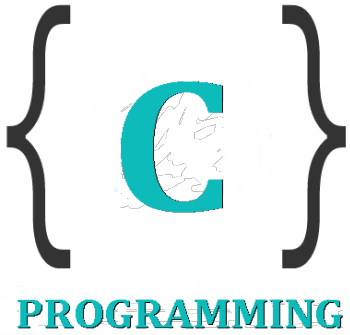
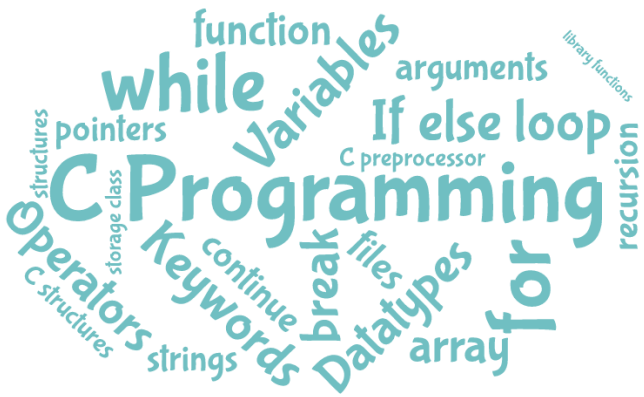


IT 112: Introduction to Programming



Dr. Manish Khare
Dr. Bakul Gohel

Loop Control Instruction

- If you wish to find averages of 100 sets of three numbers, would you actually execute the program 100 times?
- Obviously Not
- There has to be a better way out.
- And that is Looping within program

Loops

- The versatility of the computer lies in its ability to perform a set of instructions repeatedly.
- This involves repeating some portion of the program **either a specified number of times** or **until a particular condition is being satisfied**.
- There are three ways to repeat a part of a program.
- They are
  - Using a while statement
  - Using a for statement
  - Using a do-while statement

The while Loop

- It is often case in programming that you want to repeat something a fixed number of times.

- For ex. You want to calculate gross salaries of ten different persons, or you want to convert temperatures from centigrade to Fahrenheit for 15 different cities.

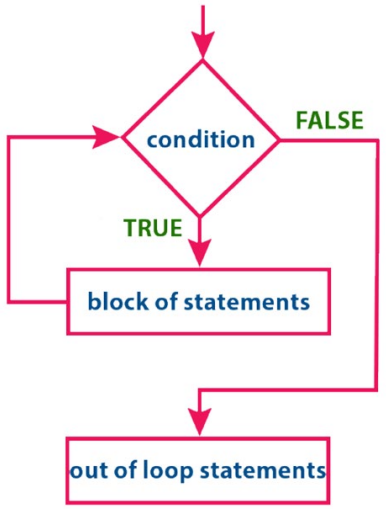
- The while loop is ideally suited for this

- The general form of while loop is

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

Syntax:

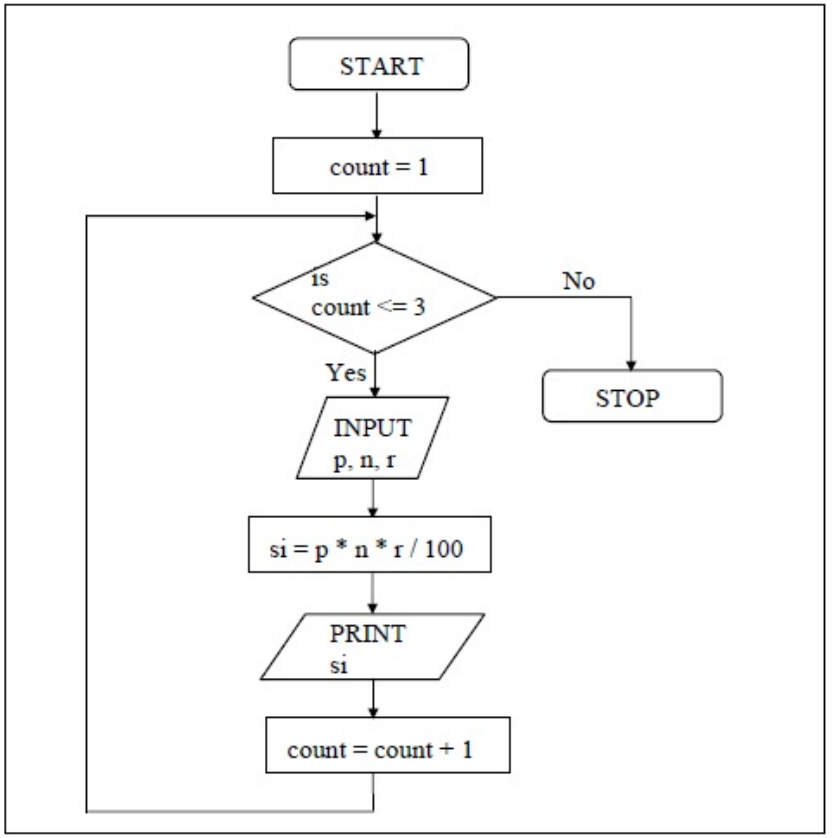
```
while( condition )
{
      ...
      block of statements;
      ...
}
```



Let us a look that uses a while loop to calculate simple interest for 3 sets of values of principal, number of years and rate of interest

/* Calculation of simple interest for 3 sets of p, n and r */

```
int main( )
{
int p, n, count ;
float r, si ;
count =1;
while ( count <= 3 )
{
   printf ( "\nEnter values of p, n and r " ) ;
   scanf ("%d %d %f", &p, &n, &r ) ;
   si = p * n * r / 100 ;
   printf ( "Simple interest = Rs. %f", si ) ;
   count = count + 1;
}
return 0;
}
```



Some Trips on while loop

- Note following point about while
  - The statement within the while loop would keep getting executed till the condition being tested remains true. When the condition becomes false, the control passes to the first statement that follows the body of the while loop.
  - In place of the condition, there can be any other valid expression. So long as the expression evaluates to a non-zero value, the statements within the loop would get executed.

- The condition being tested may use relational or logical operators as shown in the following example.
  - while (i <= 10)
  - while (i >= 10 && j <= 15)
  - while (j > 10 && (b<15 || c<20))

- The statements within the loop may consist a single line or block of statement. In the first case (single line within loop), the braces are optional, but in the second case (block of statement), the braces are mandatory.

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

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

- This is an indefinite loop, since i remains equal to 1 forever.
- The correct form would be as under:

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

- Instead of incrementing a loop counter, we can even decrement it and still manage to get the body of the loop executed repeatedly.

```
main()
{
    int i=5;
    while(i>=1)
        {
          printf("\n Make the computer literate! ");
          i=i-1;
        }
}
```

- It is not necessary that a loop counter must only be an int. It can even be a float

```
main( )
{
    float a = 10.0 ;
    while (a<=10.5)
    {
        printf ( "\nRaindrops on roses…");
        printf ( "...and whiskers on kittens");
        a = a + 0.1 ;
    }
}
```

- What do you think would be the output of the following program?

```
main( )
{
    int i;
    while (i<=32767)
    {
        printf ("%d\n",i);
        i = i + 1 ;
    }
}
```

- What will be the output of the following program?

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

- This indefinite loop, and it doesn't give any output at all. The reason is, we have carelessly given a ";" after the while. This would make the loop work like this...

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

- Since the value of i is not getting incremented the control would keep rotating within the loop, eternally.
- Note that enclosing printf() and i=i+1 within a pair of braces is not an error.
- In fact we can put a pair of braces around any individual statement or set of statements without affecting the execution of the program.

More Operators

- There are variety of operators which are frequently used with while.
- To illustrate their usage let us consider a problem wherein numbers from 1 to 10 are to be printed on the screen.
- The program for performing this task can be written using while in the following different ways:

Way 1

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

Way 2

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

- Note that the **increment operator** ++ increments the value of i by 1, every time the statement i++ gets executed.

- Similarly to reduce the value of a variable by 1 a decrement operator -- is also available.

Way 3

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

➢Note that += is a **compound assignment operator**. It increments the value of i by 1. Similarly, j = j + 10 can also be written as j += 10. Other compound assignment operators -=, *=, / = and %=.

Way 4

```
main( )
{
    int i = 0 ;
    while (i++<10)
    {
        printf ("%d\n",i);
    }
}
```

- In the statement while (i++<10), firstly the comparison od value of i with 10 is performed, and then the incrementation of i takes place.
- Since the incrementation of i happens after its usage, here the ++ operator is called a **post incrementation** operator.
- When the control reaches printf(), i has already been incremented, hence i must be initialized to 0.

Way 5

```
main( )
{
    int i = 0 ;
    while (++i<10)
    {
        printf ("%d\n",i);
    }
}
```

- In the statement while (i++<10), firstly incrementation of i takes place, then the comparison of value of i with 10 is performed.

- Since the incrementation of i happens before its usage, here the ++ operator is called a **pre-incrementation** operator.

Exercise

- What will be the output of the following program?

```
main( )
{
    int x = 4, y, z ;
    y = --x ;
    z = x-- ;
    printf ( "\n%d %d %d", x, y, z ) ;
}
```

Practice

- Write a program to find the factorial value of any number entered through the keyboard.

Practice

- Write a program to enter the numbers till the user wants and at the end it should display the count of positive, negative and zeros entered.

for Loop

- The for loop allows us to specify things about the loop in a single line:
  - Setting a loop counter to an initial value.
  - Testing the loop counter to determine whether its value has reached the number of repetitions desired.
  - Increasing the value of loop counter each time the body of the loop has been executed.

- The general form of for statement is as under:

for (**initialise counter** ; **test counter** ; **increment counter**)

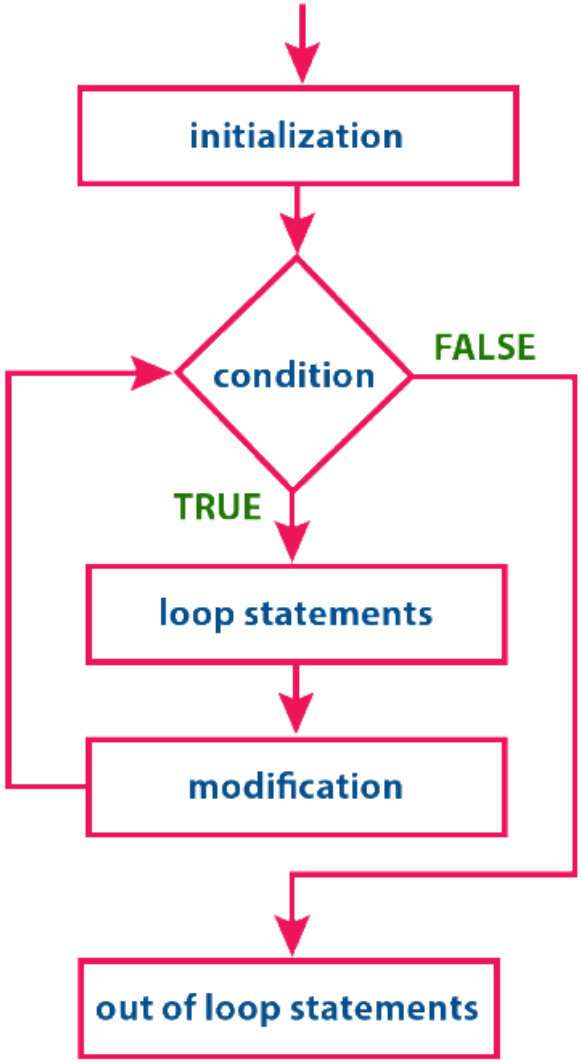
{

do this ;

and this ;

and this;

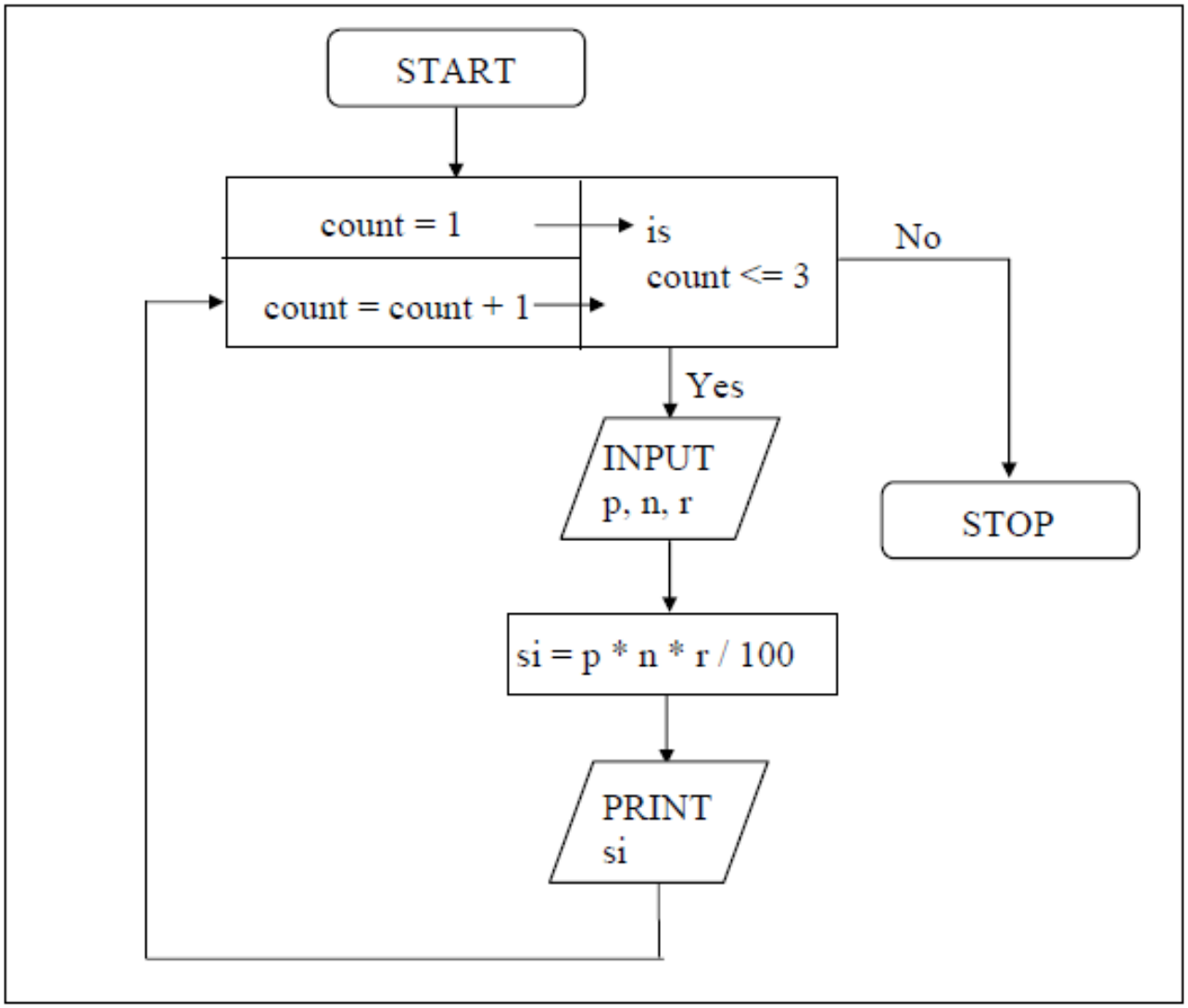
}


initialization
condition
FALSE
TRUE
loop statements
modification
out of loop statements

Let us now write down the simple interest program using for.

```
main( )
{
int p, n;
float r, si ;
for (int count=1; count <= 3; count++)
{
    printf ( "\nEnter values of p, n and r " ) ;
    scanf (“%d %d %f", &p, &n, &r ) ;
    si = n * r / 100 ;
    printf ( "Simple interest = Rs. %f", si ) ;
}
}
```

```
main( )
{
int p, n, count ;
float r, si ;
count =1;
while ( count <= 3 )
{
printf ( "\nEnter values of p, n and r " ) ;
scanf (“%d %d %f", &p, &n, &r ) ;
si = n * r / 100 ;
printf ( "Simple interest = Rs. %f", si ) ;
count++;
}
}
```


START
count = 1
count = count + 1
is
count <= 3
No
Yes
INPUT
p, n, r
STOP
si = p * n * r / 100
PRINT
si

How for statement gets executed

- When the for statement is executed for the first time, the value of count is set to an initial value 1.
- Now the condition count <= 3 is tested. Since count is 1 the condition is satisfied and the body of the loop is executed for the first time.
- Upon reaching the closing brace of for, control is sent back to the for statement, where the value of count gets incremented by 1.
- Again the test is performed to check whether the new value of count exceeds 3.
- If the value of count is still within the range 1 to 3, the statements within the braces of for are executed again.
- The body of the for loop continues to get executed till count doesn't exceed the final value 3.
- When count reaches the value 4 the control exits from the loop and is transferred to the statement (if any) immediately after the body of for.

- It is important to note that the initialization, testing and incrementation part of a for loop can be replaced by any valid expression.
- Thus the following for loops are perfectly ok.
- for (i = 10 ; i ; i --)
  - printf ("%d", i) ;
- for (i < 4 ; j = 5 ; j = 0)
  - printf ("%d", i) ;
- for (i = 1; i <=10 ; printf ("%d",i++) ;
- for (scanf ("%d", &i) ; i <= 10 ; i++)
  - printf ("%d", i) ;

- Let us now write down the program to print numbers from 1 to 10 in different ways.

- This time we would use a for loop instead of a while loop.

Way 1

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

- Note that the initialisation, testing and incrementation of loop counter is done in the for statement itself. Instead of i = i + 1, the statements i++ or i += 1 can also be used.

- Since there is only one statement in the body of the for loop, the pair of braces have been dropped. As with the while, the default scope of for is the immediately next statement after for.

Way 2

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

- Here, the incrementation is done within the body of the for loop and not in the for statement. Note that in spite of this the semicolon after the condition is necessary.

Way 3

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

- Here the initialisation is done in the declaration statement itself, but still the semicolon before the condition is necessary.

Way 4

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

➢Here, neither the initialisation, nor the incrementation is done in the for statement, but still the two semicolons are necessary.

Way 5

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

- Here, the comparison as well as the incrementation is done through the same statement, i++ < 10.
- Since the ++ operator comes after i firstly comparison is done, followed by incrementation. Note that it is necessary to initialize i to 0.

Way 6

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

- Here, both, the comparison and the incrementation is done through the same statement, ++i <= 10. Since ++ precedes i firstly incrementation is done, followed by comparison. Note that it is necessary to initialize i to 0.

Nesting of Loops

- The way if statements can be nested, similarly whiles and fors can also be nested. To understand how nested loops work, look at the program given below:

```
/* Demonstration of nested loops */
main( )
{
    int r, c, sum ;
    for ( r = 1 ; r <= 3 ; r++ ) /* outer loop */
    {
        for ( c = 1 ; c <= 2 ; c++ ) /* inner loop */
        {
            sum = r + c ;
            printf ( "r = %d c = %d sum = %d\n", r, c, sum ) ;
        }
    }
}
```

- When you run this program you will get the following output:

r = 1 c = 1 sum = 2

r = 1 c = 2 sum = 3

r = 2 c = 1 sum = 3

r = 2 c = 2 sum = 4

r = 3 c = 1 sum = 4

r = 3 c = 2 sum = 5

- Here, for each value of r the inner loop is cycled through twice, with the variable c taking values from 1 to 2. The inner loop terminates when the value of c exceeds 2, and the outer loop terminates when the value of r exceeds 3.

Multiple Initialisations in the for Loop

- The initialisation expression of the for loop can contain more than one statement separated by a comma.
- For example,
  - for (i = 1, j = 2 ; j <= 10 ; j++)
- Multiple statements can also be used in the incrementation expression of for loop; i.e., you can increment (or decrement) two or more variables at the same time.

- However, only one expression is allowed in the test expression. This expression may contain several conditions linked together using logical operators.

- Use of multiple statements in the initialisation expression also demonstrates why semicolons are used to separate the three expressions in the for loop.

- If commas had been used, they could not also have been used to separate multiple statements in the initialisation expression, without confusing the compiler.

do-while Loop

- The do-while loop looks like this:

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

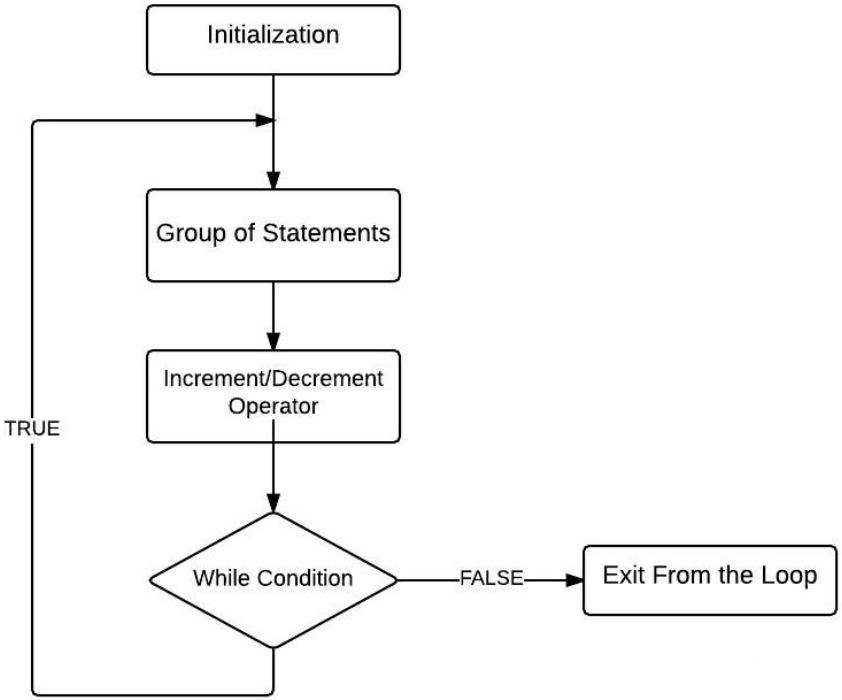
do-while

```
// while example
// Enter the password (in integer)
// if incorrect, programme ask again

#include <stdio.h>
   int main()
   {
     int password=0, key=4321;

     while(password!=key)
     {
        printf("\nEnter The password:" );
        scanf("%d",&password);
        if(password!=key)
        {
           printf("\nError - invalid password" );
        }
        else
        {
            printf("\npassword matched" );
        }
     }
     return 0;
   }
```

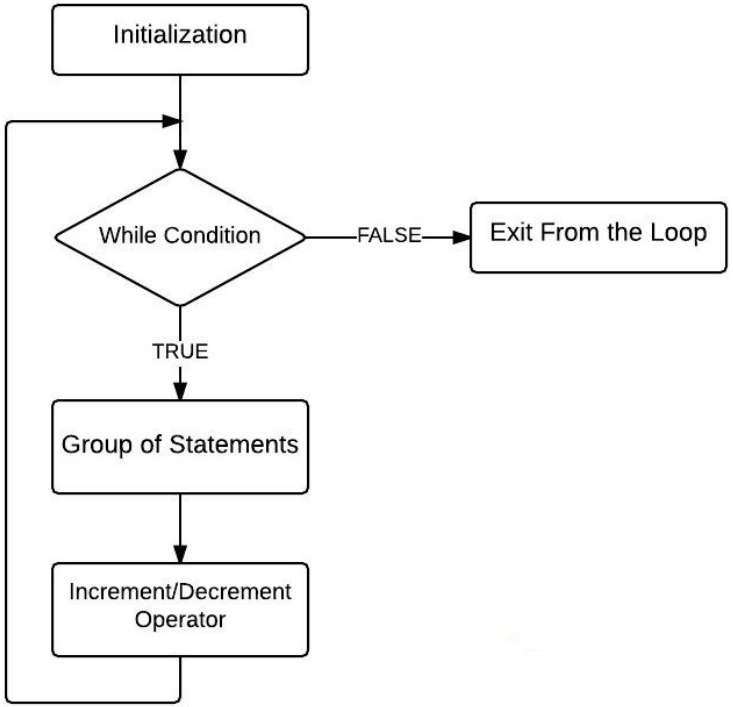
```
// do while example
// Enter the password (in integer)
// if incorrect, programme ask again

#include <stdio.h>
   int main()
   {
     int password, key=4321;

     do
     {
        printf("\nEnter The password:" );
        scanf("%d",&password);
        if(password!=key)
        {
           printf("\nError - invalid password" );
        }
        else
        {
            printf("\npassword matched" );
        }
     } while(password!=key);

     return 0;
   }
```

- 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

do..while



while



do-while

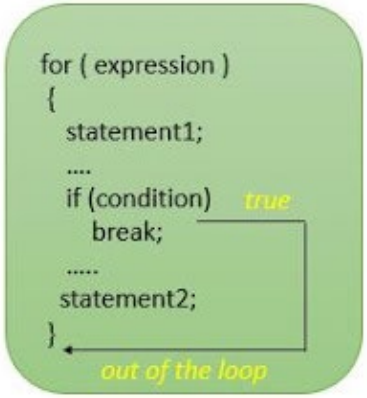
- This means that do-while would execute its statements at least once, even if the condition fails for the first time.

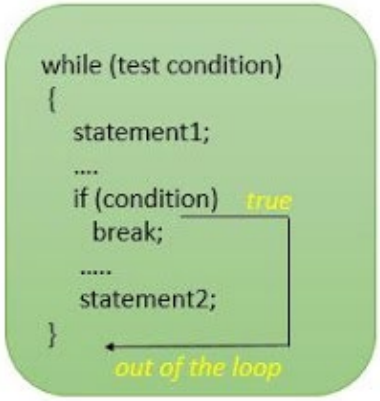
- The while, on the other hand will not execute its statements if the condition fails for the first time.

- This difference is brought about more clearly by the following program

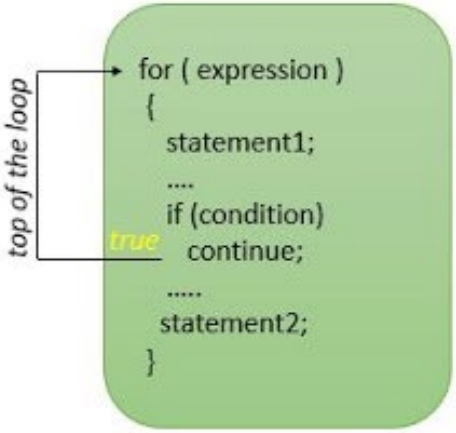
break and continue

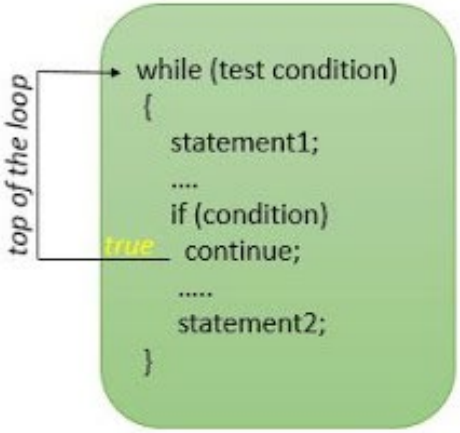
break





continue





break

// **break example using while loop**
// Enter the password (in integer)
// if incorrect, programme ask again

```
#include <stdio.h>
   int main()
   {
     int password=0, key=4321;

     while(1)
     {
        printf("\nEnter The password:" );
        scanf("%d",&password);
        if(password!=key)
        {
           printf("\nError - invalid password" );
        }
        else
        {
           printf("\npassword matched" );
           break;
        }
     }
     return 0;
   }
```

// **break example using for loop**
// Enter the password (in integer)
// if incorrect, programme ask again

```
#include <stdio.h>
   int main()
   {
     int password=0, key=4321;
     for(int i=1 ; i<=3 ; ++i)
     {
        printf("\nEnter The password:" );
        scanf("%d",&password);
        if(password!=key)
        {
           printf("\nError - invalid password" );
           if(i==3)
           {
              printf("\nNumber of chance exhausted, try after few minutes\n" );
           }
        }
        else
        {
           printf("\npassword matched" );
           break;
        }
     }
     return 0;
   }
```

break statement

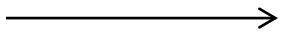
- We often come across situations where we want to jump out of a loop instantly, without waiting to get back to the conditional test.
- The keyword **break** allows us to do this. When break is encountered inside any loop, control automatically passes to the first statement after the loop.
- A break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

https://ideone.com/610SyM

https://ideone.com/gp8653

continue

```
// Program to calculate the sum of numbers (10
numbers max)
// If the user enters a negative number, it's not added
to the result

#include <stdio.h>
int main() {

   int number, sum = 0;

   for (int i = 1; i <= 5; ++i) {
      printf("Enter a n%d: ", i);
      scanf("%d", &number);

      if (number < 0) {
         continue;
      }

      sum += number;
   }

   printf("Sum = %d", sum);

   return 0;
}
```



Without Continue

```
// Program to calculate the sum of numbers (10
numbers max)
// If the user enters a negative number, it's not added
to the result

#include <stdio.h>
int main() {

   int number, sum = 0;

   for (int i = 1; i <= 5; ++i) {
      printf("Enter a n%d: ", i);
      scanf("%d", &number);

      if (number > 0) {
         sum += number;
      }

}

   printf("Sum = %d", sum);

   return 0;
}
```

Continue Statement

- In some programming situations we want to take the control to the beginning of the loop, by passing 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.

https://ideone.com/bc7nwj

Guess the output

- What would be the output of the following programs:

```
main( )
{
        int i = 0 ;
        for ( ; --i ; )
                printf ( "\nHere is some mail for you" ) ;
}

////////////////////////////////////////////////////////////////////////////////////////////////////
main( )
{
        int i = 0 ;
        for ( ; i-- ; )
                printf ( "\nHere is some mail for you" ) ;
}
```

Guess the output

```
main( )
{
int i = -5 ;
for ( i = 1 ; i <= 5 ; printf ( "\n%d", i++ ) ) ;
{
        printf("output: %d\n", i+10);
}
}
```

Guess the output

```
main( )
{
    int i = 1, j = 1 ;
    for ( ; ; )
    {
        if ( i > 5 )
          break ;
        else
          j += i ;
        printf ( "\n%d", j ) ;
        i += j ;
    }
}
```