C PROGRAMMING – CONTROL FLOW

DR. PAWAN KUMAR SINGH
DEPARTMENT OF INFORMATION TECHNOLOGY
JADAVPUR UNIVERSITY
KOLKATA

LOOPS :

- ***WHY DO WE NEED LOOPS ???***

- There may be a situation, when you need to execute a block of code several number of times.
- In general statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- A loop statement allows us to execute a statement or group of statements multiple times

LOOPS :

- ***TYPES OF LOOPS :***

- WHILE LOOP
- FOR LOOP
- DO-WHILE LOOP
- NESTED LOOP

- ***LETS HAVE A CLOSER LOOK***

LOOPS => WHILE LOOP

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

Syntax:

The syntax of a while loop in C is:

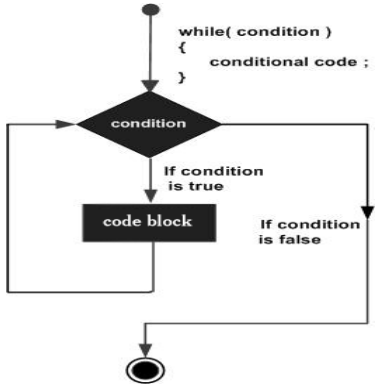
```
while(condition)
{
   statement(s);
}
```

LOOPS => WHILE LOOP

- Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.
- When the condition becomes false, program control passes to the line immediately following the loop

LOOPS => WHILE LOOP

FLOW DIAGRAM



LOOPS => WHILE LOOP

- EXAMPLE :

```
#include<stdio.h>
void main ()
{
   // Local variable declaration:
    int a = 10;
    // while loop execution
    while( a < 20 )
  {
      printf("value of a:%d /n", a);
       a++;
  }
  getch();
}
```


LOOPS => WHILE LOOP

- 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
```

LOOPS :

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 is:

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


LOOPS => FOR LOOP

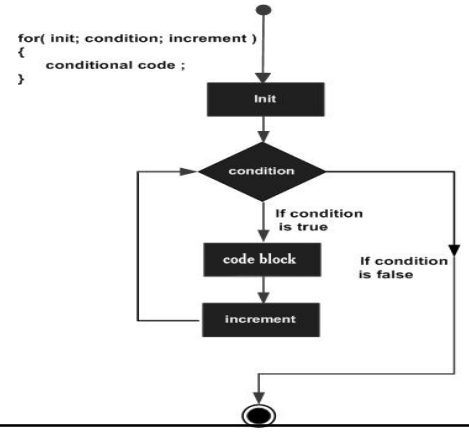
- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.

LOOPS => FOR LOOP

- After the body of the for loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- 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.

LOOPS=> FOR LOOP

- **Flow Diagram:**



LOOPS => FOR LOOP

Example:

```
#include<stdio.h>
void main ()
{   // for loop execution
   for( int a = 10; a < 20; a = a + 1 )
   {
      printf("value of a: %d /n", a);
   }
  getch();
}
```


LOOPS => FOR LOOP

- 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
```


LOOPS=> FOR LOOP

Different Ways of Implementing For Loop

Form	Comment
for (i=0 ; i < 10 ; i++) Statement1;	**Single** Statement
for (i=0 ;i <10; i++) { Statement1; Statement2; Statement3; }	**Multiple** Statements within for
for (i=0 ; i < 10;i++) ;	For Loop with no Body (**Carefully Look at the Semicolon**)
for (i=0,j=0;i<100;i++,j++) Statement1;	**Multiple initialization** & Multiple **Update** Statements Separated by Comma
for (; i<10 ; i++)	**Initialization not used**
for (; i<10 ;)	**Initialization & Update not used**
for (; ;)	**Infinite** Loop,Never Terminates

LOOPS :

- ***DO-WHILE LOOP:***
- Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop checks its condition at the bottom of the loop.
- A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

LOOPS => DO-WHILE LOOP

Syntax:

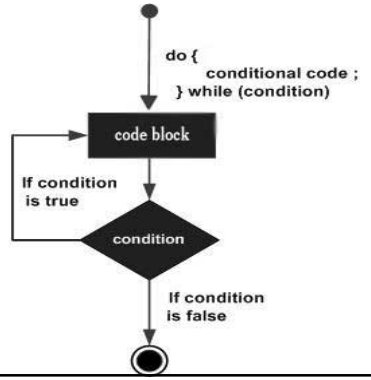
The syntax of a do...while loop in C is:

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

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

LOOPS => DO-WHILE LOOP

- **Flow Diagram:**



LOOPS => DO-WHILE LOOP

- **Example:**

```
#include<stdio.h>
 void main ()
{    // Local variable declaration:
     int a = 10;
     // do loop execution
    do
     {
       printf( "value of a: %d\n " ,a);
       a = a + 1;
     } while( a < 20 );
    getch();
}
```


LOOPS => DO-WHILE LOOP

- 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
```

LOOPS :

- ***NESTED LOOPS :***
- A loop can be nested inside of another loop.
- **Syntax:**
  The syntax for a **nested for loop** statement in C is as follows:

```
for ( init; condition; increment )
  {
     for ( init; condition; increment )
        {
           statement(s);
        }
     statement(s);
     // you can put more statements.
  }
```


LOOPS => NESTED LOOP

- ***EXAMPLE :***

```
#include<stdio.h>
void main ()
{
    int a=1,b;
    while(a<=3)
     {
        for(b=1;b<=3;b++)
         {
            printf("a = %d , b = %d\n",a,b);
         }
        printf("\n");
        a++;
     }
   system("pause");
 }
```


LOOPS => NESTED LOOP

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

```
a = 1 , b = 1
a = 1 , b = 2
a = 1 , b = 3

a = 2 , b = 1
a = 2 , b = 2
a = 2 , b = 3

a = 3 , b = 1
a = 3 , b = 2
a = 3 , b = 3
```


BREAK STATEMENT

- It is an jump instruction and can be used inside the switch and loop statements.
- The execution of the break statements causes the control transfer to the statement immediately after the loop.

BREAK STATEMENT

Break in For Loop :

```
for(initialization ; condition ; increment
ation)
{
Statement1;
Statement2;
break;
}
```

Break in While Loop :

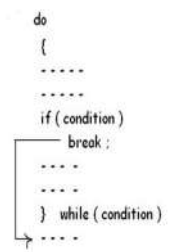
```
initialization ;
while(condition)
{
Statement1;
Statement2;
incrementation
break;
}
```


BREAK STATEMENT

Break Statement in Do-While :

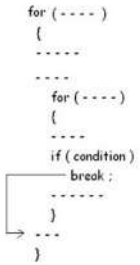
```
initialization ;
do
{
Statement1;
Statement2;
incrementation
break;
}while(condition);
```

Do-While Loop

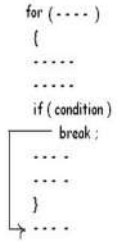


BREAK STATEMENT

Nested for

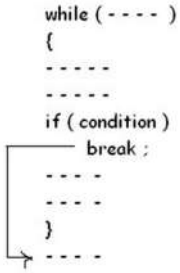


For Loop

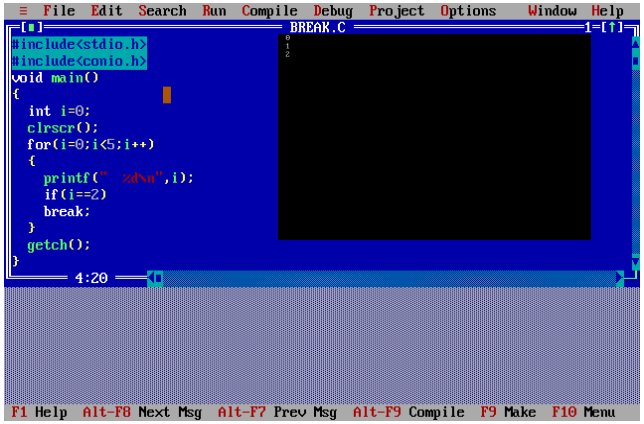


BREAK STATEMENT

While Loop



BREAK STATEMENT



CONTINUE STATEMENT

- It is a jump statement.
- It is used only inside the loop.
- Its execution does not exit from the loop but escape the loop for that iteration and transfer the control back to the loop for the new iteration.

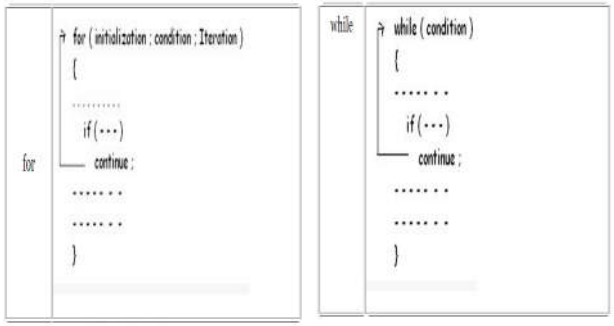
CONTINUE STATEMENT

Continue Statement :

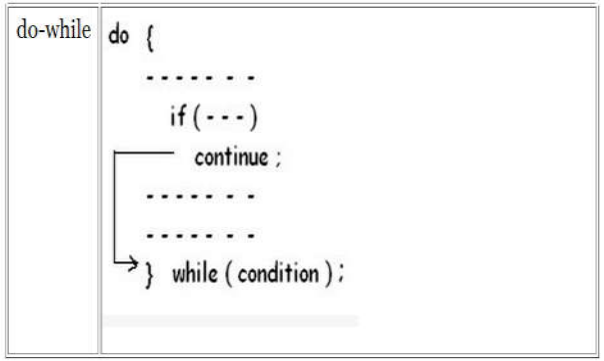
```
loop
{
  continue;
  //code
}
```

- It is used for Skipping part of Loop.
- Continue causes the remaining code inside a loop block to be skipped and causes execution to jump to the top of the loop block

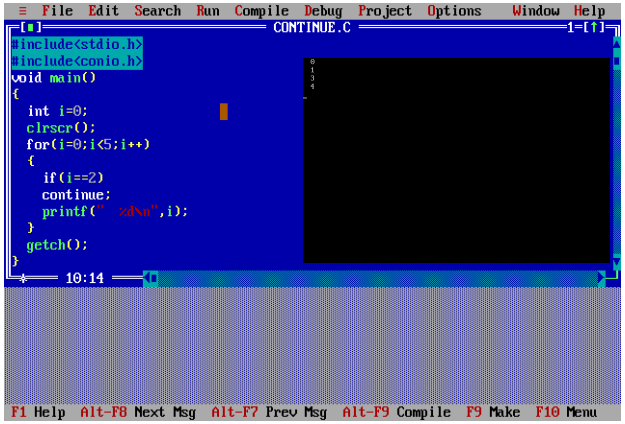
CONTINUE STATEMENT



CONTINUE STATEMENT



CONTINUE STATEMENT



GOTO STATEMENT

- It is used to alter the normal sequence of flow by transferring the control to some other part of the program unconditionally.
- It is followed by the label statement which determine the instruction to be handled next after the execution of the goto statement.

GOTO STATEMENT

Goto Statement

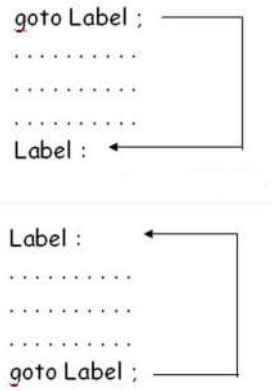
```
goto label;
-------
-------
label :
```

Whenever goto keyword encountered then it causes the program to continue on the line , so long as it is in the scope .

Types of Goto :

- Forward
- Backward

GOTO STATEMENT



GOTO STATEMENT

ASSIGNMENT

- Write a program to find whether a number is even or odd.
- Write a program to check whether a year is leap year or not.
- Write a program to calculate the sum of digits of a number.
- Write a program to swap two variables with and without using a third variable.
- Write a program to reverse a given number.
- Write a program to find the factorial of a given number.
- Write a program to print all the prime numbers between 1 and 300.
- Write a program to print all the armstrong numbers between 1 and 500.