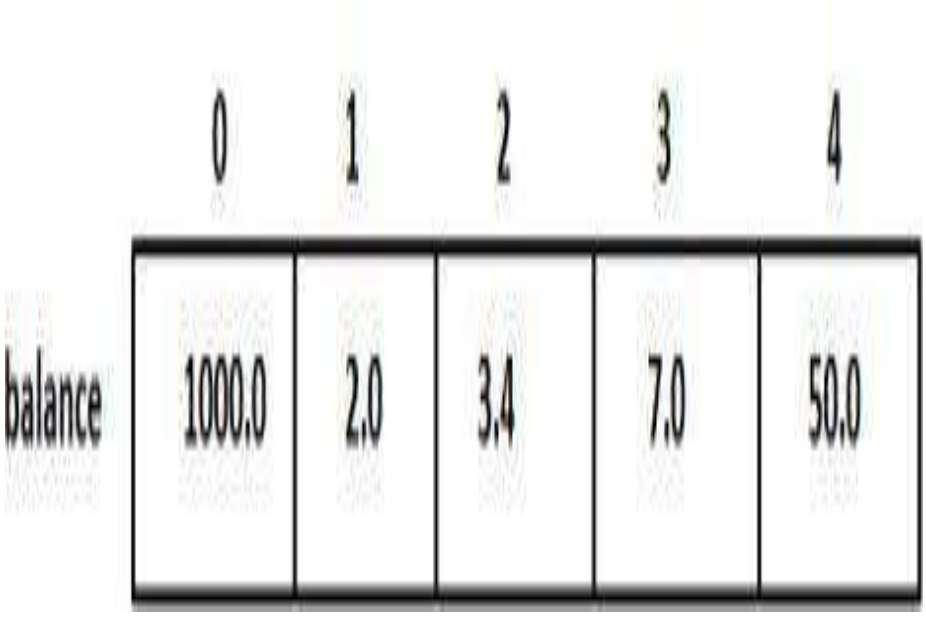
If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write -

double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array -

balance[4] = 50.0;

The above statement assigns the 5th element in the array with a value of 50.0. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be total size of the array minus 1. Shown below is the pictorial representation of the array we discussed above -



In C programming language, array can be divided into following types:

1.One Dimensional Array
2.Multi Dimensional Array

1.One Dimensional Array

An array where data is arranged in a single dimension is called one dimensional array.

Syntax:-

datatype array_name[size];

Here, array_name is an array of type datatype and size is the number of elements in that array.

Example

```
#include

int main()
{
int i,a[10];
printf("Enter 5 numbersn");
for(i=0;i<5;i++)
{
scanf("%d",&a[i]);
}
printf("Reverse ordern");
for(i=4;i>=0;i--)
{
printf("%dn",a[i]);
}
return 0;
}
```

In this program, a one dimensional integer array with 5 elements is declared where these values are entered by the user. These values are then stored in respective index of array a using a for loop. Then, these values are printed in reverse order, i.e. the value with higher index is printed at first, by traversing through the array.

Output:-

Enter 5 numbers
11
34
-21
6
90
Reverse order
90
6
-21
34
11

2. Multi-Dimensional Array:-

An array where data are arranged in multiple dimensions is called multi-dimensional array. In other words, a multi-dimensional array is an array of another array with one dimension less i.e. a two dimensional array is an array of one dimensional array, a three dimensional array is an array of two dimensional array and so on. An array can have as many dimensions as required. However, two dimensional and three dimensional array are commonly used.

Syntax

datatype array_name[d1][d2]...[dn];

Here, array_name is an array of type datatype and it has n dimensions.

The number of elements in a multi dimensional array is equal to the product of size of all dimensions i.e. total number of elements in array array_name is d1*d2* **?** dn.

a).Two Dimensional Array :-

An array where data is arranged in two dimensions is called two dimensional array.

Syntax

datatype array_name[d1][d2];

Here, array_name is an array of type datatype and it has 2 dimensions. The number of elements in array_name is equal to d1*d2.

Example:-

```
#include

int main()
{
int i,j,a[2][3],sum=0;
printf("Enter elements of matrixn");
for(i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
printf("a[%d][%d] = ",i+1,j+1);
scanf("%d",&a[i][j]);
sum = sum + a[i][j];
}
}
printf("Sum = %d",sum);
return 0;
}
```

In this program, a 2 x 3 matrix is represented by a two dimensional array a[2][3]. The elements of matrix are entered by user. A variable sum is declared and initialized to zero that holds the sum. A nested for loop is used to enter the elements of matrix and to add them with sum. Finally, the sum is printed after the loop is terminated.

Output:-

```
Enter elements of matrix
a[1][1] = 10
a[1][2] = 4
a[1][3] = 21
a[2][1] = 7
a[2][2] = 19
a[2][3] = 0
Sum = 61
```

b).Three Dimensional Array:-

An array where data is arranged in three dimensions is called three dimensional array.

Syntax

```
datatype array_name[d1][d2][d3];
```

Here, array_name is an array of type datatype and it has 3 dimensions. The number of elements in array_name is equal to d1*d2*d3.

Example:-

```
#include

int main()
{
int a[10][10][10],d1,d2,d3,i,j,k;
printf("Enter size of three dimensions: ");
scanf("%d%d%d",&d1,&d2,&d3);
printf("Enter elements of arrayn");
for(i=0;i<d1;i++)< br="" style="box-sizing: inherit;"> {
for(j=0;j<d2;j++)< br="" style="box-sizing: inherit;"> {
for(k=0;k<d3;k++)< br="" style="box-sizing: inherit;"> {
printf("a[%d][%d][%d] = ",i,j,k);
scanf("%d",&a[i][j][k]);
}
}
}
printf("Displaying elements of arrayn");
for(i=0;i<d1;i++)< br="" style="box-sizing: inherit;"> for(j=0;j<d2;j++)< br="" style="box-sizing: inherit;"> for(k=0;k<d3;k++)< br="" style="box-sizing: inherit;"> printf("a[%d][%d][%d] = %dn",i,j,k,a[i][j][k]);
getch();
return 0;
}</d3;k++)<></d2;j++)<></d1;i++)<></d3;k++)<></d2;j++)<></d1;i++)<>
```

This example show how data is stored and accessed from a three dimensional array. The values of size of three dimensions: d1, d2 and d3 are entered by user. According to these values, a nested loop is created to enter the value of elements of array and display them. The outermost loop runs d1 times, middle loop runs d2 times and the innermost loop runs d3 times printing the values of the array a as they go accessed by their index.

Output:-

```
Enter size of three dimensions: 2 2 2
Enter elements of array
a[0][0][0] = 5
a[0][0][1] = 9
a[0][1][0] = -11
a[0][1][1] = 23
a[1][0][0] = 45
a[1][0][1] = -111
a[1][1][0] = 2
a[1][1][1] = 0
```

Displaying elements of array
a[0][0][0] = 5
a[0][0][1] = 9
a[0][1][0] = -11
a[0][1][1] = 23
a[1][0][0] = 45
a[1][0][1] = -111
a[1][1][0] = 2
a[1][1][1] = 0

Data types

Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in C can be classified as follows -

1).**Basic Types** - They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.
2).**Enumerated types** - They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.
3).**The type void** - The type specifier void indicates that no value is available.
4).**Derived types** - They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

The array types and structure types are referred collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see the basic types in the following section, where as other types will be covered in the upcoming chapters.

Integer Types

The following table provides the details of standard integer types with their storage sizes and value ranges -

C Basic Data Types	32-bit CPU		64-bit CPU	
	Size (bytes)	Range	Size (bytes)	Range
char	1	-128 to 127	1	-128 to 127
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	8	-9,223,372,036,854,775,808-9,223,372,036,854,775,807
long long	8	9,223,372,036,854,775,808-9,223,372,036,854,775,807	8	9,223,372,036,854,775,808-9,223,372,036,854,775,807
float	4	3.4E +/- 38	4	3.4E +/- 38
double	8	1.7E +/- 308	8	1.7E +/- 308

To get the exact size of a type or a variable on a particular platform, you can use the sizeof operator. The expressions sizeof(type) yields the storage size of the object or type in bytes. Given below is an example to get the size of various type on a machine using different constant defined in limits.h header file -

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <float.h>

int main(int argc, char** argv) {
printf("CHAR_BIT : %d\n", CHAR_BIT);
printf("CHAR_MAX : %d\n", CHAR_MAX);
printf("CHAR_MIN : %d\n", CHAR_MIN);
printf("INT_MAX : %d\n", INT_MAX);
```

```
printf("INT_MIN : %d\n", INT_MIN);
printf("LONG_MAX : %ld\n", (long) LONG_MAX);
printf("LONG_MIN : %ld\n", (long) LONG_MIN);
printf("SCHAR_MAX : %d\n", SCHAR_MAX);
printf("SCHAR_MIN : %d\n", SCHAR_MIN);
printf("SHRT_MAX : %d\n", SHRT_MAX);
printf("SHRT_MIN : %d\n", SHRT_MIN);
printf("UCHAR_MAX : %d\n", UCHAR_MAX);
printf("UINT_MAX : %u\n", (unsigned int) UINT_MAX);
printf("ULONG_MAX : %lu\n", (unsigned long) ULONG_MAX);
printf("USHRT_MAX : %d\n", (unsigned short) USHRT_MAX);
return 0;
}
```

When you compile and execute the above program, it produces the following result on Linux -

```
CHAR_BIT : 8
CHAR_MAX : 127
CHAR_MIN : -128
INT_MAX : 2147483647
INT_MIN : -2147483648
LONG_MAX : 9223372036854775807
LONG_MIN : -9223372036854775808
SCHAR_MAX : 127
SCHAR_MIN : -128
SHRT_MAX : 32767
SHRT_MIN : -32768
UCHAR_MAX : 255
UINT_MAX : 4294967295
ULONG_MAX : 18446744073709551615
USHRT_MAX : 65535
```

Floating-Point Types

The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision -

Floating Point Data Types

Type	Size(in bytes)	Range	Digits of Precision
float	4	$3.4 \cdot 10^{-38}$ to $3.4 \cdot 10^{38}$	7
double	8	$1.7 \cdot 10^{-308}$ to $1.7 \cdot 10^{308}$	15
long double	10	$3.4 \cdot 10^{-4932}$ to $3.4 \cdot 10^{4932}$	18

The header file float.h defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs. The following example prints the storage space taken by a float type and its range values -

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <float.h>

int main(int argc, char** argv) {
printf("Storage size for float : %d \n", sizeof(float));
printf("FLT_MAX : %g\n", (float) FLT_MAX);
printf("FLT_MIN : %g\n", (float) FLT_MIN);
printf("-FLT_MAX : %g\n", (float) -FLT_MAX);
printf("-FLT_MIN : %g\n", (float) -FLT_MIN);
printf("DBL_MAX : %g\n", (double) DBL_MAX);
printf("DBL_MIN : %g\n", (double) DBL_MIN);
printf("-DBL_MAX : %g\n", (double) -DBL_MAX);
printf("Precision value: %d\n", FLT_DIG );
return 0;
}
```

When you compile and execute the above program, it produces the following result on Linux -

```
Storage size for float : 4
FLT_MAX : 3.40282e+38
FLT_MIN : 1.17549e-38
-FLT_MAX : -3.40282e+38
-FLT_MIN : -1.17549e-38
DBL_MAX : 1.79769e+308
DBL_MIN : 2.22507e-308
-DBL_MAX : -1.79769e+308
Precision value: 6
```

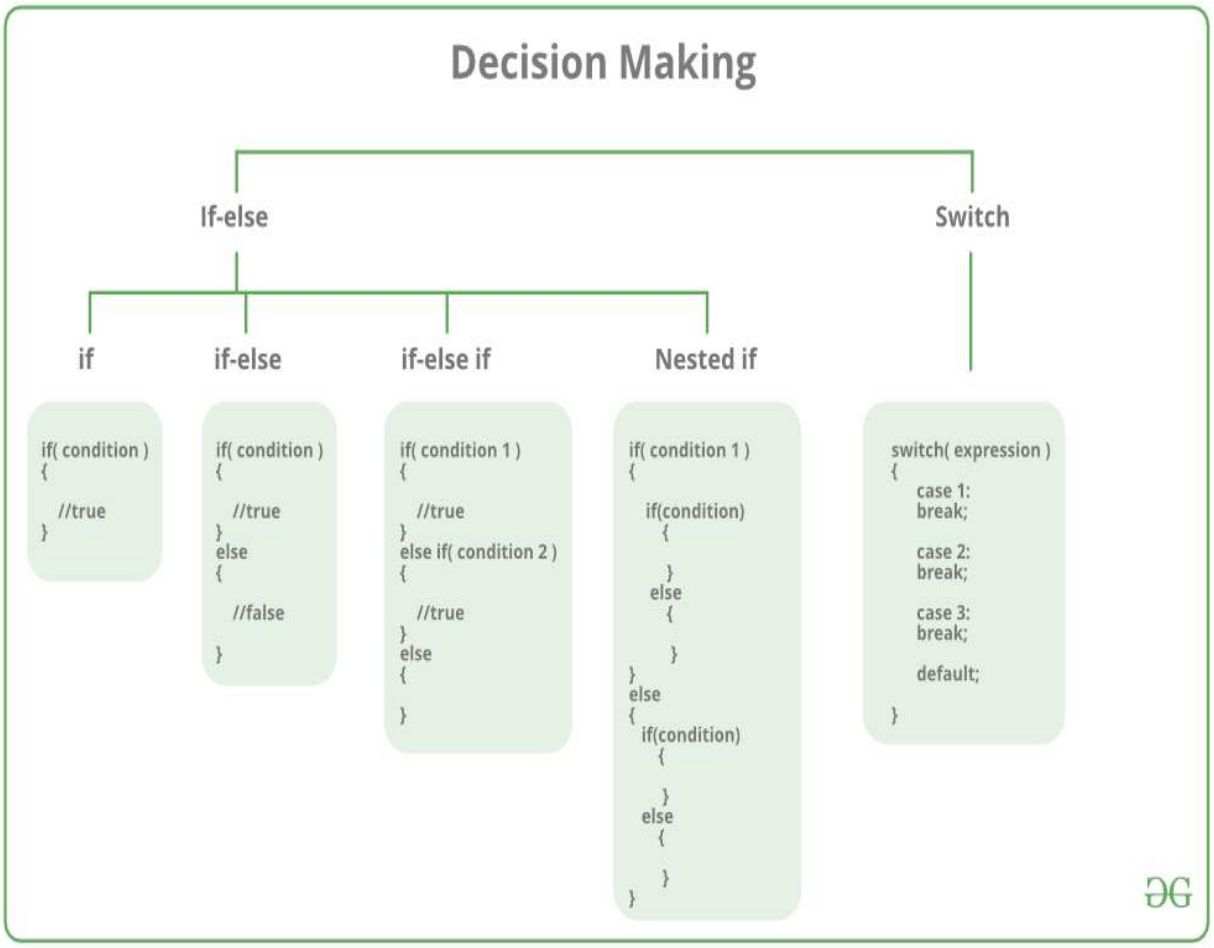
Void Types

the void type specifies that no value is available. It is used in three kinds of situations -

1).**Function returns as void** - There are various functions in C which do not return any value or you can say they return void. A function with no return value has the return type as void. For example, void exit (int status);
2).**Function arguments as void** - There are various functions in C which do not accept any parameter. A function with no parameter can accept a void. For example, int rand (void);
3).**Pointers to void** - A pointer of type void * represents the address of an object, but not its type. For example, a memory allocation function void *malloc(size_t size); returns a pointer to void which can be casted to any data type.

Decision Making

There come situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code. For example, in C if x occurs then execute y else execute z. There can also be multiple conditions like in C if x occurs then execute p, else if condition y occurs execute q, else execute r. This condition of C else-if is one of the many ways of importing multiple conditions



Decision-making statements in programming languages decide the direction of the flow of program execution. Decision-making statements available in C are:

1).if statement
2).if..else statements

3).nested if statements
4).if-else-if ladder
5).switch statements
6).Jump Statements:

- break
- continue
- goto
- return

1).if statement

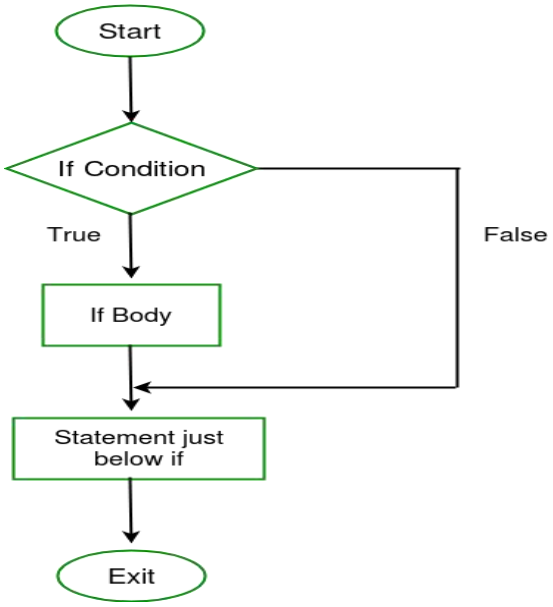
if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

Syntax:-

```
if(condition)
{
// Statements to execute if
// condition is true
}
```

Here, the condition after evaluation will be either true or false. C if statement accepts boolean values - if the value is true then it will execute the block of statements below it otherwise not. If we do not provide the curly braces '{' and '}' after if(condition) then by default if statement will consider the first immediately below statement to be inside its block.

Flowchart:-



Example:-

```
#include

int main() {
int i = 10;
if (i > 15)
{
printf("10 is less than 15");
}
printf("I am Not in if");
}
```

Output:-

I am Not in if

2).if..else statement

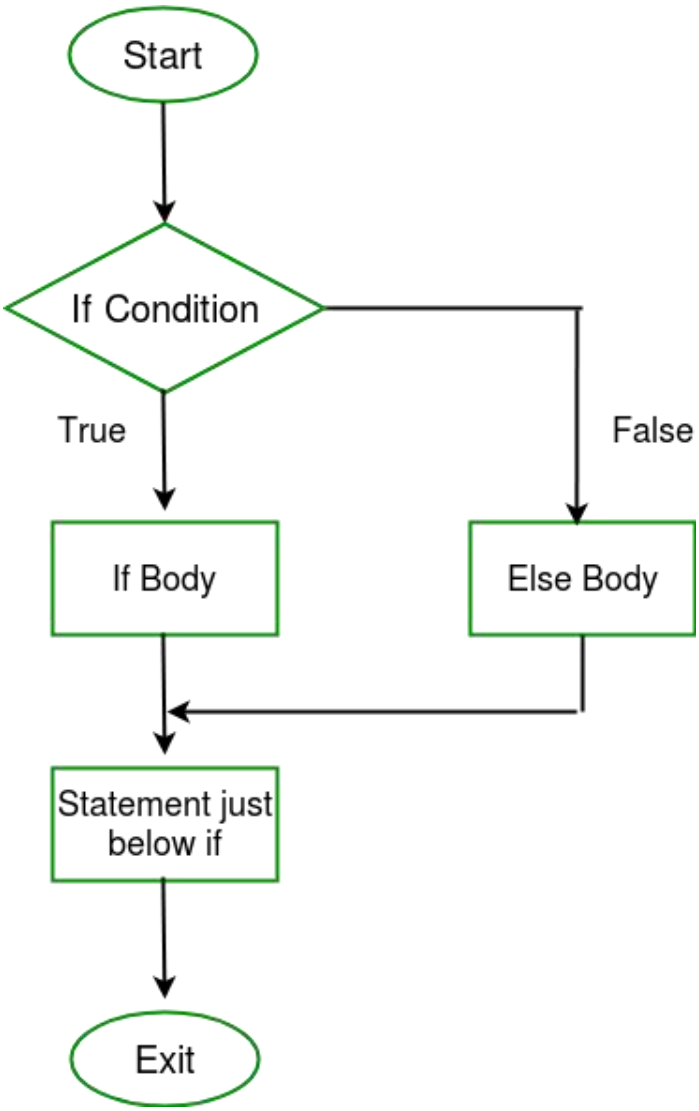
The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the C else statement. We can use the else statement with if statement to execute a block of code when the condition is false.

Syntax:-

```
if (condition)
{
// Executes this block if
```

// condition is true
}
else
{
// Executes this block if
// condition is false
}

Flowchart:-



Example:-

```
#include

int main() {
int i = 20;
if (i < 15){
```

```
printf("i is smaller than 15");
}
else{
printf("i is greater than 15");
}
return 0;
}
```

Output:-

i is greater than 15

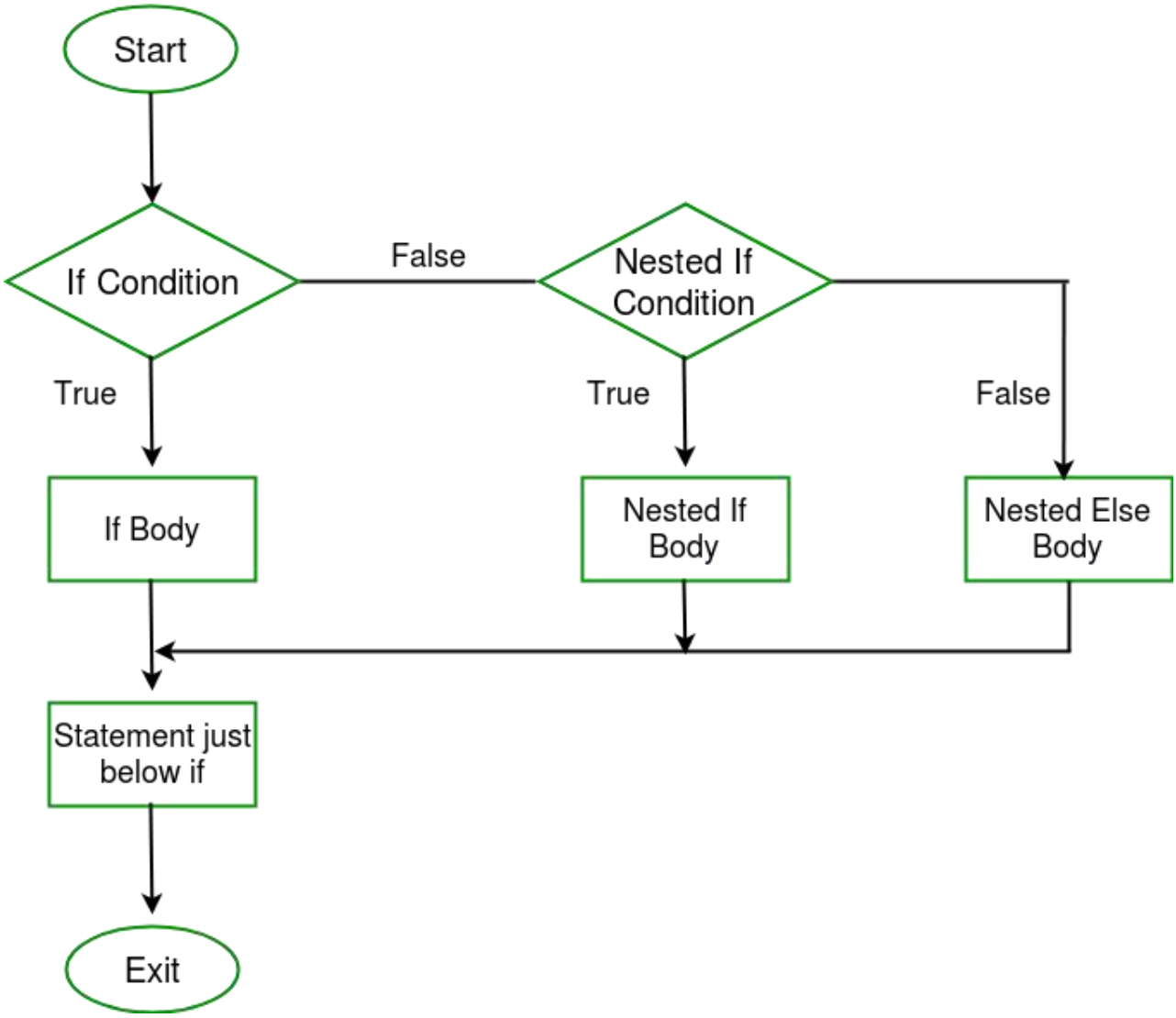
3).nested if statements:-

A nested if in C is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, both C and C++ allow us to nested if statements within if statements, i.e, we can place an if statement inside another if statement.

Syntax:-

```
if (condition1)
{
// Executes when condition1 is true
if (condition2)
{
// Executes when condition2 is true
}
}
```

Flowchart:-



Example

```
#include

int main() {
int i = 10;
if (i == 10)
{
// First if statement
if (i < 15)
printf("i is smaller than 15\n");
// Nested - if statement
// Will only be executed if statement above
// is true
if (i < 12)
printf("i is smaller than 12 too\n");
else
```

```
printf("i is greater than 15");
}
return 0;
}
```

Example

```
i is smaller than 15
i is smaller than 12 too
```

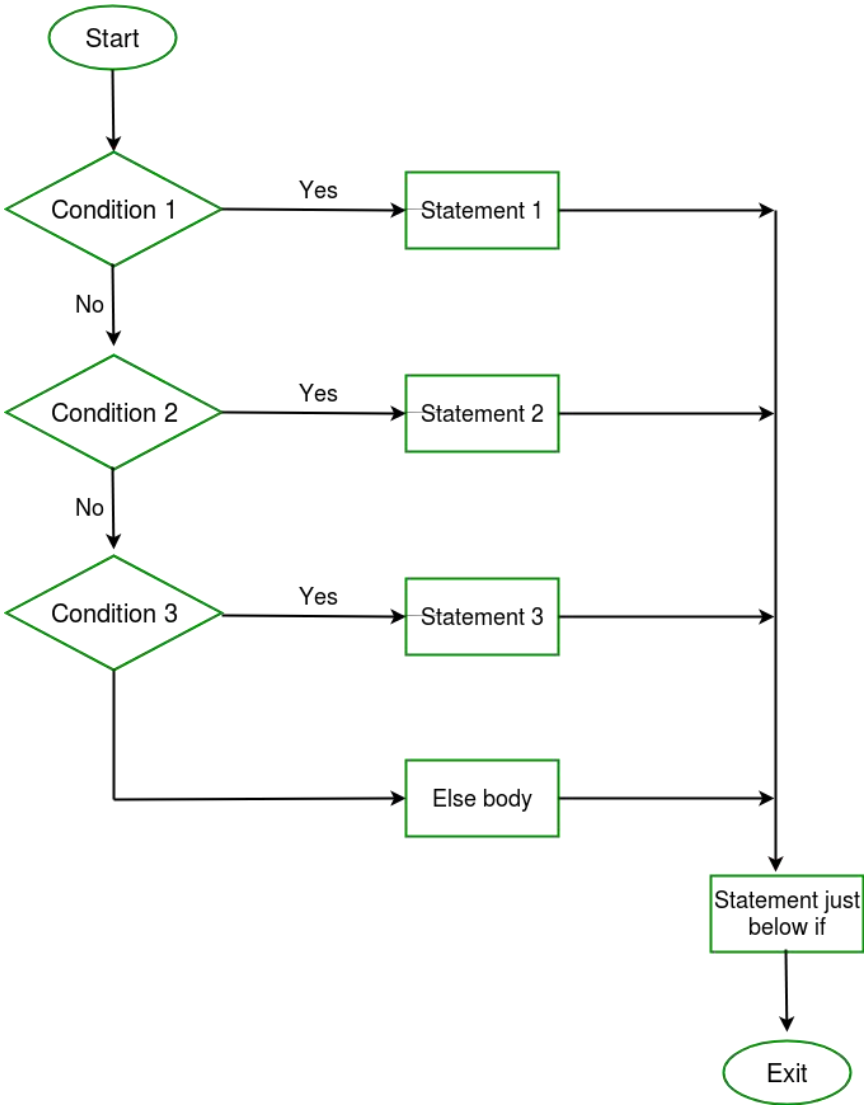
4).if-else-if ladder Statement

Here, a user can decide among multiple options. The C if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed. If none of the conditions are true, then the final else statement will be executed.

Syntax:-

```
if (condition)
statement;
else if (condition)
statement;
.
.
else
statement;
```

Flowchart:-



Example

```
#include

int main() {
int i = 20;
if (i == 10)
printf("i is 10");
else if (i == 15)
printf("i is 15");
else if (i == 20)
printf("i is 20");
else
printf("i is not present");
}
```

Output:-

i is 20

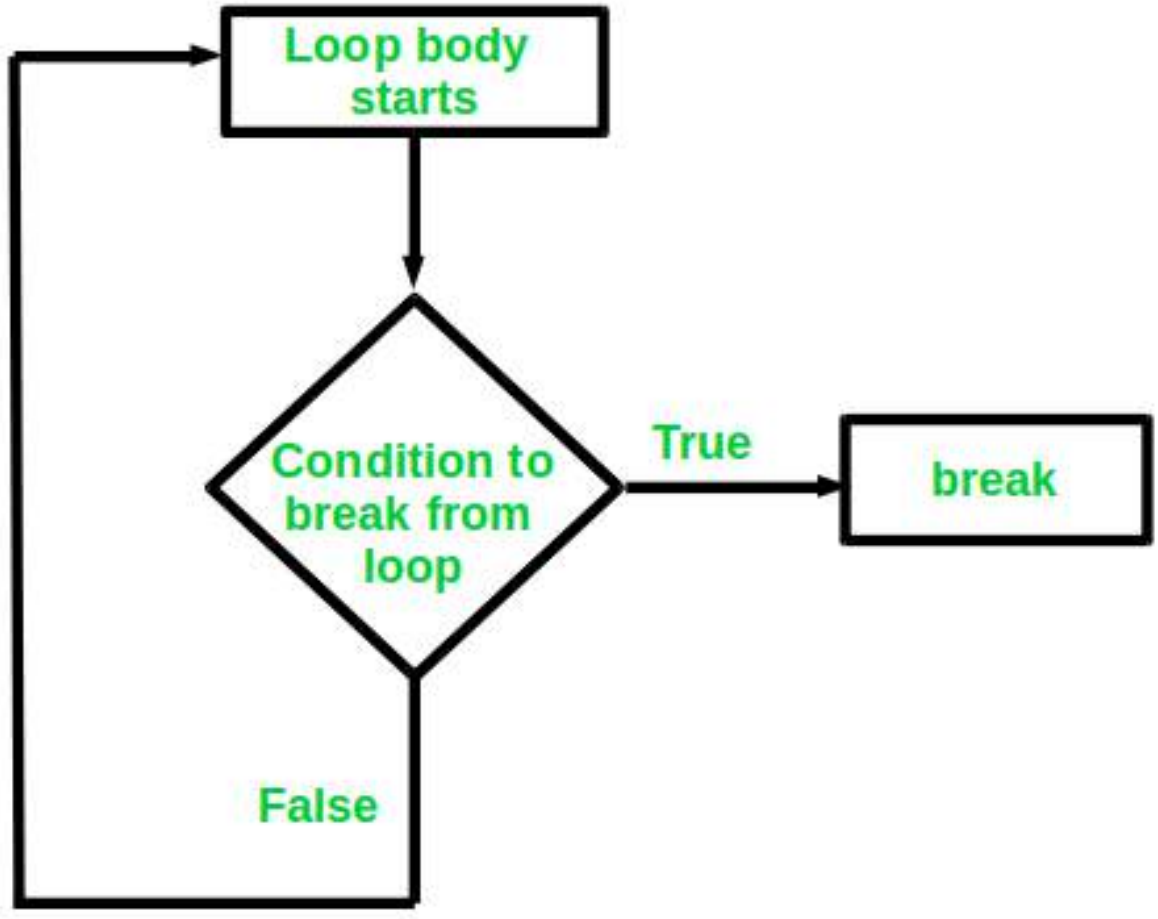
6).Jump Statements:-

These statements are used in C orC++ for the unconditional flow of control throughout the functions in a program. They support four types of jump statements:

- **Break:-**This loop control statement is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stop there, and control returns from the loop immediately to the first statement after the loop.

Syntax:-break;

Basically, break statements are used in situations when we are not sure about the actual number of iterations for the loop or we want to terminate the loop based on some condition.



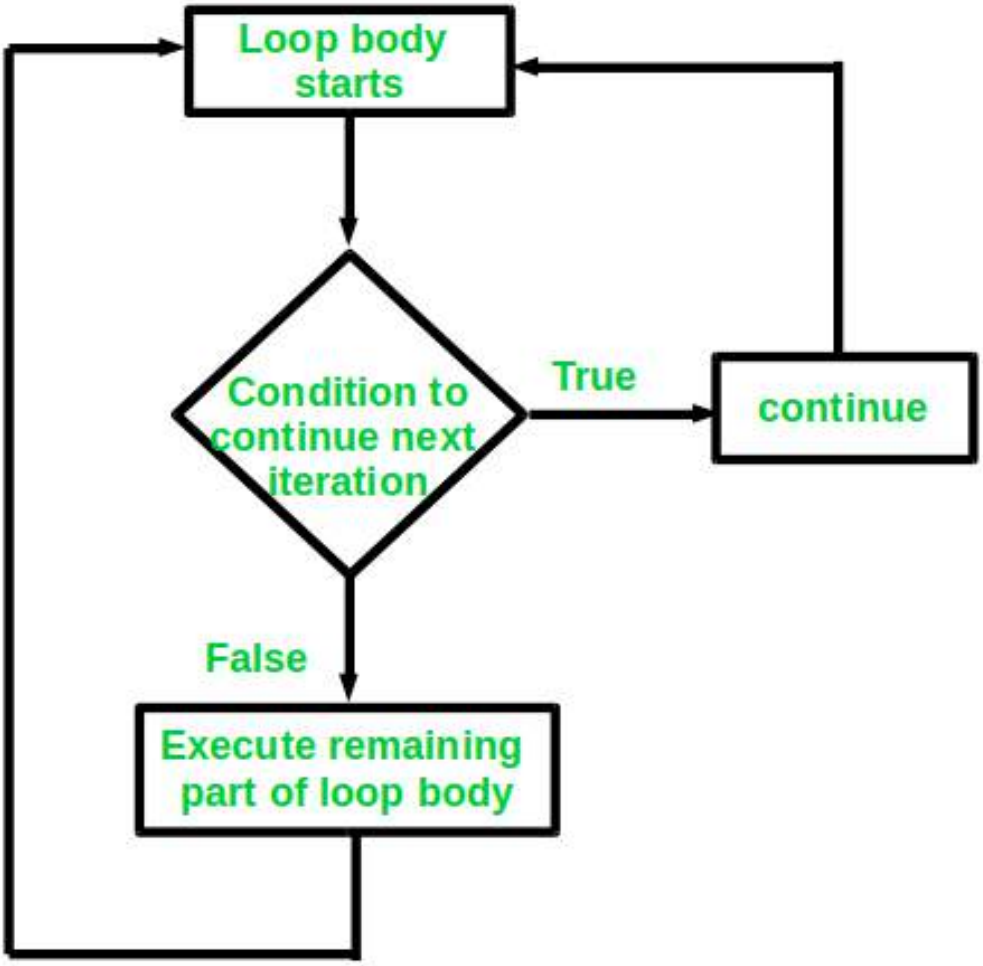
```
#include

void findElement(int arr[], int size, int key)
{
// loop to traverse array and search for key
for (int i = 0; i < size; i++) {
if (arr[i] == key) {
printf("Element found at position: %d", (i + 1));
break;
}
}
}

int main() {
int arr[] = { 1, 2, 3, 4, 5, 6 };
// no of elements
int n = 6;
// key to be searched
int key = 3;
// Calling function to find the key
findElement(arr, n, key);
return 0;
}
```

Output:-

Element found at position: 3

- **Continues:-**This loop control statement is just like the break statement. The continue statement is opposite to that of the break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.
  As the name suggests the continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

Syntax:-continue;

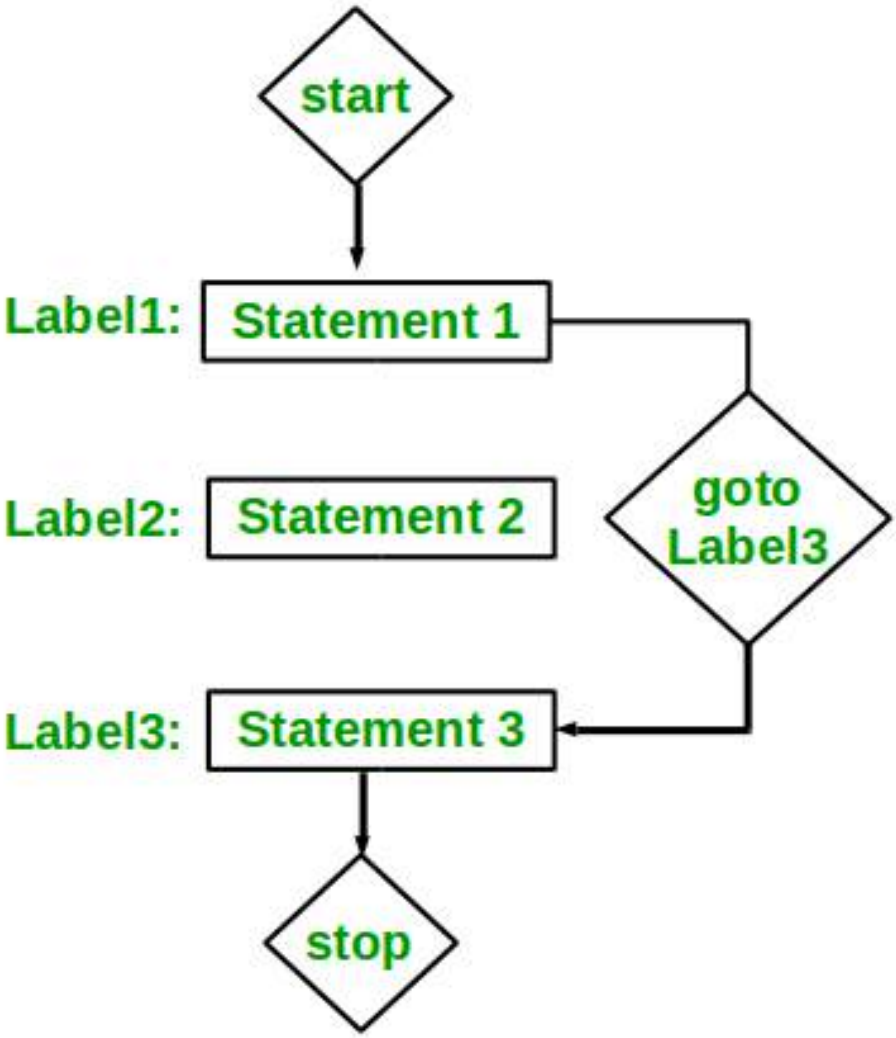



```
#include

int main() {
// loop from 1 to 10
for (int i = 1; i <= 10; i++) {
// If i is equals to 6,
// continue to next iteration
// without printing
if (i == 6)
continue;
else
// otherwise print the value of i
printf("%d ", i);
}
return 0;
}
```

- **goto:-**The goto statement in C/C++ also referred to as unconditional jump statement can be used to jump from one point to another within a function.

Syntax:-goto label;

In the above syntax, the first line tells the compiler to go to or jump to the statement marked as a label. Here label is a user-defined identifier that indicates the target statement. The statement immediately followed after 'label:' is the destination statement. The 'label:' can also appear before the 'goto label;' statement in the above syntax.




```
#include

// function to print numbers from 1 to 10
void printNumbers()
{
int n = 1;
label:
printf("%d ",n);
n++;
```

```
if (n <= 10)
goto label;
}
// Driver program to test above function
int main() {
printNumbers();
return 0;
}
}
```

Output

1 2 3 4 5 6 7 8 9 10

- **return:-**The return in C or C++ returns the flow of the execution to the function from where it is called. This statement does not mandatorily need any conditional statements. As soon as the statement is executed, the flow of the program stops immediately and return the control from where it was called. The return statement may or may not return anything for a void function, but for a non-void function, a return value is must be returned.

Syntax:-return[expression];

Example:-

```
#include

// non-void return type
// function to calculate sum
int SUM(int a, int b)
{
int s1 = a + b;
return s1;
}
// returns void
// function to print
void Print(int s2)
{
printf("The sum is %d", s2);
return;
}
int main()
{
int num1 = 10
; int num2 = 10;
int sum_of = SUM(num1, num2);
```

```
Print(sum_of);
return 0;
}
```

Output

```
The sum is 20
```

Loops

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language. In this part of the tutorial, we are going to learn all the aspects of C loops.

Why use loops in C language?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the printf statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantage of loops in C

1) It provides code reusability.
2) Using loops, we do not need to write the same code again and again.
3) Using loops, we can traverse over the elements of data structures (array or linked lists).

Types of C Loops

There are three types of loops in C language that is given below:
1.do while
2.while
3.for

1.do-while loop

The do while loop is a post tested loop. Using the do-while loop, we can repeat the execution of several parts of the statements. The do-while loop is mainly used in the case where we need to execute the loop at least once. The do-while loop is mostly used in menu-driven programs where the termination condition depends upon the end user.

Syntax:-:

```
do{
//code to be executed
}while(condition);
```

Flowchart:-

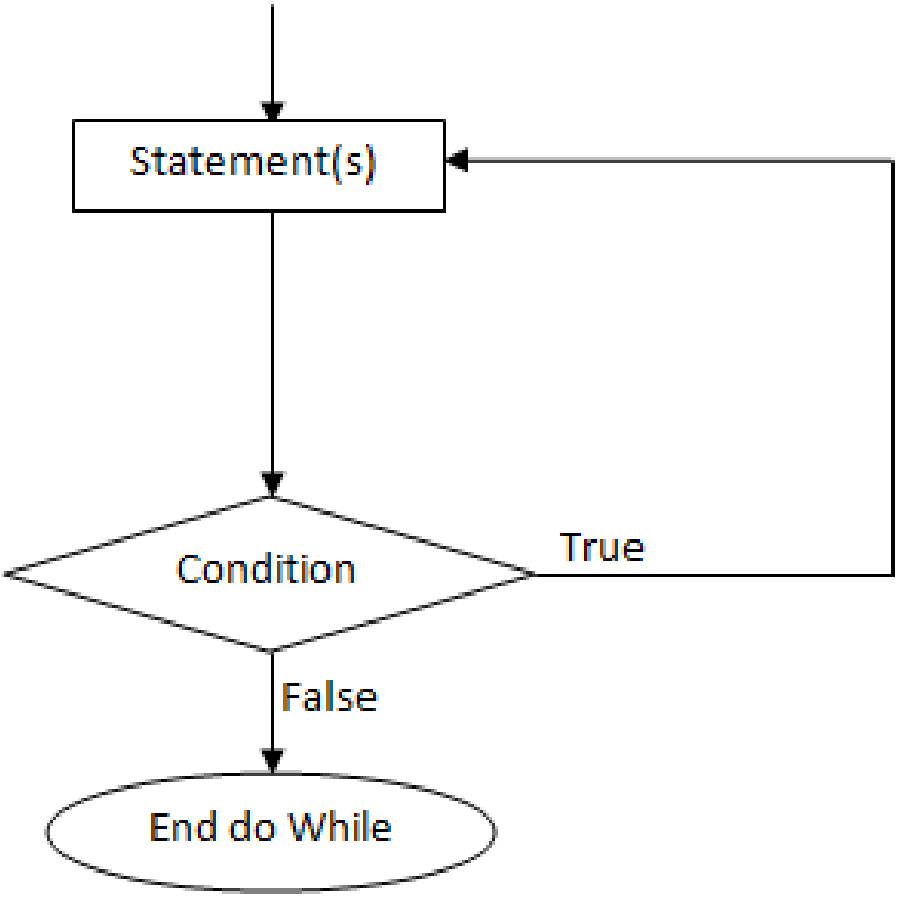


fig: Flowchart for do-while loop

Example:-

```
#include

int main(){
int i=1;
do{
printf("%d \n",i);
i++;
}
while(i<=10);
```

```
return 0;
}
```

Output:-

```
1
2
3
4
5
6
7
8
9
10
```

2).while loop

While loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed multiple times depending upon a given boolean condition. It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

Syntax:-

```
while(condition){
//code to be executed
}
```

Flowchart:-

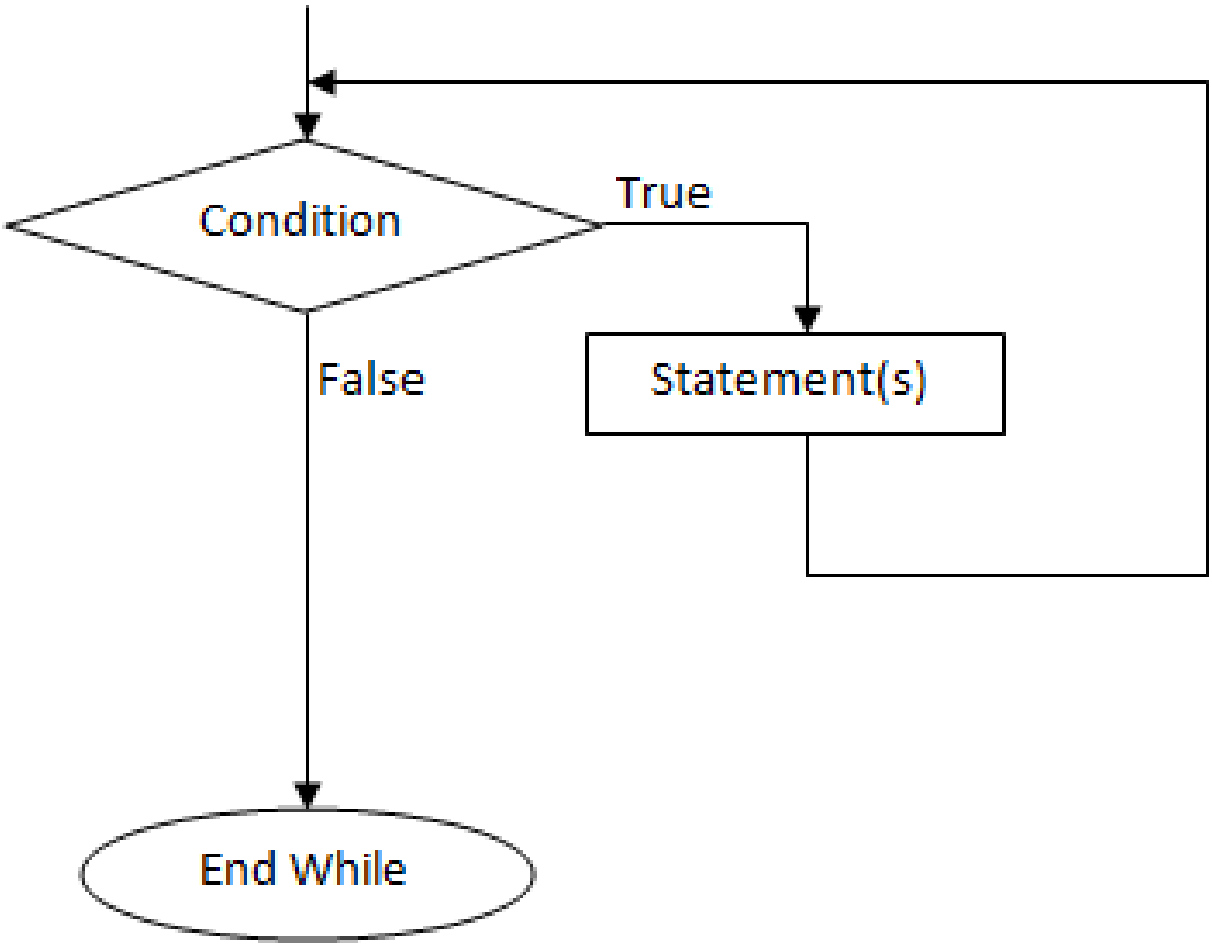


fig: Flowchart for while loop

Properties of while loop:-

- A conditional expression is used to check the condition. The statements defined inside the while loop will repeatedly execute until the given condition fails.
- The condition will be true if it returns 0. The condition will be false if it returns any non-zero number.
- In while loop, the condition expression is compulsory.
- Running a while loop without a body is possible.
- We can have more than one conditional expression in while loop.
- If the loop body contains only one statement, then the braces are optional.

Example:-

```
#include

void main ()
{
int j = 1;
while(j+=2,j<=10)
{
printf("%d ",j);
}
printf("%d",j);
}
```

Output:-

3 5 7 9 11

3).For loop-

The for loop in C language is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

Syntax:-

```
for(Expression 1; Expression 2; Expression 3){
//code to be executed
}
```

Properties of Expression 1:-

- The expression represents the initialization of the loop variable.
- We can initialize more than one variable in Expression 1.
- Expression 1 is optional.
- In C, we can not declare the variables in Expression 1. However, It can be an exception in some compilers.

Properties of Expression 2:-

- Expression 2 is a conditional expression. It checks for a specific condition to be satisfied. If it is not, the loop is terminated.

- Expression 2 can have more than one condition. However, the loop will iterate until the last condition becomes false. Other conditions will be treated as statements.
- Expression 2 is optional.
- Expression 2 can perform the task of expression 1 and expression 3. That is, we can initialize the variable as well as update the loop variable in expression 2 itself.
- We can pass zero or non-zero value in expression 2. However, in C, any non-zero value is true, and zero is false by default.

Properties of Expression 3:-

- Expression 3 is used to update the loop variable.
- We can update more than one variable at the same time.
- Expression 3 is optional.

Flowchart:-

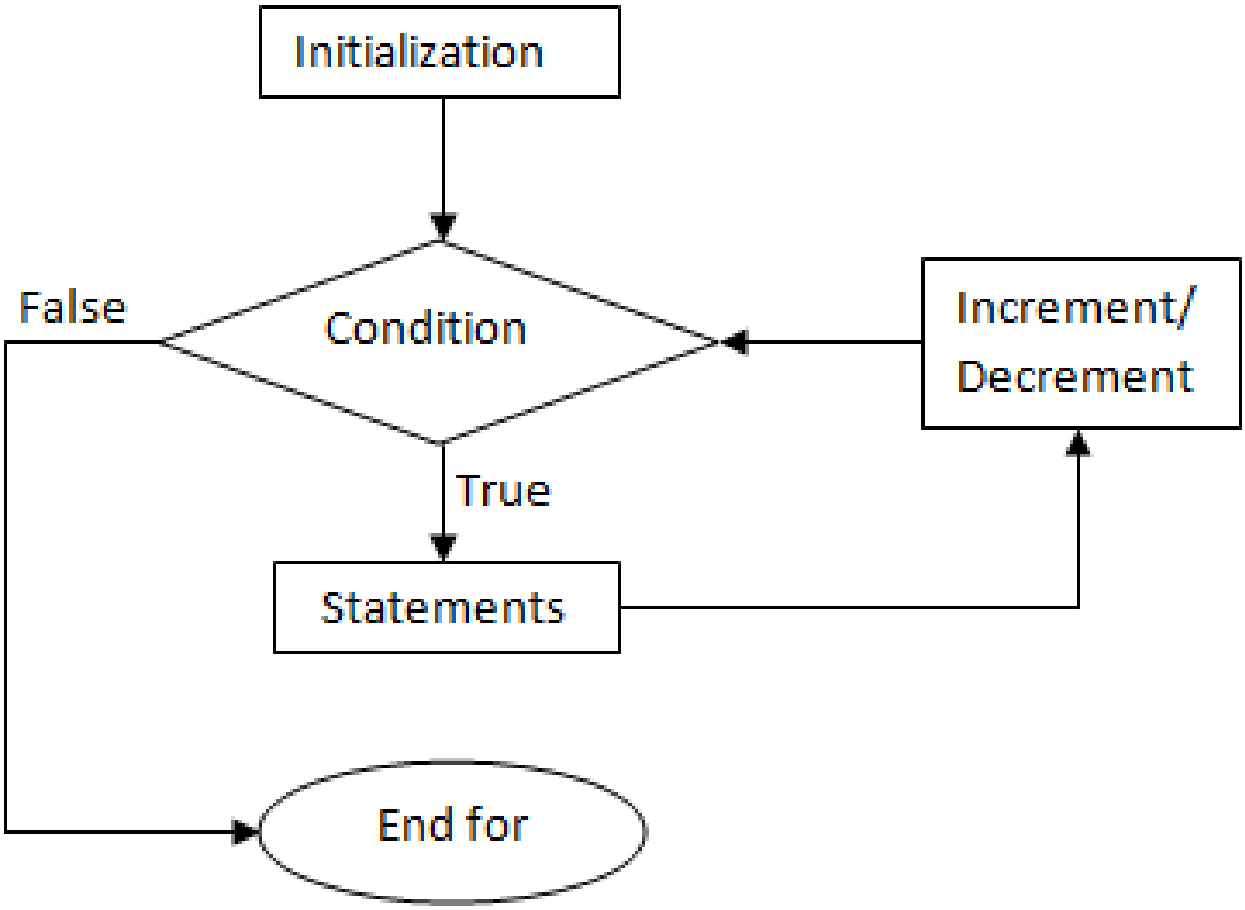


fig: Flowchart for for loop

Example

```
#include

int main(){
int i=1,number=0;
printf("Enter a number: ");
scanf("%d",&number);
for(i=1;i<=10;i++){
printf("%d \n",(number*i));
}
return 0;
}
```

Output:-

```
Enter a number: 1000
1000
```

2000
3000
4000
5000
6000
7000
8000
9000
10000

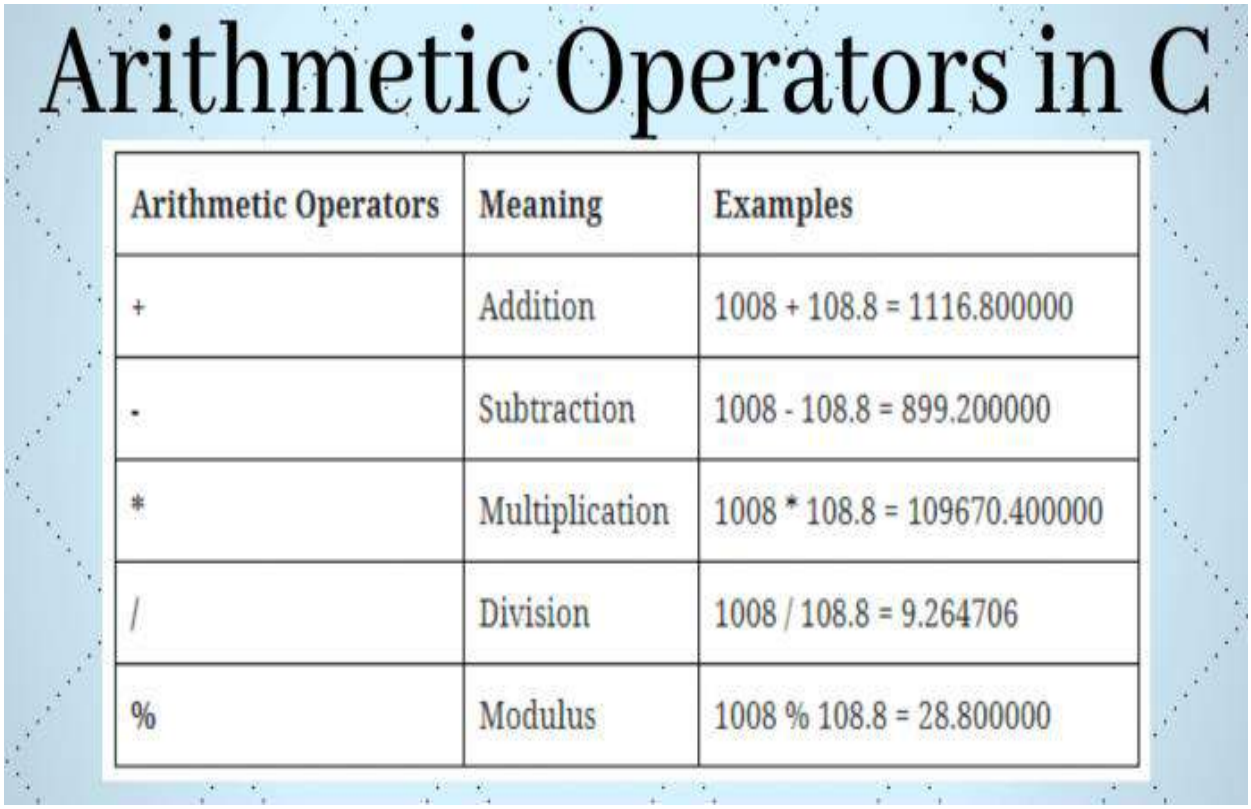
Operators

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

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable A holds some number and variable B holds some number then -



Arithmetic Operators	Meaning	Examples
+	Addition	1008 + 108.8 = 1116.800000
-	Subtraction	1008 - 108.8 = 899.200000
*	Multiplication	1008 * 108.8 = 109670.400000
/	Division	1008 / 108.8 = 9.264706
%	Modulus	1008 % 108.8 = 28.800000

Relational Operators

The following table shows all the relational operators supported by C.Assume variable A holds some number and variable B holds some number then -

Operator	Example	Meaning
>	3 > 0	True; output 1
>=	3 >= 0	True; output 1
<	3 < 0	False; output 0
<=	3 <= 0	False; output 0
==	3 == 0	False; output 0
!=	3 != 0	True; output 1

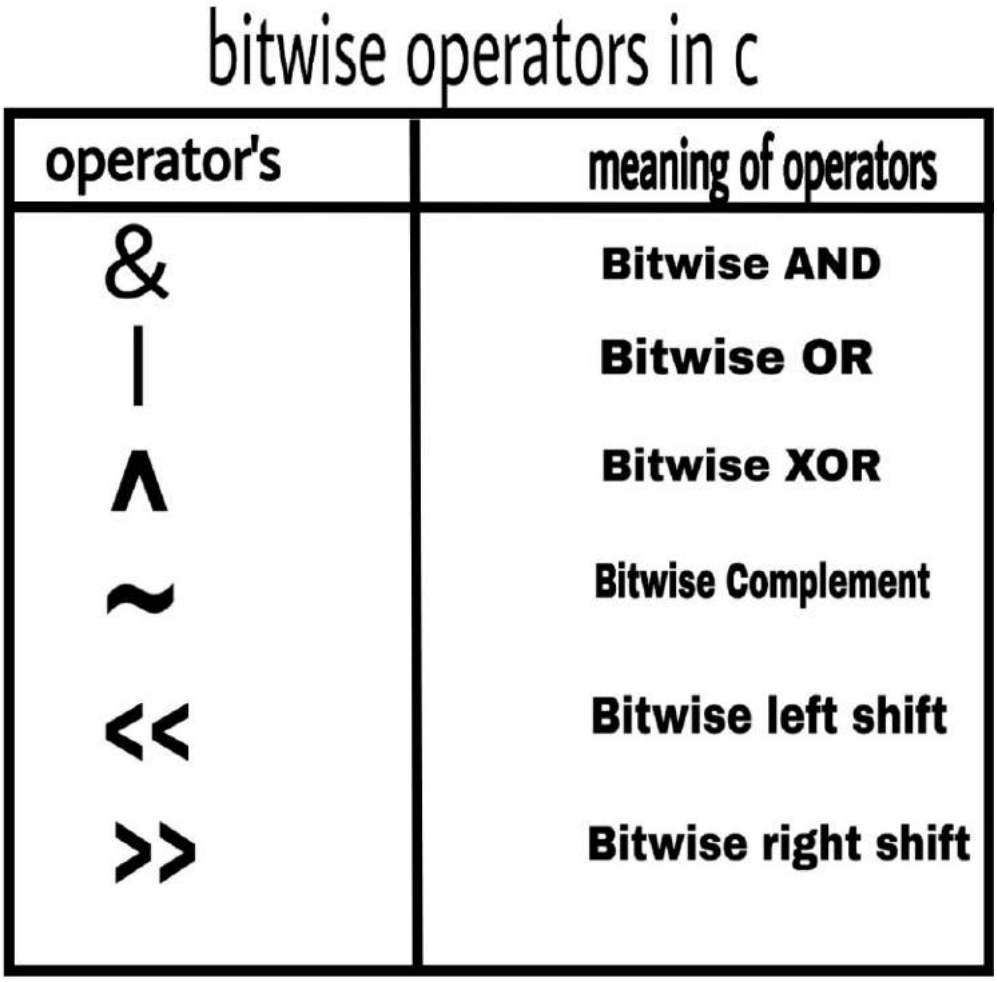
Logical Operators

The following table shows all the Logical Operators supported by C.Assume variable A holds some number and variable B holds some number then -

Logical Operators		
Operator	Description	Example
&&	AND	x=6 y=3 x<10 && y>1 Return True
||	OR	x=6 y=3 x==5 || y==5 Return False
!	NOT	x=6 y=3 !(x==y) Return True

Bitwise Operators

The following table shows all the Bitwise Operators supported by C.Assume variable A holds some number and variable B holds some number then -



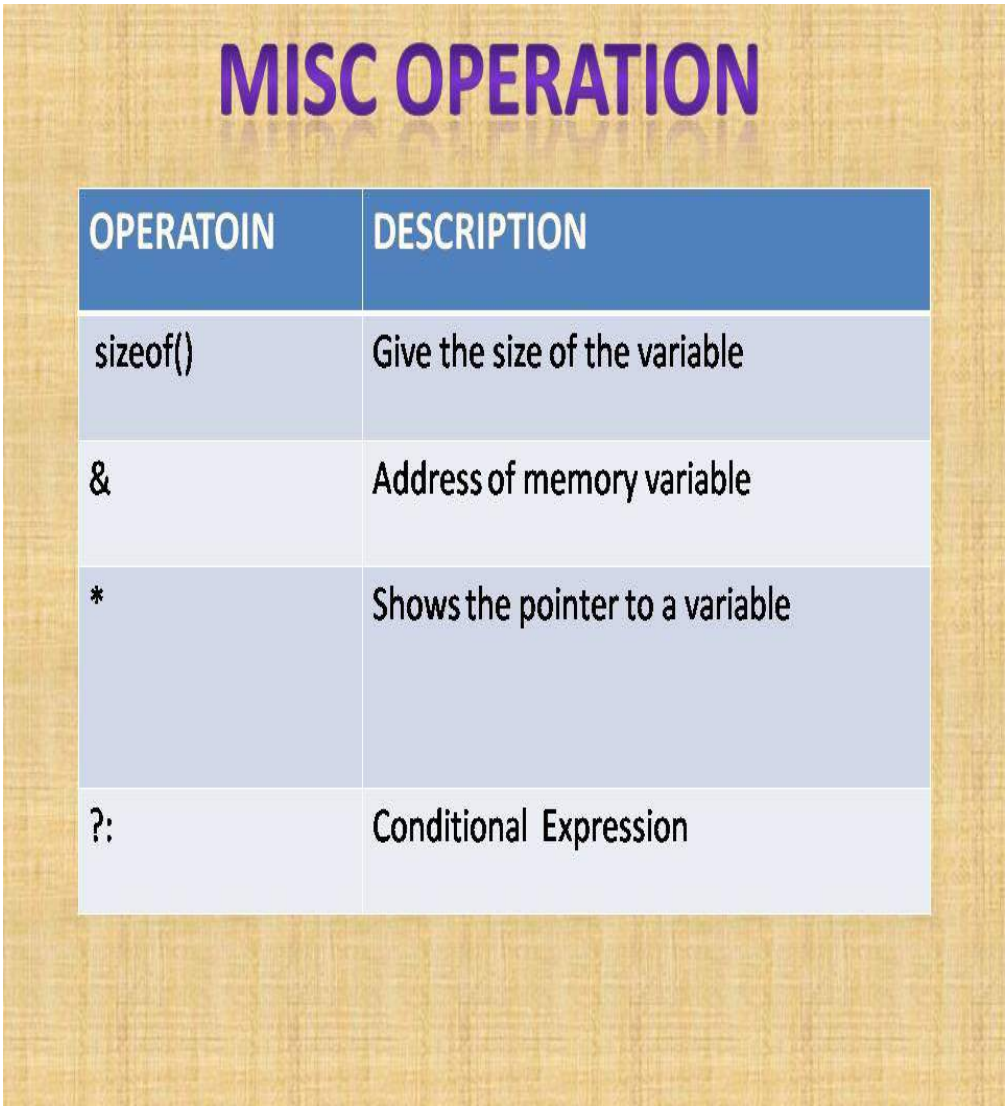
Assignment Operators

The following table shows all the Assignment Operators supported by C.Assume variable A holds some number and variable B holds some number then -

Shorthand Operator	Example	Equivalent expression
+=	x += 2;	x = x + 2;
-=	x -= 2;	x = x - 2;
*=	x *= 2;	x = x * 2;
/=	x /= 2;	x = x / 2;
%=	x %= 2;	x = x % 2;
<<=	x <<= 2;	x = x << 2;
>>=	x >>= 2;	x = x >> 2;
&=	x &= 2;	x = x & 2;
|=	x |= 2;	x = x | 2;
^=	x ^= 2;	x = x ^ 2;

Misc Operators

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



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.

OPERATOR	TYPE	ASSOCIAVITY
( ) [ ] . ->		left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
|	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
||	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right

Overview

C is a general-purpose, high-level language that was originally developed by Dennis M. Ritchie to develop the UNIX operating system at Bell Labs. C was originally first implemented on the DEC PDP-11 computer in 1972.

In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard.

The UNIX operating system, the C compiler, and essentially all UNIX application programs have been written in C. C has now become a widely used professional language for various reasons -

1).Easy to learn
2).Structured language
3).It produces efficient programs
4).It can handle low-level activities
5).It can be compiled on a variety of computer platforms

Facts about C

- C was invented to write an operating system called UNIX.

- C is a successor of B language which was introduced around the early 1970s.

- The language was formalized in 1988 by the American National Standard Institute (ANSI).

- The UNIX OS was totally written in C.

- Today C is the most widely used and popular System Programming Language.

- Most of the state-of-the-art software have been implemented using C.

- Today's most popular Linux OS and RDBMS MySQL have been written in C.

Why use C?

C was initially used for system development work, particularly the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as the code written in assembly language. Some examples of the use of C might be -

- Operating Systems

- Language Compilers

- Assemblers

- Text Editors

- Print Spoolers

- Network Drivers

- Modern Programs

- Databases

- Language Interpreters

- Utilities

Before we study the basic building blocks of the C programming language, let us look at a bare minimum C program structure so that we can take it as a reference in the upcoming chapters.

Hello World Example

A C program basically consists of the following parts -

- Preprocessor Commands

- Functions

- Variables

- Statements & Expressions

- Comments

Let us look at a simple code that would print the words "Hello World" -

```
#include

int main() {
/* my first program in C */
printf("Hello, World! \n");
return 0;
}
```

Let us take a look at the various parts of the above program -

- The first line of the program #include is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.

- The next line int main() is the main function where the program execution begins.

- The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.

- The next line printf(...) is another function available in C which causes the message "Hello, World!" to be displayed on the screen.

- The next line return 0; terminates the main() function and returns the value 0.

Compile and Execute C Program

Let us see how to save the source code in a file, and how to compile and run it. Following are the simple steps -

- Open a text editor and add the above-mentioned code.

- Save the file as hello.c

- Open a command prompt and go to the directory where you have saved the file.

- Type gcc hello.c and press enter to compile your code.

- If there are no errors in your code, the command prompt will take you to the next line and would generate a.out executable file.

- Now, type a.out to execute your program.

- You will see the output "Hello World" printed on the screen.

```
$ gcc hello.c
$ ./a.out
Hello, World!
```

Make sure the gcc compiler is in your path and that you are running it in the directory containing the source file hello.c.

Variable & Constant

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 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 -

When you compile and execute the above program, it produces the following result on Linux -

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 (all bytes have the value 0); the initial value of all other variables are undefined.

Variable Declaration in C

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 for further compilation without requiring the 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 of 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 -

```
#include

// 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;
}
```

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 -
1).**lvalue** - 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.

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

Variables are lvalues and so they may appear 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 -

```
int g = 20; // valid statement
10 = 20; // invalid statement; would generate compile-time error
```

Constant

A constant is a value or variable that can't be changed in the program, for example: 10, 20, 'a', 3.4, "c programming" etc.

There are different types of constants in C programming.

List of Constants in C

Constant	Example
Decimal Constant	10, 20, 450 etc.
Real or Floating-point Constant	10.3, 20.2, 450.6 etc.
Octal Constant	021, 033, 046 etc.
Hexadecimal Constant	0x2a, 0x7b, 0xaa etc.
Character Constant	'a', 'b', 'x' etc.
String Constant	"c", "c program", "c in javatpoint" etc.

2 ways to define constant in C

There are two ways to define constant in C programming.

1).const keyword
2).#define preprocessor

1) C const keyword

The const keyword is used to define constant in C programming.

const float PI=3.14;

Now, the value of PI variable can't be changed.

#include

int main(){
const float PI=3.14;
printf("The value of PI is: %f",PI);

```
return 0;
}
```

Output.

The value of PI is: 3.140000

2).The #define Preprocessor:-

Given below is the form to use #define preprocessor to define a constant -

#define identifier value

The following example explains it in detail -

#include

```
#define LENGTH 10
< #define WIDTH 5
#define NEWLINE '\n'

int main() {
int area;
area = LENGTH * WIDTH;
printf("value of area : %d", area);
printf("%c", NEWLINE);
return 0;
}
```

Output -

value of area : 50