UNIT- 2

LOOP CONTROL STRUCTURE

Loops

The versatility of the computer lies in its ability to perform a set of instructions repeatedly. This involves repeating some portion ofthe program either a specified number of times or until a particular condition is being satisfied. This repetitive operation is done through a loop control instruction.

There are three methods by way of which we can repeat a part of a program. They are:

(a) Using a **for** statement
(b) Using a **while** statement
(c) Using a **do-while** statement

The *while* Loop

A **while** loop in C programming repeatedly executes a target statement as long as a given condition is true.

Syntax:

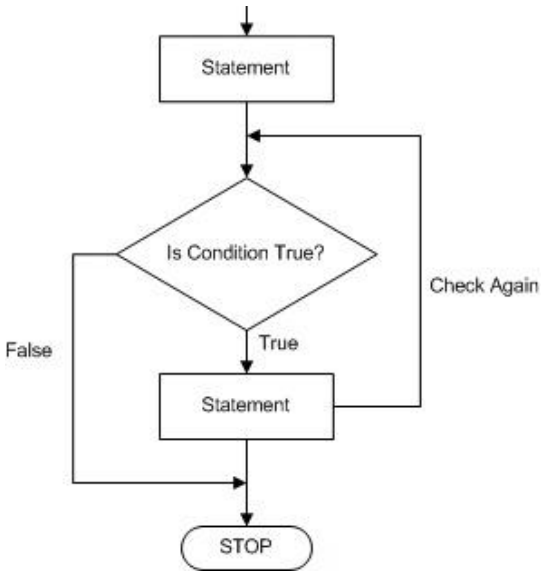
The syntax of a **while** loop in C programming language is

```
while(condition)
{
  statements;
}
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any nonzero value. The loop iterates while the condition is true.

When the condition becomes false, the program control passes to the line immediately following the loop.

Flow Diagram:



Here, the key point to note is that a while loop might not execute at all. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example:

```
#include <stdio.h>

int main ()
{
    int a = 10;
  while( a < 20 )
{
    printf("value of a: %d\n", a);
    a++;
  }
  return 0;
}
```

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

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

The for loop

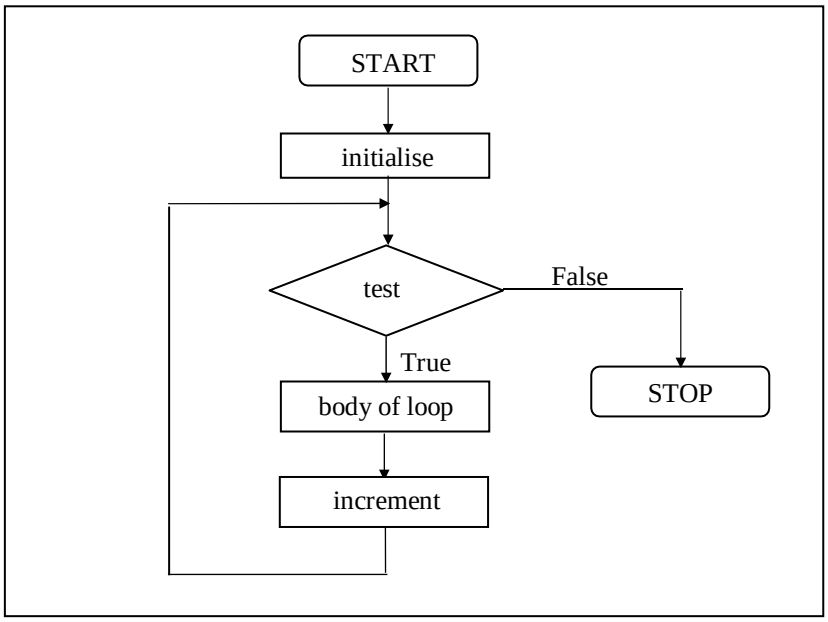
A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The syntax of a **for** loop in C programming language is –

```
for ( initialize ; condition; increment )
{
  statements;
}
```

Flow Diagram



- Initialize - Setting a loop counter to an initial value.
- Condition - Testing the loop counter to determine whether its value has reached the number of repetitions desired.
- Increment - Increasing the value of loop counter each time the program segment within the loop has been executed.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

increment operator: a=a+1; is same as (a++;)

Example

```
#include <stdio.h>
 int main ()
{
  int a;
  for( a = 10; a < 20; a ++)
{
    printf("value of a: %d\n", a);
  }
   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: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Nesting of Loops

The way **if** statements can be nested, similarly **while**s and **for**s can also be nested.

Syntax: for loop

```
for ( initialize ; condition; increment )
{
for ( initialize ; condition; increment )
{
statement(s);
}
}
```

Syntax: while loop

```
while(condition)
{
while(condition)
{
        statement(s);
        }
}
```

The Odd Loop(do while):

Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop in C programming checks its condition at the bottom of the loop.

A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax:

The syntax of a **do...while** loop in C programming language is −

```
do
{
  statements;
} while( condition );
```

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

Flow Diagram

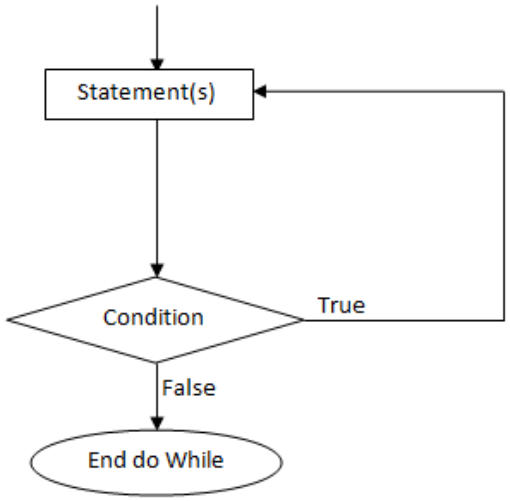


fig: Flowchart for do-while loop

```
int main () {

   int a = 10;

  do {
    printf("value of a: %d\n", a);
    a = a + 1;
  }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: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

The *break* Statement

We often come across situations where we want to jump out of a loop instantly, without waiting to get back to the conditional test. The keyword **break** allows us to do this. When **break** is encountered inside any loop, control automatically passes to the first statement after the loop. A **break** is usually associated with an **if**

Syntax

jump-statement:
break ;

Within nested statements, the **break** statement terminates only the **do**, **for**, **switch**, or **while** statement that immediately encloses it.

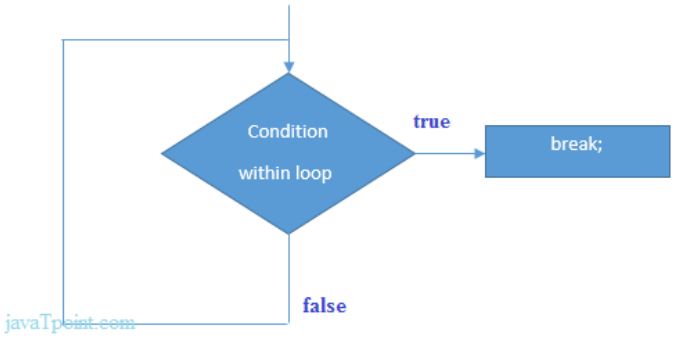


Figure: Flowchart of break statement

Ex:

```
main( )
{
    int  i = 1 , j = 1 ;

    while ( i++ <= 100 )
    {
        while ( j++ <= 200 )
        {
            if ( j == 150 )
                break ;
                else
                printf("%d %d /n", i,j);
            }
      }
  }
```

In this program when **j** equals 150, **break** takes the control outsidethe inner **while** only, since it is placed inside the inner **while**.

The *continue* Statement

In some programming situations we want to take the control to the beginning of the loop, bypassing the statements inside the loop, which have not yet been executed. The keyword **continue** allows us to do this. When **continue** is encountered inside any loop,control automatically passes to the beginning of the loop.

```
main( )
{
    int i, j ;

    for ( i = 1 ; i <= 2 ; i++ )
    {
        for ( j = 1 ; j <= 2 ; j++ )
        {
            if ( i == j )
                continue ;


            printf ( "\n%d %d\n", i, j ) ;
        }
    }
}
```

The output of the above program would be...

```
1 2
2 1
```

The *do-while* Loop

The **do-while** loop looks like this:

```
do
{
    this ;
    and this ;
    and this ;
    and this ;
} while ( this condition is true ) ;
```

There is a minor difference between the working of **while** and **do- while** loops. This difference is the place where the condition is tested. The **while** tests the condition before executing any of the statements within the **while** loop. As against this, the **do-while**tests the condition after having executed the statements within the loop.

The Case Control Structure Decisions Using *switch*

The control statement that allows us to make a decision from the number of choices is called a **switch**, or more correctly a **switch- case-default**, since these three keywords go together to make up the control statement. They most often appear as follows: syntax:

```
switch ( integer expression )
{
    case constant 1 :
    do this ;
    case constant 2 :
    do this ;
    case constant 3 :
    do this ;
    default :
        do this ;
}
```

The integer expression following the keyword **switch** is any C expression that will yield an integer value. It could be an integer constant like 1, 2 or 3, or an expression that evaluates to an integer. The keyword **case** is followed by an integer or a character constant. Each constant in each **case** must be different from all the others. The "do this" lines in the above form of **switch** represent any valid C statement.

Ex:

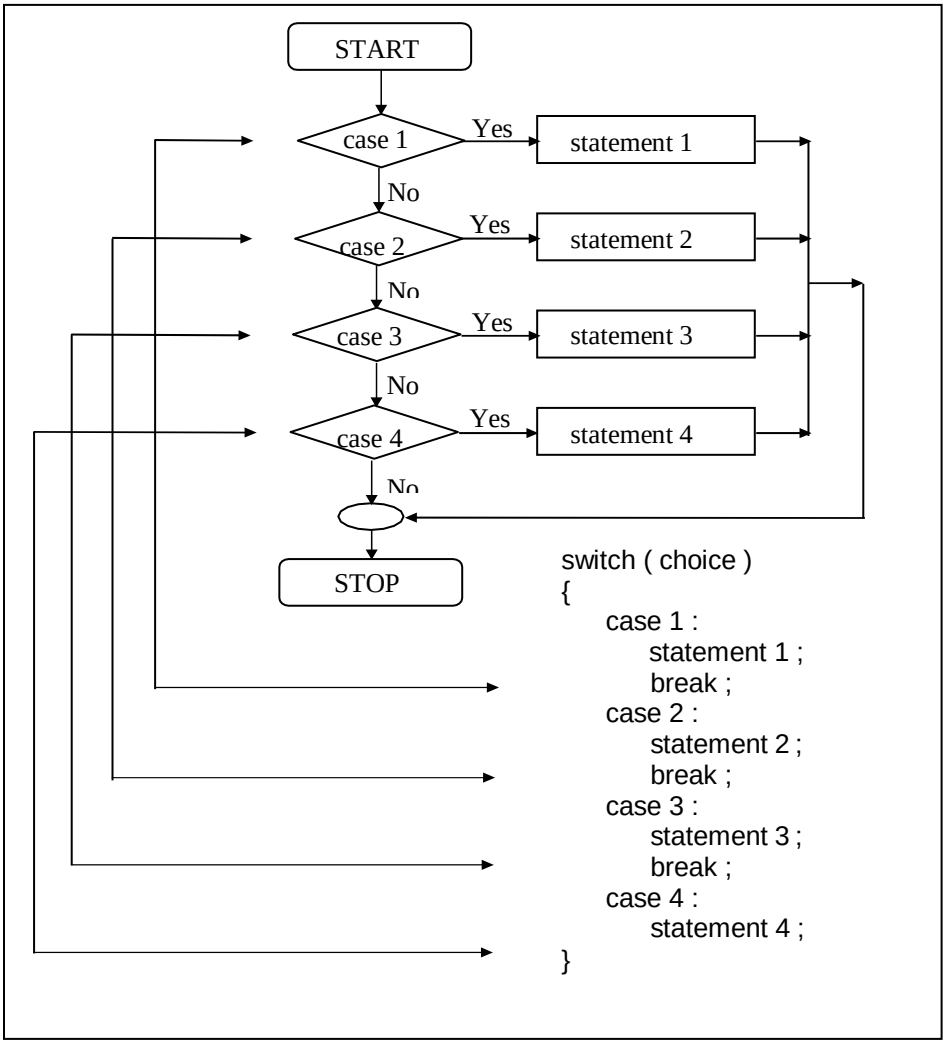
```
main( )
{
    int  i = 2 ;

    switch ( i )
    {
        case 1 :
            printf ( "I am in case 1 \n" ) ;
        case 2 :
            printf ( "I am in case 2 \n" ) ;
        case 3 :
            printf ( "I am in case 3 \n" ) ;
        default :
            printf ( "I am in default \n" ) ;
    }
}
```

The output of this program would be:

```
I am in case 2
```

Flow chart:



switch Versus *if-else* Ladder

There are some things that you simply cannot do with a **switch**.These are:

(a) A float expression cannot be tested using a **switch**

(b) Cases can never have variable expressions (for example it iswrong to say **case a +3 :**)

(c) Multiple cases cannot use same expressions. Thus thefollowing **switch** is illegal:

```
switch ( a )
{
case 3 :
...
case 1 + 2 :
...
}
```

If on the other hand the conditions in the **if-else** were simple and less in number then **if-else** would work out faster than the lookup mechanism of a **switch**.
Hence a **switch** with two **case**s would work slower than an equivalent **if-else**.
Thus, you as a programmershould take a decision which of the two should be used when.

The *goto* Keyword:

The goto statement is a jump statement which is sometimes also referred to as unconditional jump statement.

The goto statement can be used to jump from anywhere to anywhere within a function.