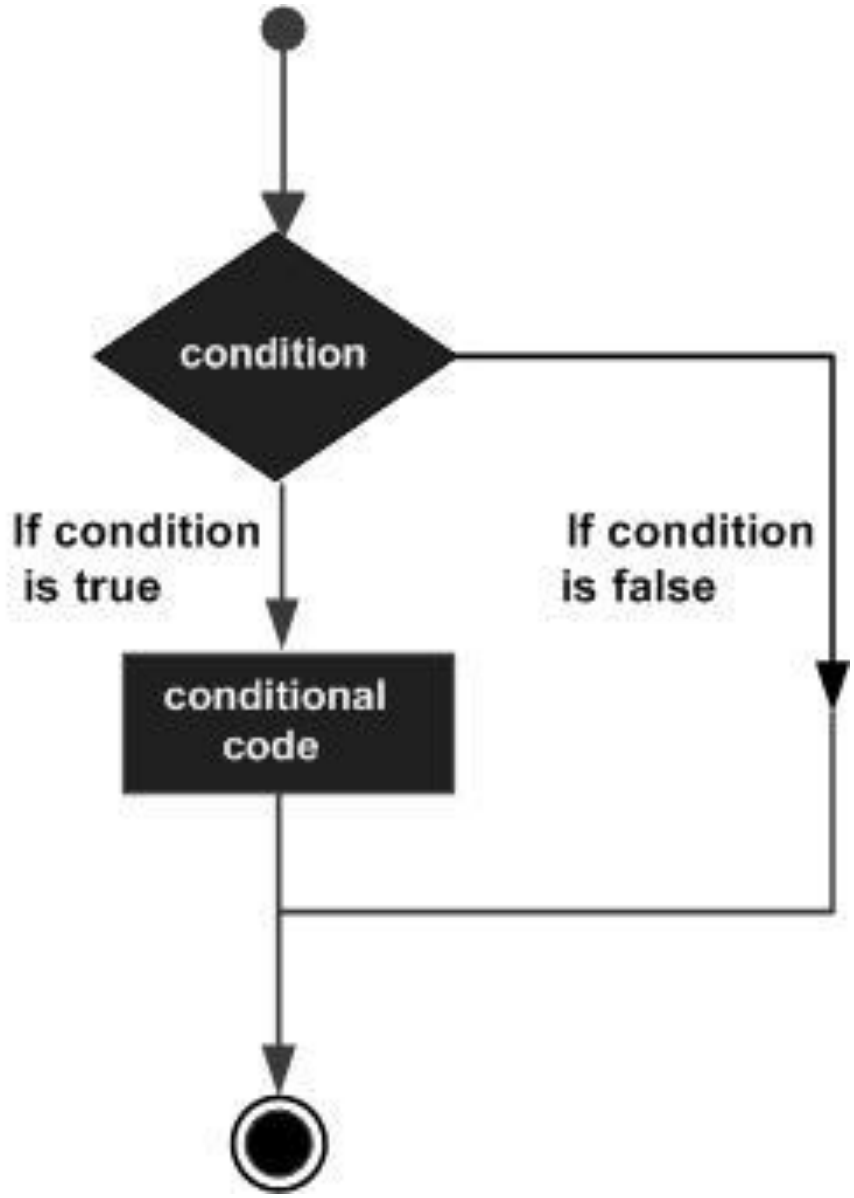
- Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
- Show below is the general form of a typical decision making structure found in most of the programming languages −
- C programming language assumes any non-zero and non-null values as true, and if it is either zero or null, then it is assumed as false value.
- C programming language provides the following types of decision making statements.

Sr.No.	Statement & Description
1	**if statement** An if statement consists of a boolean expression followed by one or more statements.
2	**if...else statement** An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.
3	**nested if statements** You can use one if or else if statement inside another if or else if statement(s).
4	**switch statement** A switch statement allows a variable to be tested for equality against a list of values.
5	**nested switch statements** You can use one switch statement inside another switch statement(s).



Simple if **statement**

The general form of a simple if statement is,

```
if(expression)

{

    statement inside;

}

    statement outside;
```

If the expression returns true, then the statement-inside will be executed, otherwise statement-inside is skipped and only the statement-outside is executed.

Example:

```
#include <stdio.h>



void main( )

{

    int x, y;

    x = 15;

    y = 13;

    if (x > y )

    {

        printf("x is greater than y");

    }

}
```

```
x is greater than y
```

---

if...else statement

The general form of a simple if...else statement is,

```
if(expression)

{
```

```
    statement block1;

}

else

{

    statement block2;

}
```

If the expression is true, the statement-block1 is executed, else statement-block1 is skipped and statement-block2 is executed.

Example:

```
#include <stdio.h>



void main( )

{

    int x, y;

    x = 15;

    y = 18;

    if (x > y )

    {

        printf("x is greater than y");

    }

    else

    {
```

```
        printf("y is greater than x");

    }

}
```

y is greater than x

Nested if....else statement

The general form of a nested if...else statement is,

```
if( expression )

{

    if( expression1 )

    {

        statement block1;

    }

    else

    {

        statement block2;

    }

}

else

{

    statement block3;
```

```
}
```

if expression is false then statement-block3 will be executed, otherwise the execution continues and enters inside the first if to perform the check for the next if block, where if expression 1 is true the statement-block1 is executed otherwise statement-block2 is executed.

Example:

```
#include <stdio.h>



void main( )

{

    int a, b, c;

    printf("Enter 3 numbers...");

    scanf("%d%d%d",&a, &b, &c);

    if(a > b)

    {

        if(a > c)

        {

            printf("a is the greatest");

        }

        else

        {

            printf("c is the greatest");

        }
```

```
    }
    else
    {
        if(b > c)
        {
            printf("b is the greatest");
        }
        else
        {
            printf("c is the greatest");
        }
    }
}
```

---

else if ladder

The general form of else-if ladder is,

```
if(expression1)
{
    statement block1;
}
else if(expression2)
```

```
{
    statement block2;
}
else if(expression3 )
{
    statement block3;
}
else
    default statement;
```

The expression is tested from the top(of the ladder) downwards. As soon as a true condition is found, the statement associated with it is executed.

Example :

```
#include <stdio.h>

void main( )
{
    int a;
    printf("Enter a number...");
    scanf("%d", &a);
    if(a%5 == 0 && a%8 == 0)
    {
```

```
        printf("Divisible by both 5 and 8");

    }

    else if(a%8 == 0)

    {

        printf("Divisible by 8");

    }

    else if(a%5 == 0)

    {

        printf("Divisible by 5");

    }

    else

    {

        printf("Divisible by none");

    }

}
```

Points to Remember

In if statement, a single statement can be included without enclosing it into curly braces { ... }

```
1. int a = 5;

2. if(a > 4)

        printf("success");
```

No curly braces are required in the above case, but if we have more than one statement inside if condition, then we must enclose them inside curly braces.

3. == must be used for comparison in the expression of if condition, if you use = the expression will always return true, because it performs assignment not comparison.
4. Other than 0(zero), all other values are considered as true.

```
5. if(27)
```

```
    printf("hello");
```

In above example, hello will be printed.

Switch statement in C

- When you want to solve multiple option type problems, for example: Menu like program, where one value is associated with each option and you need to choose only one at a time, then, switch statement is used.
- Switch statement is a control statement that allows us to choose only one choice among the many given choices. The expression in switch evaluates to return an integral value, which is then compared to the values present in different cases. It executes that block of code which matches the case value. If there is no match, then default block is executed(if present). The general form of switch statement is,

```
switch(expression)
{
   case value-1:
        block-1;
        break;
   case value-2:
        block-2;
        break;
   case value-3:
        block-3;
        break;
   case value-4:
        block-4;
           break;
   default:
           default-block;
        break;
}
```

Rules for using switch statement

The expression (after switch keyword) must yield an integer value i.e the expression should be an integer or a variable or an expression that evaluates to an integer.

- The case label values must be unique.
- The case label must end with a colon(:)
- The next line, after the case statement, can be any valid C statement.

Points to Remember

1. We don't use those expressions to evaluate switch case, which may return floating point values or strings or characters.
2. break statements are used to exit the switch block. It isn't necessary to use break after each block, but if you do not use it, then all the consecutive blocks of code will get executed after the matching block.

```
3. int i = 1;
4. switch(i)
5. {
6.      case 1:
7.          printf("A");          // No break
8.      case 2:
9.          printf("B");          // No break
10.     case 3:
11.         printf("C");
12.         break;
}
```

```
A B C
```

The output was supposed to be only A because only the first case matches, but as there is no break statement after that block, the next blocks are executed too, until it a break statement in encountered or the execution reaches the end of the switch block.

13. default case is executed when none of the mentioned case matches the switch expression. The default case can be placed anywhere in the switch case. Even if we don't include the default case, switch statement works.

14. Nesting of switch statements are allowed, which means you can have switch statements inside another switch block. However, nested switch statements should be avoided as it makes the program more complex and less readable.

C break statement

- The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios:
- With switch case
- With loop

Syntax:

```
1. //loop or switch case
2. break;
```

Flowchart of break in c

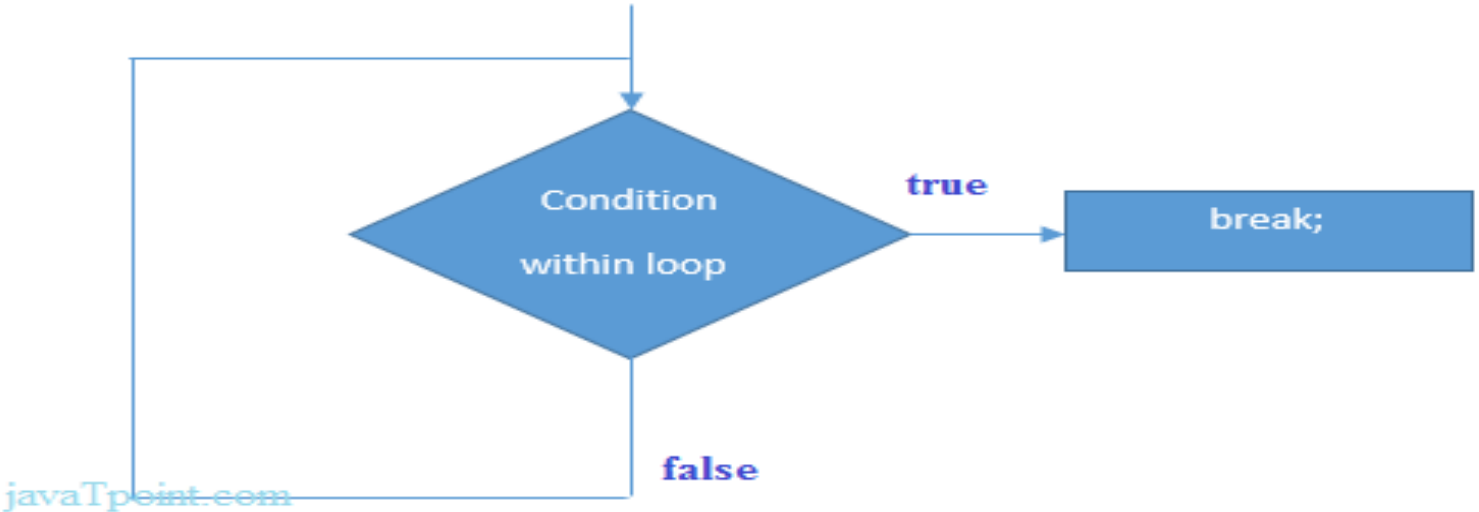


Figure: Flowchart of break statement

Example

```
1. #include<stdio.h>
2. #include<stdlib.h>
3. void main ()
4. {
5.     int i;
6.     for(i = 0; i<10; i++)
7.     {
8.         printf("%d ",i);
9.         if(i == 5)
10.        break;
11.    }
12.    printf("came outside of loop i = %d",i);
13.
14.}
```

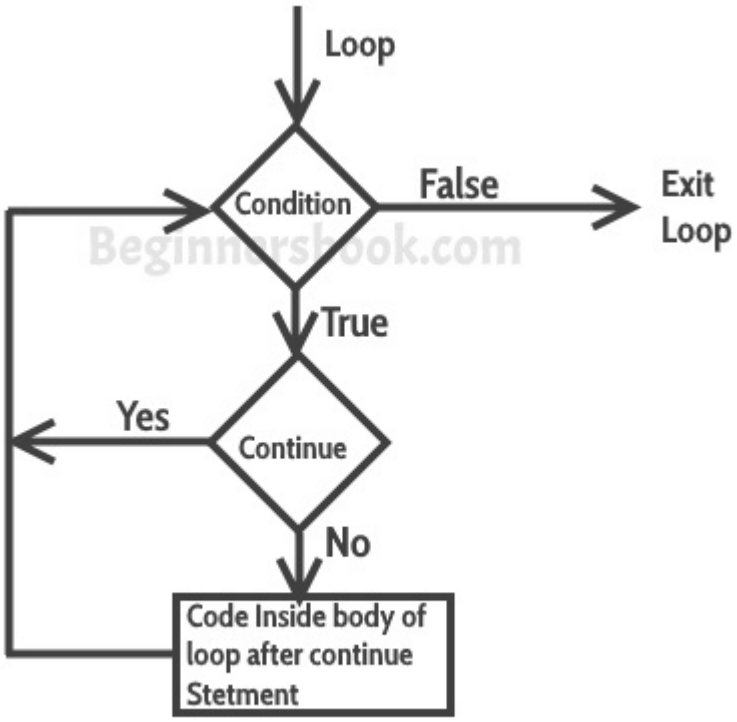
Output

```
0 1 2 3 4 5 came outside of loop i = 5
```

C – Continue statement

```
15.                 Example: continue statement inside for loop
16.#include <stdio.h>
17.int main()
18.{
19.   for (int j=0; j<=8; j++)
20.   {
21.     if (j==4)
22.     {
23.            /* The continue statement is encountered when
24.             * the value of j is equal to 4.
25.             */
26.            continue;
27.      }
```

```
28.
29.       /* This print statement would not execute for the
30.          * loop iteration where j ==4  because in that case
31.          * this statement would be skipped.
32.          */
33.       printf("%d ", j);
34.    }
35.   return 0;
36.}
37.Output:
38.0 1 2 3 5 6 7 8
```



The ? : Operator

We have covered conditional operator ? : in the previous chapter which can be used to replace if...else statements. It has the following general form −

Exp1 ? Exp2 : Exp3;

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

The value of a ? expression is determined like this −

- Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression.

- If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

Goto Statement

A goto statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

NOTE − Use of goto statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.

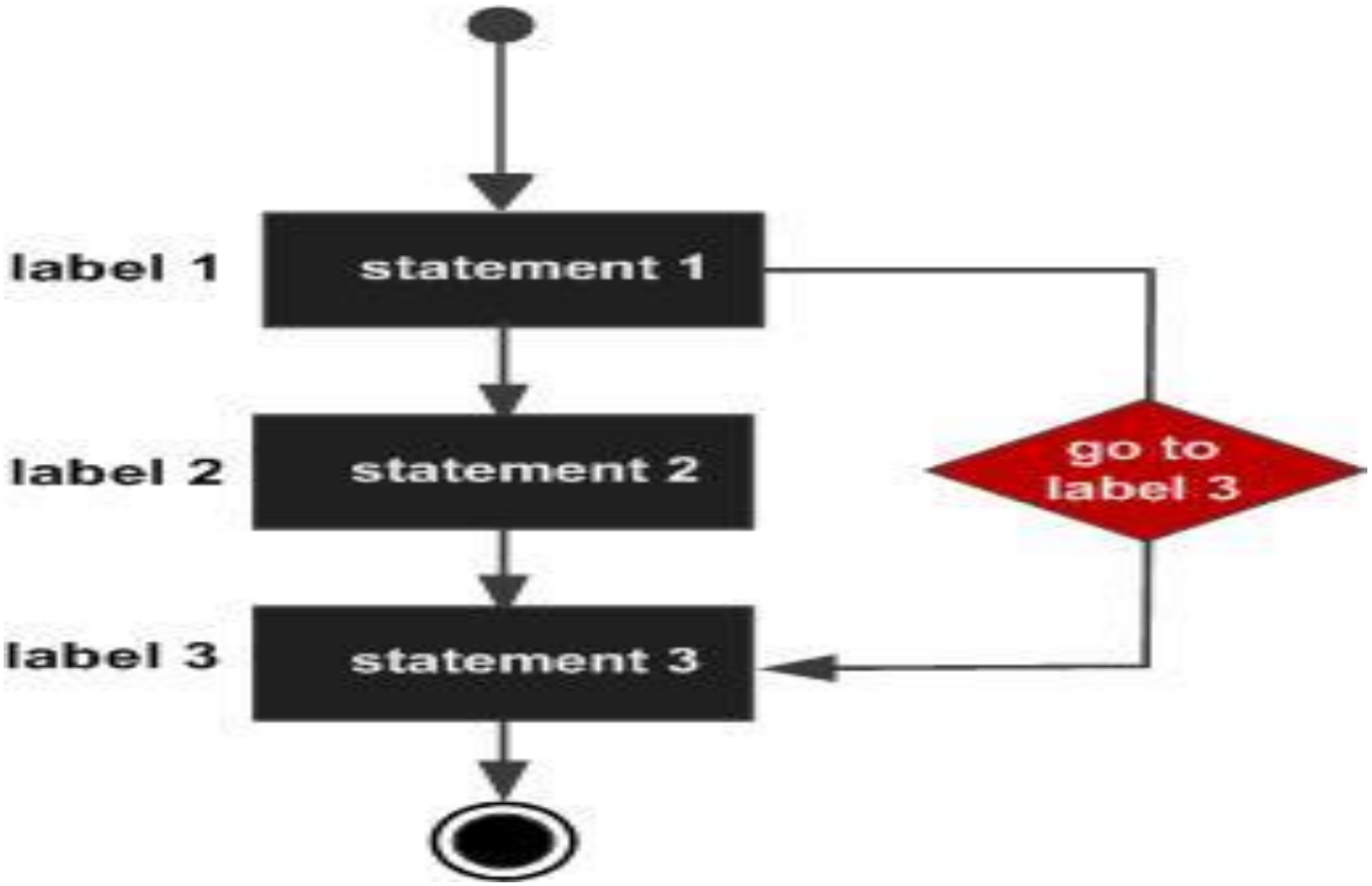
Syntax

The syntax for a goto statement in C is as follows −

```
goto label;
..
.
label: statement;
```

Here label can be any plain text except C keyword and it can be set anywhere in the C program above or below to goto statement.

Flow Diagram



Example

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   LOOP:do {

      if( a == 15) {
         /* skip the iteration */
         a = a + 1;
         goto LOOP;
      }

      printf("value of a: %d\n", a);
      a++;

   }
while( a < 20 );
   return 0;
}
```

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

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Loops in C

Loops in programming come into use when we need to repeatedly execute a block of statements. For example: Suppose we want to print “Hello World” 10 times. This can be done in two ways as shown below:

An iterative method to do this is to write the printf() statement 10 times.

Program

```
// C program to illustrate need of loops
#include <stdio.h>
int main()
{
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    return 0;
}
```

Output:

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

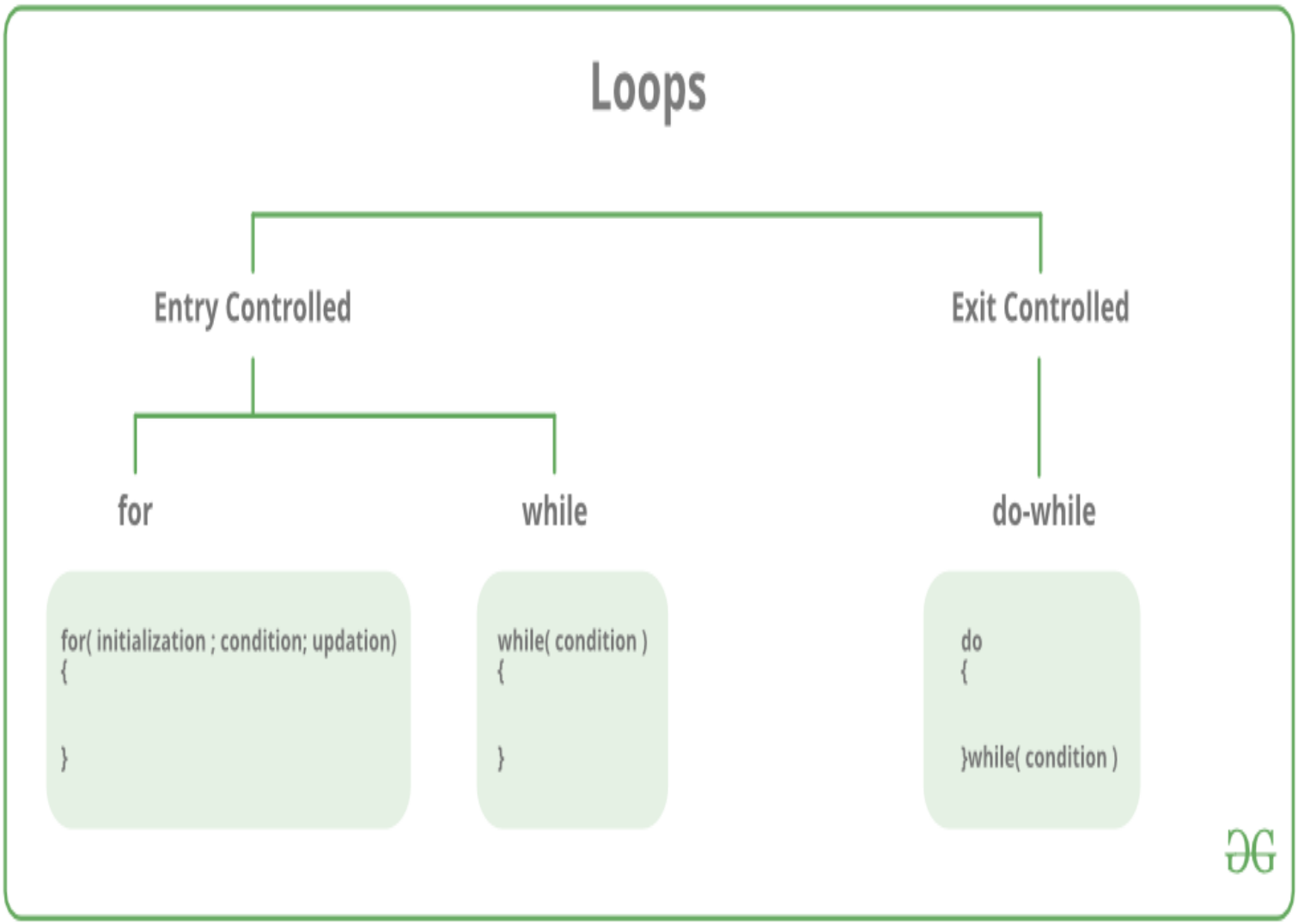
Using Loops

In Loop, the statement needs to be written only once and the loop will be executed 10 times as shown below.

- In computer programming, a loop is a sequence of instructions that is repeated until a certain condition is reached.
- An operation is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number.
- Counter not Reached: If the counter has not reached the desired number, the next instruction in the sequence returns to the first instruction in the sequence and repeat it.
- Counter reached: If the condition has been reached, the next instruction "falls through" to the next sequential instruction or branches outside the loop.

There are mainly two types of loops:

- Entry Controlled loops: In this type of loops the test condition is tested before entering the loop body. For Loop and While Loop are entry controlled loops.
- Exit Controlled Loops: In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore, the loop body will execute atleast once, irrespective of whether the test condition is true or false. do – while loop is exit controlled loop.



for Loop

A for loop is a repetition control structure which allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.

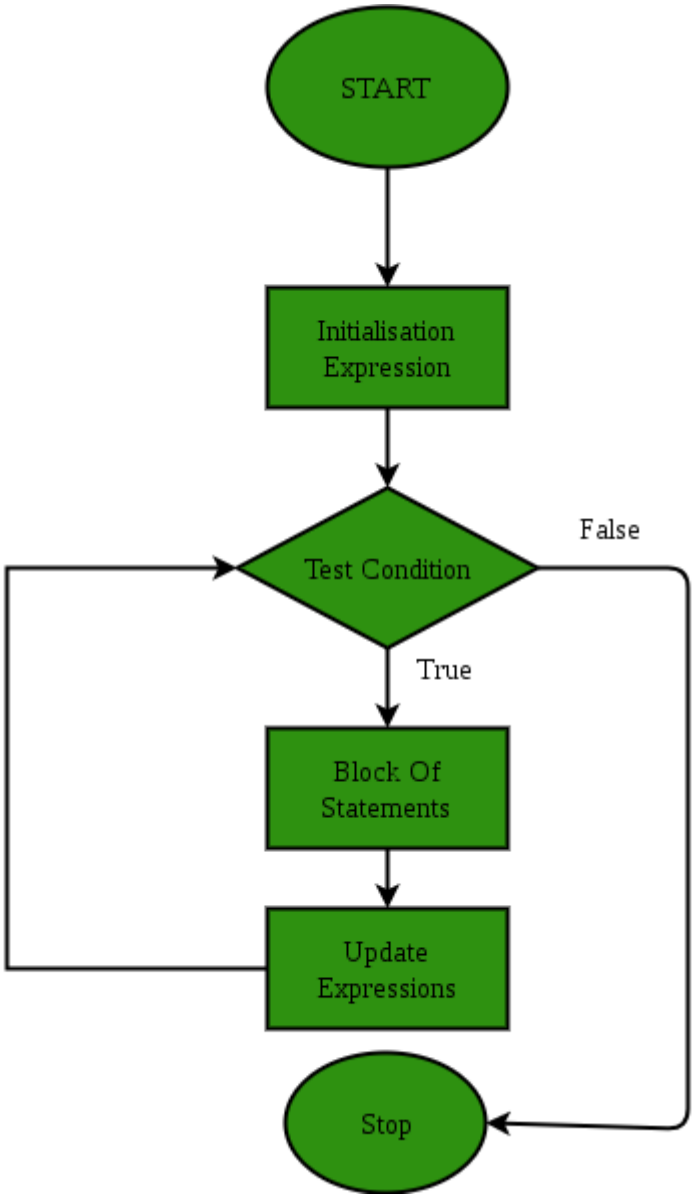
Syntax:

```
for (initialization expr; test expr; update expr)
{
    // body of the loop
    // statements we want to execute
}
```

In for loop, a loop variable is used to control the loop. First initialize this loop variable to some value, then check whether this variable is less than or greater than counter value. If statement is true, then loop body is executed and loop variable gets updated . Steps are repeated till exit condition comes.

- Initialization Expression: In this expression we have to initialize the loop counter to some value. for example: int i=1;
- Test Expression: In this expression we have to test the condition. If the condition evaluates to true then we will execute the body of loop and go to update expression otherwise we will exit from the for loop. For example: i <= 10;
- Update Expression: After executing loop body this expression increments/decrements the loop variable by some value. for example: i++;

Equivalent flow diagram for loop :



Example:

```
// C program to illustrate for loop
#include <stdio.h>
int main()
{
   int i=0;
   for (i = 1; i <= 10; i++)
   {
      printf( "Hello World\n");
   }
   return 0;
}
```

Output:

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

Nested Loops in C

- C supports nesting of loops in C. **Nesting of loops** is the feature in C that allows the looping of statements inside another loop. Let's observe an example of nesting loops in C.
- Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops. The nesting level can be defined at n times.

Syntax of Nested loop

```
Outer_loop
{
   Inner_loop
  {
       // inner loop statements.
  }
     // outer loop statements.
}
```

Nested for loop

The nested for loop means any type of loop which is defined inside the 'for' loop.

```
for (initialization; condition; update)
{
   for(initialization; condition; update)
   {
        // inner loop statements.
   }
   // outer loop statements.
}
```

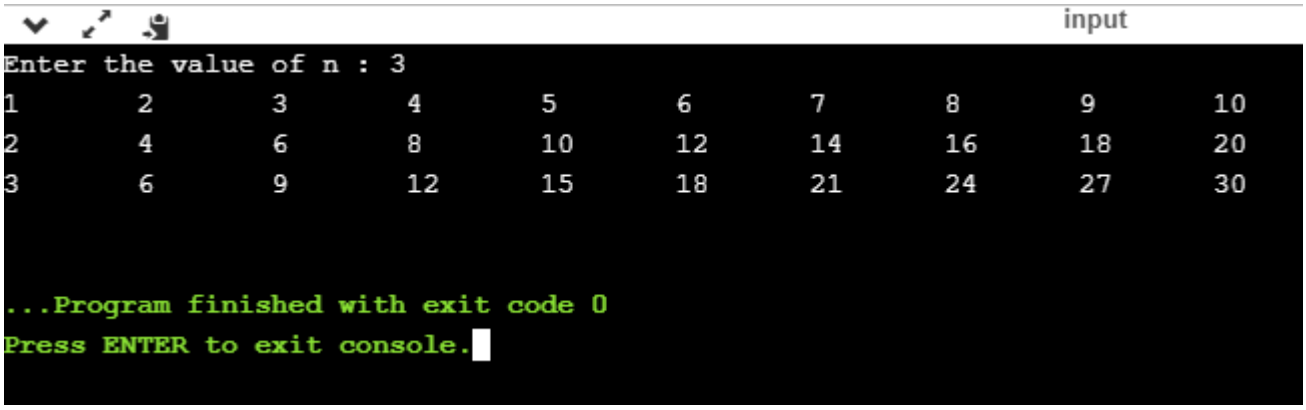
Example of nested for loop

```
#include <stdio.h>
int main()
{
  int n;// variable declaration
  printf("Enter the value of n :");
  // Displaying the n tables.
  for(int i=1;i<=n;i++)  // outer loop
  {
     for(int j=1;j<=10;j++)  // inner loop
     {
        printf("%d\t",(i*j)); // printing the value.
       printf("\n");
  }
```

Explanation of the above code

- First, the 'i' variable is initialized to 1 and then program control passes to the i<=n.
- The program control checks whether the condition 'i<=n' is true or not.
- If the condition is true, then the program control passes to the inner loop.
- The inner loop will get executed until the condition is true.
- After the execution of the inner loop, the control moves back to the update of the outer loop, i.e., i++.
- After incrementing the value of the loop counter, the condition is checked again, i.e., i<=n.
- If the condition is true, then the inner loop will be executed again.
- This process will continue until the condition of the outer loop is true.

Output:



While Loop

- While studying for loop we have seen that the number of iterations is known beforehand, i.e. the number of times the loop body is needed to be executed is known to us. while loops are used in situations where we do not know the exact number of iterations of loop beforehand. The loop execution is terminated on the basis of test condition.
- We have already stated that a loop is mainly consisted of three statements – initialization expression, test expression, update expression. The syntax of the three loops – For, while and do while mainly differs on the placement of these three statements.

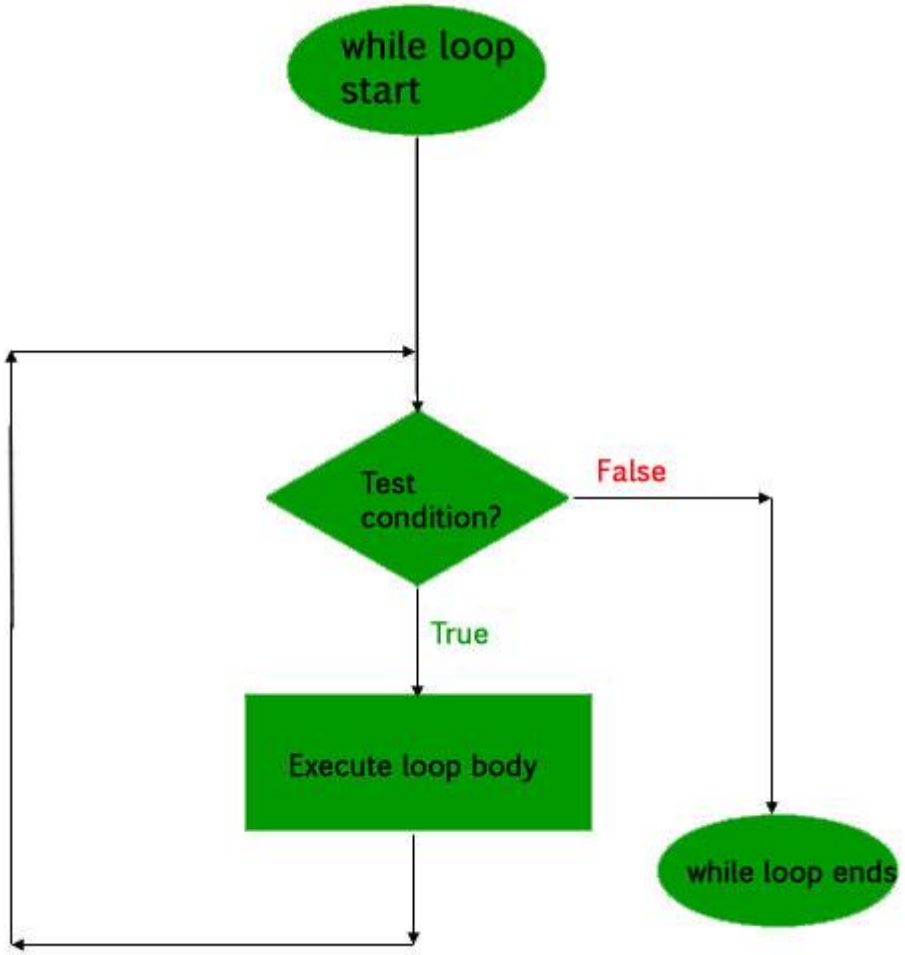
Syntax:

Initialization expression;

while (test_expression)

```
{
   // statements
  update_expression;
}
```

Flow Diagram:



Example:

```
// C program to illustrate while loop
#include <stdio.h>
int main()
{
```

```
    // initialization expression
    int i = 1;
    // test expression
    while (i < 6)
    {
        printf( "Hello World\n");
        // update expression
        i++;
    }
    return 0;
}
```

Output:

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

do while loop

In do while loops also the loop execution is terminated on the basis of test condition. The main difference between do while loop and while loop is in do while loop the condition is tested at the end of loop body, i.e do while loop is exit controlled whereas the other two loops are entry controlled loops.

Note: In do while loop the loop body will execute at least once irrespective of test condition.

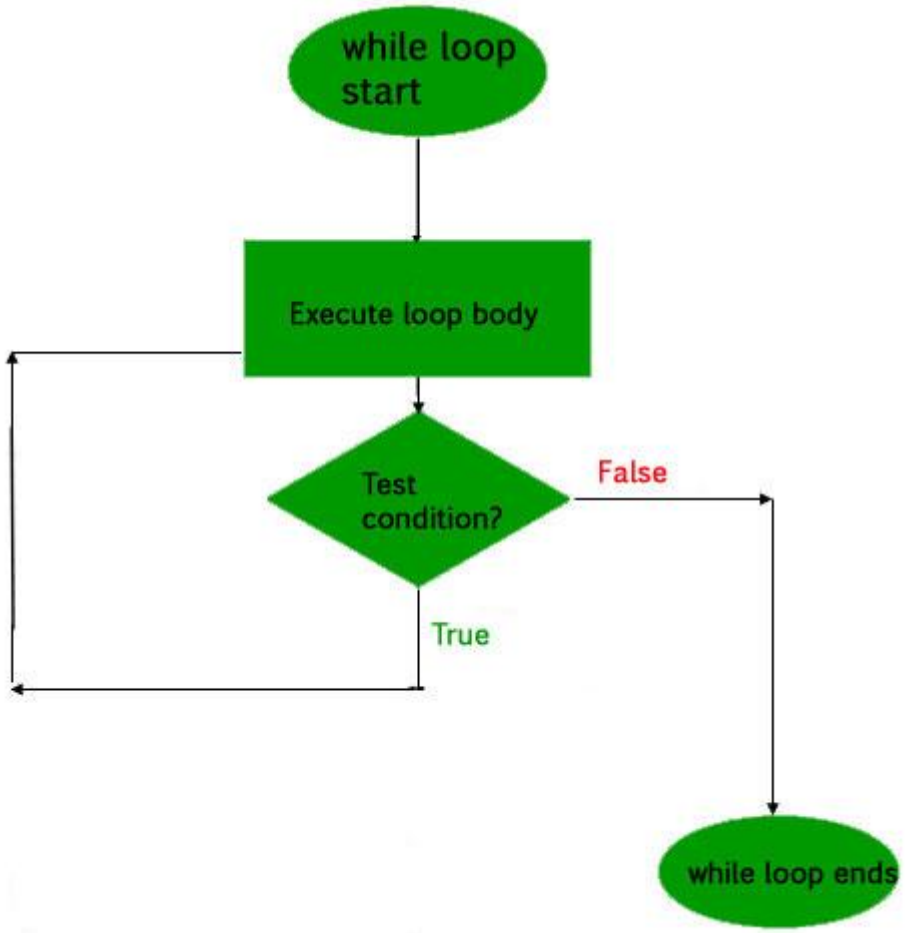
Syntax:

```
initialization expression;
do
{
  // statements
```

```
    update_expression;
} while (test_expression);
```

Note: Notice the semi – colon(“;”) in the end of loop.

Flow Diagram:



Example:

```
// C program to illustrate do-while loop
#include <stdio.h>
int main()
{
    int i = 2; // Initialization expression
```

```
    do
    {
        // loop body
        printf( "Hello World\n");
        // update expression
        i++;
    }  while (i < 1);   // test expression
    return 0;
}
```

Output:

Hello World

In the above program the test condition (i<1) evaluates to false. But still as the loop is exit – controlled the loop body will execute once.

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};

- The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

- 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 −

Accessing Array Elements

- An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −

double salary = balance[9];

The above statement will take the 10th element from the array and assign the value to salary variable. The following example shows how to use all the three above mentioned concepts viz. declaration, assignment, and accessing arrays −

```
#include <stdio.h>

int main ()
{
  int n[ 10 ]; /* n is an array of 10 integers */
  int i,j;
  /* initialize elements of array n to 0 */
  for ( i = 0; i < 10; i++ ) {
    n[ i ] = i + 100; /* set element at location i to i + 100 */
  }
  /* output each array element's value */
  for (j = 0; j < 10; j++ ) {
    printf("Element[%d] = %d\n", j, n[j] );
  }
  return 0;
```

}

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

Element[0] = 100

Element[1] = 101

Element[2] = 102

Element[3] = 103

Element[4] = 104

Element[5] = 105

Element[6] = 106

Element[7] = 107

Element[8] = 108

Element[9] = 109

Arrays in Detail

Arrays are important to C and should need a lot more attention. The following important concepts related to array should be clear to a C programmer −

S.No.	Concept & Description
1	**Multi-dimensional arrays** C supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array.
2	**Passing arrays to functions** You can pass to the function a pointer to an array by specifying the array's name without an index.
3	**Return array from a function** C allows a function to return an array.
4	**Pointer to an array** You can generate a pointer to the first element of an array by simply specifying the array name, without any index.

One-dimensional array

Conceptually you can think of a one-dimensional array as a row, where elements are stored one after another.

Syntax: `datatype array_name[size];`

datatype: It denotes the type of the elements in the array.

array_name: Name of the array. It must be a valid identifier.

size: Number of elements an array can hold. here are some example of array declarations:

```
int num[100];
float temp[20];
char ch[50];
```

num is an array of type int, which can only store 100 elements of type int.
temp is an array of type float, which can only store 20 elements of type float.
ch is an array of type char, which can only store 50 elements of type char.

Note: When an array is declared it contains garbage values.

The individual elements in the array:

```
num[0], num[1], num[2], ....., num[99]
temp[0], temp[1], temp[2], ....., temp[19]
ch[0], ch[1], ch[2], ....., ch[49]
```

Two dimensional (2D) arrays in C programming with example

An array of arrays is known as 2D array. The two dimensional (2D) array in c programming is also known as matrix. A matrix can be represented as a table of rows and columns. Before we discuss more about two Dimensional array lets have a look at the following C program.

Simple Two dimensional(2D) Array Example

For now don’t worry how to initialize a two dimensional array, we will discuss that part later. This program demonstrates how to store the elements entered by user in a 2d array and how to display the elements of a two dimensional array.

```
#include<stdio.h>
int main()
{
  /* 2D array declaration*/
  int disp[2][3];
  /*Counter variables for the loop*/
  int i, j;
  for(i=0; i<2; i++) {
    for(j=0;j<3;j++) {
      printf("Enter value for disp[%d][%d]:", i, j);
      scanf("%d", &disp[i][j]);
    }
  }
  //Displaying array elements
  printf("Two Dimensional array elements:\n");
  for(i=0; i<2; i++) {
    for(j=0;j<3;j++) {
      printf("%d ", disp[i][j]);
      if(j==2){
        printf("\n");
      }
    }
  }
  return 0;
}
```

Output:

```
Enter value for disp[0][0]:1
Enter value for disp[0][1]:2
```

Enter value for disp[0][2]:3

Enter value for disp[1][0]:4

Enter value for disp[1][1]:5

Enter value for disp[1][2]:6

Two Dimensional array elements:

1 2 3

4 5 6

Multi Dimensional Array in C

In C Programming Language, by placing n number of brackets [ ], we can declare n-dimensional array where n is dimension number. For example,

int a[2][3][4] = Three Dimensional Array

int a[2][2][3][4] = Four Dimensional Array

Syntax of a Multi Dimensional Array in C Programming

Data_Type Array_Name[Tables][Row_Size][Column_Size]

- Data_type: It will decide the type of elements it will accept. For example, If we want to store integer values then we declare the Data Type as int, If we want to store Float values then we declare the Data Type as float etc
- Array_Name: This is the name you want to give it to Multi Dimensional array in C.
- Tables: It will decide the number of tables an array can accept. Two Dimensional Array is always a single table with rows and columns. In contrast, Multi Dimensional array in C is more than 1 table with rows and columns.
- Row_Size: Number of Row elements an array can store. For example, Row_Size =10, the array will have 10 rows.
- Column_Size: Number of Column elements an array can store. For example, Column_Size = 8, the array will have 8 Columns.

We can calculate the maximum number of elements in a Three Dimensional using: [Tables] * [Row_Size] * [Column_Size]

C Multi Dimensional Array Initialization

int Employees[2][4][3] = { { {10, 20, 30}, {15, 25, 35}, {22, 44, 66}, {33, 55, 77} }, { {1, 2, 3}, {5, 6, 7}, {2, 4, 6}, {3, 5, 7} }

- Here, We have 2 tables and the 1st table holds 4 Rows * 3 Columns, and the 2nd table also holds 4 Rows * 3 Columns

- The first three elements of the first table will be 1st row, the second three elements will be 2nd row, the next three elements will be 3rdrow, and the last 3 elements will be 4th row. Here we divided them into 3 because our column size = 3, and we surrounded each row with curly braces ({}). It is always good practice to use the curly braces to separate the rows.

We can also write

```
int Employees[2][4][ ] = { { {10, 20, 30}, {15, 25, 35}, {22, 44, 66}, {33, 55, 77} },
                           { {1, 2, 3}, {5, 6, 7}, {2, 4, 6}, {3, 5, 7} }

                         };
```

Third Approach for Multi Dimensional Array in C

```
int Employees[2][4][3] = { { { 10 }, {15, 25}, {22, 44, 66}, {33, 55, 77} },

                           { {1, 2, 3}, {5, 6, 7}, {2, 4, 6}, {3, 5, 7} }

                         };
```

Here, we declared Employees array with row size = 4 and column size = 3. But we only assigned 1 column in the 1st row and 2 columns in the 2nd row of the first table. In these situations, the remaining values will assign to default values(0inthiscase).

Accessing Multi Dimensional Array in C

We can access the C Multi Dimensional array elements using indexes. Index starts at 0 and ends at n-1, where n is the size of a row or column.
For example, if an Array_name[4][8][5] will store 8-row elements and 5 column elements in each table where table size = 4. To access 1st value of the 1st table, use Array_name[0][0][0], to access 2nd row 3rd column value of the 3rd table then use Array_name[2][1][2] and to access the 8th row 5th column of the last table (4th table), use Array_name[3][7][4]. Lets see the example of C Multi Dimensional Array for better understanding:

```
int Employees[2][4][3] = { {10, 20, 30}, {15, 25, 35}, {22, 44, 66}, {33, 55, 77} },
                           { {1, 2, 3}, {5, 6, 7}, {2, 4, 6}, {3, 5, 7} }
                         };


//To Access the values in the Employees[2][4][3] array
//Accessing First Table Rows & Columns

Printf("%d", Employees[0][0][0]) = 10

Printf("%d", Employees[0][0][1]) = 20
```

Printf("%d", Employees[0][0][2]) = 30

Printf("%d", Employees[0][1][0]) = 15

Printf("%d", Employees[0][1][1]) = 25

Printf("%d", Employees[0][1][2]) = 35

Printf("%d", Employees[0][2][0]) = 22

Printf("%d", Employees[0][2][1]) = 44

Printf("%d", Employees[0][2][2]) = 66

Printf("%d", Employees[0][3][0]) = 33

Printf("%d", Employees[0][3][1]) = 55

Printf("%d", Employees[0][3][2]) = 77

//Accessing Second Table Rows & Columns
Printf("%d", Employees[1][0][0]) = 1

Printf("%d", Employees[1][0][1]) = 2

Printf("%d", Employees[1][0][2]) = 3

Printf("%d", Employees[1][1][0]) = 5

Printf("%d", Employees[1][1][1]) = 6

Printf("%d", Employees[1][1][2]) = 7

Printf("%d", Employees[1][2][0]) = 2

Printf("%d", Employees[1][2][1]) = 4

Printf("%d", Employees[1][2][2]) = 6

Printf("%d", Employees[1][3][0]) = 3

Printf("%d", Employees[1][3][1]) = 5

Printf("%d", Employees[1][3][2]) = 7