The loop control structure

 # Loops:

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

There are three methods in **C** language by way of which we can repeat a part of the program. These are:

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

b) The ***while*** loop: ⇒

The general form of while is:

```
initial loop counter;
while (test loop counter using condition)
{
        do this;
        and this;
        increment loop counter;
}
```

The statements with in the ***while*** loop would keep on getting executed till the condition being tested remains true. When the condition becomes false, the control bypasses the body of the ***while*** loop.

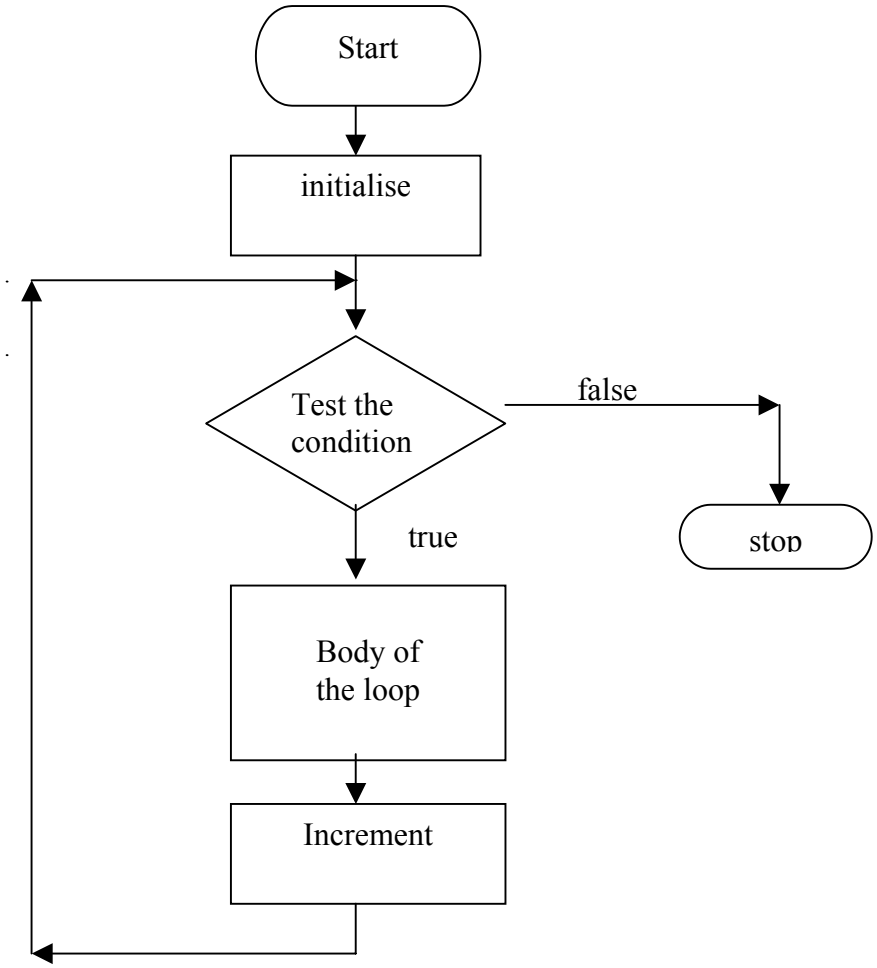
e.g.,
```
c = 1;
while (c<=3)
{
        x = x + c;
        c = c + 1;
}
```

Example program
File name: EP8.C

```
/* Program to demonstrate the while loop*/
/* This program will add first 10 natural numbers */
main( )
{
        int i, s;
        i = 1;
        s = 0;
        while (i<=10)
        {
                s = s + i;
                i = i + 1;
        }
        printf ("The sum of first 10 natural numbers = %d",s);
}
```

Output:
The sum of first 10 natural numbers = 55

The following flowchart explains the working of ***while*** loop as well as the ***for*** loop:



The condition being tested may use relational or logical operators as shown in the following example:

```
while(a<=10)
while(a>=10 && b<=15)
while(a>10 && (b<15 ‖ c<14))        etc.
```

The statement with in the loop may be a single line or a block of statements. In the first case parentheses are optional whereas for 2nd case parentheses must be used.

As a rule the ***while*** must test a condition that will eventually become false otherwise the loop would be executed forever, indefinitely.

e.g.,
```
main( )
{
        int a=1;
        while (a<=10)
        {
                printf (“ASD”);
        }
}
```

this is an indefinite loop, since a remains equal to 1 forever. The correct form will be,

.,
```
main( )
{
        int a=1;
        while (a<=10)
        {
                printf (“ASD”);
                a = a+1;
        }
}
```

Instead of incrementing a loop counter, we can even decrement it.

e.g.,
```
main( )
{
        int a = 5;
        while (a >= 1)
        {
                printf (“%d\n”,a);
                a = a – 1;
        }
}
```

(a) The ***for*** loop: ⇒

C provides another looping structure which is more general. This structure is realized by a statement called the ***for*** statement. The general form of ***for*** statement is:

```
for (initialize the counter; test counter; increment the counter)
{
        do this;
        and this;
        and this;
}
```

Example program-09
Filename: EP9.C

```
/* Demonstration of for loop */
main( )
{
        int i, s;
        s = 0;
        for ( i=1; i<=10; i++)          /* i=i+1 ⇔ i++ ⇔ i+ = 1 */
                s=s+i;
        printf ("The sum of first 10 natural number = %d",s);
}
```

Output:
The sum of first 10 natural number = 55

Various techniques to write ***for*** loop:

Let us write a program to print the numbers 1 to 10 in different ways using ***for*** loop:

Example program-10
Filename: EP10.C

1.
```
main( )
{
        int a;
        for (a=1;a<=10;a=a+1)
                printf("%d\n",a);
}
```

2.
```
main( )
{
        int a;
        for (a=1;a<=10;)
        {
                printf("%d\n",a);
                a=a+1;
        }
}
```

3.
```
main( )
{
        int a=1;
        for (;a<=10;a=a+1)
                printf("%d\n",a);
}
```

4.
```
main( );
{
        int a=1;
        for (; a<=10;)
        {
                printf("%d\n",a);
                a=a+1;
        }
}
```

5.
```
main( )
int a;
{
        for (a=0;a++<10;)
                printf("%d\n",a);
}
```

6.
```
main( )
{
        int a:
        for(a=0;++a<=10;)
                printf("%d\n",a);
}
```

N.B. In example 5 since '++' precedes **a** so firstly comparison is done, followed by incrimination and hence it is necessary to initialize **a** to 0. Whereas in 6 firstly incrimination is done followed by comparison.

Nesting of loops:

The way if statement can be nested, similarly ***while***s and ***for***s are also nested:

Syntax:

Nesting of ***for***:

```
for(initialize counter; test counter; increment counter)
{
        for(initialize counter; test counter; increment counter)
        {
                do this;
                and this;
        }
}
```

Nesting of ***while***:

```
while(test loop counter using condition)
{
        while(test loop counter using condition)
        {
                do this;
                and this;
                increment the loop counter;
        }
        increment the loop counter;
}
```

The ***break*** statement:

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

The ***continue*** statement:

In some situations we want to take the control to the beginning of the loop, bypassing the statements inside the loop, which have not been yet executed. The key word ***continue*** allows us to do this in **C**. When the key word ***continue*** is encountered inside any C loop, control automatically passes at the beginning of the loop. A ***continue*** is usually associated with an ***if***.

The following two programs demonstrate the use of ***continue*** and ***break*** statements:

Example program-11
Filename:EP11.C

```
/* Program to demonstrate the break statement */
/* This program will check whether a number is prime or not*/
main( )
{
        int num, a;
        printf("Enter any number");
        scanf("%d",&num);
        a=2;
        while(a<=num-1)
        {
                if(num%a= =0)
                {
                        printf("Not a prime number");
                        break;
                }
                a++;
        }
        if(a= = num)
                printf("Prime number");
}
```

Output:

```
Enter a number
11 ( supplied by key board)
Prime number
```

Example program-12
File name: EP12.C

```
/* Program to demonstrate continue statement */
main( )
{
        int i, j;
        for(i=1;i<=2;i++)
        {
                for(j=1;j<=2;j++)
                {
                        if(i= =j)
                                continue;
                        printf("\n%d\t%d",i,j);
                }
        }
}
```

Output:

```
1       2
2       1
```

(c) The ***do-while*** loop:

The general form of ***do-while*** loop is:

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

The ***do-while*** loop is very similar to the ***while*** loop except the place where the condition is tested. Here the statements with in the body will be executed first and then repeat if the condition is found to be true. Whereas in ***while*** loop first check the condition and then execute the statements with in the body of loop, i.e., in ***do-while*** loop the statements will be executed once even if the condition fails where as in ***while*** loop if condition fails no execution of statements will occur.

Apart from this peculiarity of the ***do-while*** loop the ***while*** and ***do-while*** loop behave exactly identically. However, ***do-while*** loops are rarely used in **C**-program, since there are comparatively fewer occasions when we want to execute a loop at least once no matter what the condition be.