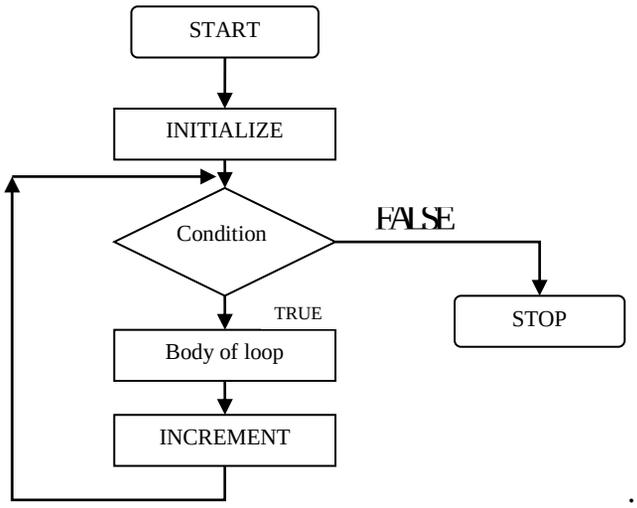
Loops

Loops are used to repeat a block of code. The concept of true and false in C should be clear before loops are discussed, because it will be necessary when working with loops.
(0 is FALSE rest everything is TRUE). There are three types of loops: for, while, (and) do while. Each of them has their specific uses. They are all outlined below.

FOR

for loops are the most useful type. The layout is
for(variable initialization; conditional expression; modification of variable)
{
  repeat these statements
}

The variable initialization allows you to either declare a variable and give it a value or give a value to an already declared variable. Second, the conditional expression tells the program that while the conditional expression is true the loop should continue to repeat itself. The variable modification section should indeed modify a variable otherwise it will it will always check the same values in the condition.



.

Note: STOP doesn't refer to end of program but it only refers to end of loop.

```
int main() {   /*We always need this */
int x;    /*We need an variable for loop*/
          /*The loop goes while x<100, and x increases by one
     every loop */
for(x=0;x<100;x++) {
                        /*Keep in mind that the loop condition
                        checks the x value before it Actually
                        repeats, or loops, again.  So if x==100
                        the loop will end.*/
  printf("\n X is now %d",x); /*write the current value of x*/
} }
```

This program is a very simple example of a for loop. x is set to zero, while x is less than 100 it calls printf("\n X is now %d",x); and it adds 1 to x until the loop ends. Keep in mind also that the variable is incremented after the code in the loop is run for the first time. (See the flow chart)

WHILE

WHILE loops are very simple. The basic structures is...WHILE (TRUE) then execute all the code in the loop. The TRUE represents a boolean expression which could be x==1 or while(x!=7) (x does not equal 7). It can be any combination of boolean statements that are legal. Even, (while x==5 || v==7) which says execute the code while x equals five or while v equals 7

```
int main() {
int x=0;              /*Don't forget to declare variables  */
while (x<100) {      /*While x is less than 100 do  */
     printf("\n X is now %d",x);   /*write the current value of x*/
     x++;           /*Adds 1 to x every time it repeats, in for
                 loops the loop structure allows this to be done
                 in the structure  but not here*/
}}
```

This was another simple example, but it is longer than the above FOR loop. The easiest way to think of the loop is to think the code executes, when it reaches the brace at the end it goes all the way back up to while, which checks the condition.

DO WHILE

DO WHILE loops are useful for only things that want to loop at least once. In while loop, we first the check the condition and if it is FALSE , the code is not executed even single time. In situations we want to guarantee that the code runs atleast one time we use a DO WHILE loop. The structure is

```
do {
      THIS;
      And this ;
} while (condition);
```

```
int main() {
  int x = 0;
  do   {
     printf("\n Hello World);
  }while(x!=0);
}
```

Keep in mind that you must include a trailing semi-colon after while in the above example. Notice that this loop will also execute once, because it automatically executes before checking the truth statement.

Nesting of Loops

C allows one loop to complete lie inside another loop. This is called the nesting of loops. The only condition imposed is that one loop should be completely inside the other.

(1,1)	(1,2)	(1,3)
(2,1)	(2,2)	(2,3)
(3,1)	(3,2)	(3,3)

To move in the matrix above, we need to change the positions as written above, this can be done using nesting of loops.

```
main() {
     int row,col;
     clrscr();
     for(row=1;row<=3;row++) {
          for(col=1;col<=3;col++) {
```

```
                printf("  (%d,%d)",row,col);
            }
        printf("\n");
}}
```

The purpose writing printf("\n"); separately is to move to next line only after all columns of current row are written. i.e.
Write (1,1) (1,2) (1,3) then move to next line. Also observe that inner for loop (dealing with col) is completely inside the outer for loop.

Coming out of the loop – the *break* statement

If you want to come across situation where you want to come out of the loop without completing it, this can be done by a *break* statement. If a *break* keyword is encountered in the loop, control automatically transfers to the first statement after the loop. A break is usually associated with an *if* statement. E.g. A loop is capable of writing integers from 1 to 100, but it asks from the user the maximum integer to write the loop automatically terminates after that number. This can be implemented using break statement.

```
main() {
     int num,count;
     printf("\nEnter the maximum number (less than 100) to stop:");
     scanf("%d",&num);
     for(count=1;count<=100;count++) {
          if(count > num) break;
          printf("\n%d",count);
     }
     printf("\nOut of the loop");
}
```

Once count exceeds num, its time to stop printing numbers. This is done by break statement as it throws control out of the loop the statement at printf("\nOut of the loop");

Going to the top of the loop – the *continue* statement

In some situations we want to move to the beginning of the loop without executing all the statements in the loop. This can be done with the help of *continue* statement. When *continue* is encountered, control automatically passes to the beginning of the loop.

```
main() {
     int row,col;
     clrscr();
     for(row=1;row<=2;row++) {
          for(col=1;col<=2;col++) {
               if(row == col) continue;
               printf("  (%d,%d)",row,col);
          }
     printf("\n");
}}
```

the output of above program will be
1 2
2 1
because when row is equal to col the condition if(row == col) the *continue* statement is executed and the control directly transfers to the for(col=1;col<=3;col++) statement without executing printf(" (%d,%d)",row,col);