Keep track of a condition true

Organised & Supported by **RuggedBOARD**

Agenda

- Unconditional and Looping Statements
- Looping (Iterative)Statements
- While
- do-while
- for
- goto
- continue
- break
- Infinite loop

Looping and Un-conditional statements

These can be used for repeating a block of code in specified no. of times or untill the condition returns false.

Ex : 1. while loop
2. do-while loop
3. for loop.

Advantages :
- To avoid the re-writing of the same code
- To reduce the length of a program
- Executing time is low
- Save memory space
- Debugging is easy.

The control will moves from one location to another location without checking any condition in a program.

Ex: 1. goto statement
2.break statement
3.continue statement
4.return statemen

while

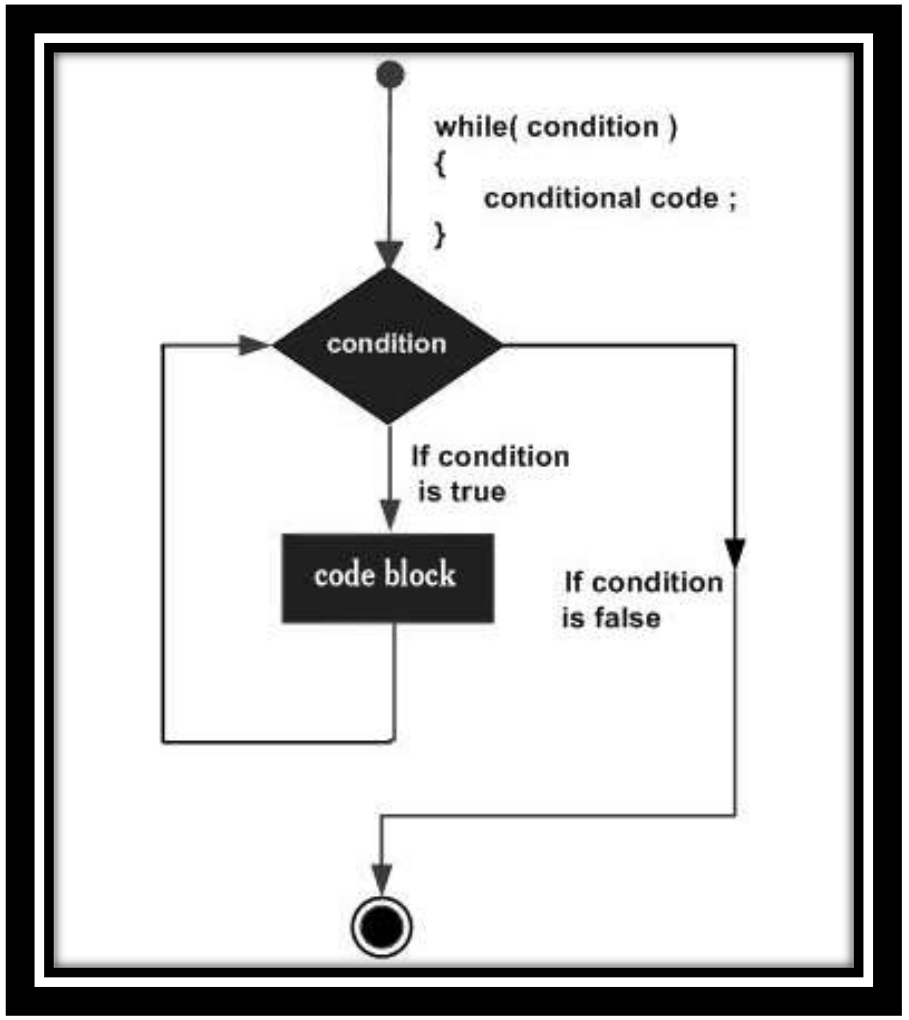
While:

- It is a pre-checking looping statement, checks the given condition. If it returns TRUE then the block of code is executed.
- That the block of code is executed until condition returns FALSE .
- If it returns FALSE then exit from the loop.

These can be used for repeating a block of code in specified number of times or untill the condition returns false.

Syn :

```
initialization ;
while (cond)
{
-----------------------;
-----------------------;
updatable statements;
}
```



while

```
#include <stdio.h>
 int main ()
 {
    /* local variable definition */
    int a = 10;
    /* while loop execution */
   while( a < 20 )
   {
        printf("value of a: %d\n", a);
        a++;
   }
   return 0;
 }
```

```
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
```

do-while

-It is post-checking looping statement.
-after executing the block of code then the condition will be checked.
If it returns TRUE then only -repeat the code otherwise control transfers to the next statement of that loop.

Syn:

```
        Initialization;
        do
            {
                    --------------------;
                    --------------------;
                    updatable statement;
            }
while(cond) ;
```

Note: It can execute the block of code at least once. This is the main difference between do-while and while looping statements.

do-while

```
#include <stdio.h>
 int main ()
 {
    /* local variabledefinition */
    int a = 10; /* do loop execution */
    do
    {
            printf("value of a:%d\n", a); a++;
     } while( a < 20 );
     return 0;
}
```

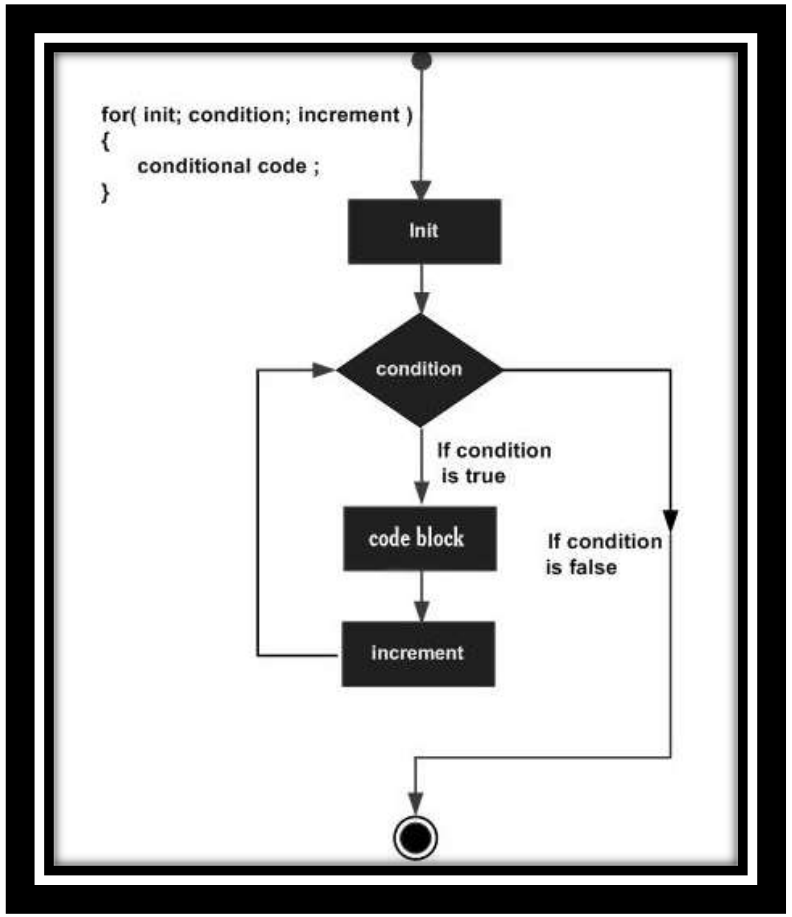
```
value of a:10
value of a:11
value of a:12
value of a:13
value of a:14
value of a:15
value of a:16
value of a:17
value of a:18
value of a:19
```

for

-It is also a pre-checking looping statement like while statement.
That means, initially checks the given condition.
-If it returns TRUE then the block of code is executed until condition returns FALSE .
-If it returns FALSE then exit from the loop.

Syn:

```
for ( initialization ; condition ; updatable statement )
{
// body of loop
}
```



Note:The main difference is the statements
(Initialization, condition and updatable) are in the same line in for loop

for

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

     /* for loop execution */
     for( a = 10; a < 20; a = a + 1 )
     {
          printf("value of a: %d\n", a);
      }

     return 0;
}
```

```
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
```

goto

It is the un-conditional control statement.
It can transfer the control from one location to another location in the same program without checking any condition.

The word **goto** is the one of the key word.

```
goto label ;            // declaration or/and calling a label
label :
        executable statements ;
--------------------------------
--------------------------------
```

When the goto statement is executed, control transfers to the definition of the label.
Any no. of gotos can be used in a program.
But, each label must be defined uniquely.

goto

```
#include <stdio.h>

int main()
{
      int num = 26;
       if (num % 2 == 0)
            // jump to even
            goto even;
       else
            // jump to odd
            goto odd;

even:
   printf("%d is even\n", num);
   return 0;
odd:
   printf("%d is odd\n", num);
   return 0;
}
```

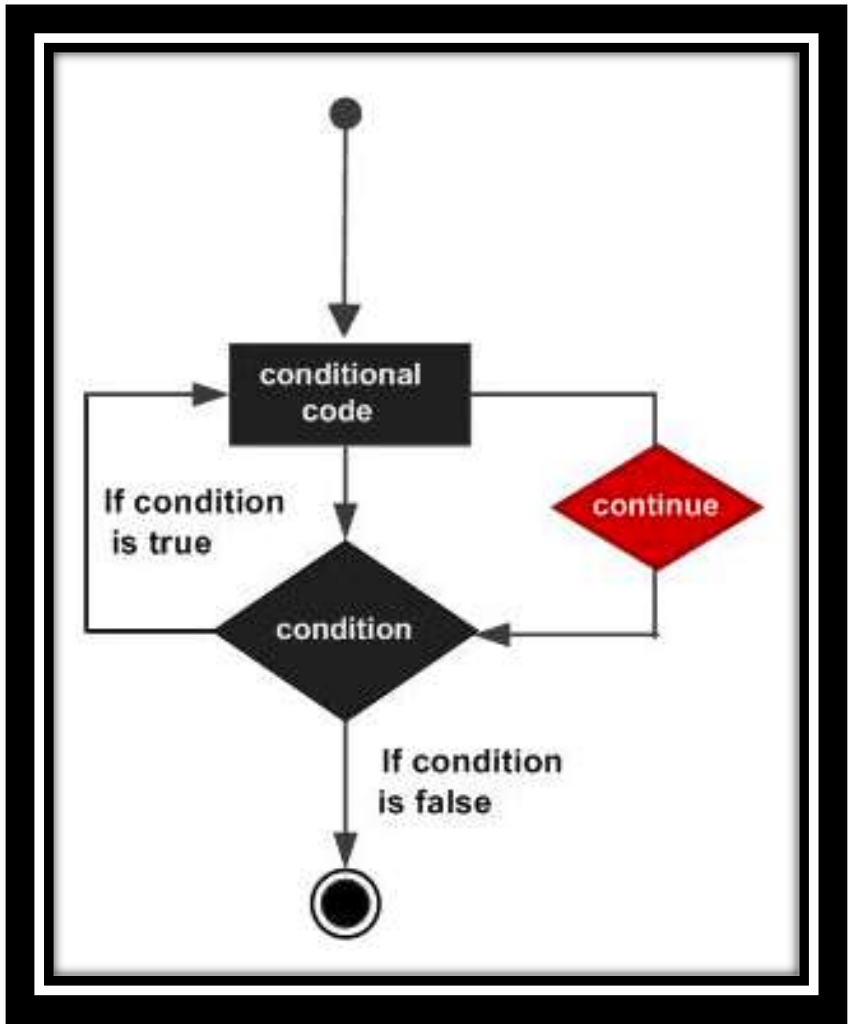
```
26 is even
```

continue

Continue forces the next iteration of the loop to take place, skipping any code in between.

```
#include <stdio.h>
int main ()
{
        /* local variable definition */
        int a = 10;
        /* do loop execution */
        do
        {
                if( a == 15)
                {
                        /* skip the iteration */

                        a = a + 1; continue;
                }
                printf("value of a: %d\n", a);
                a++;
        } while( a < 20 );
        return 0;
}
```



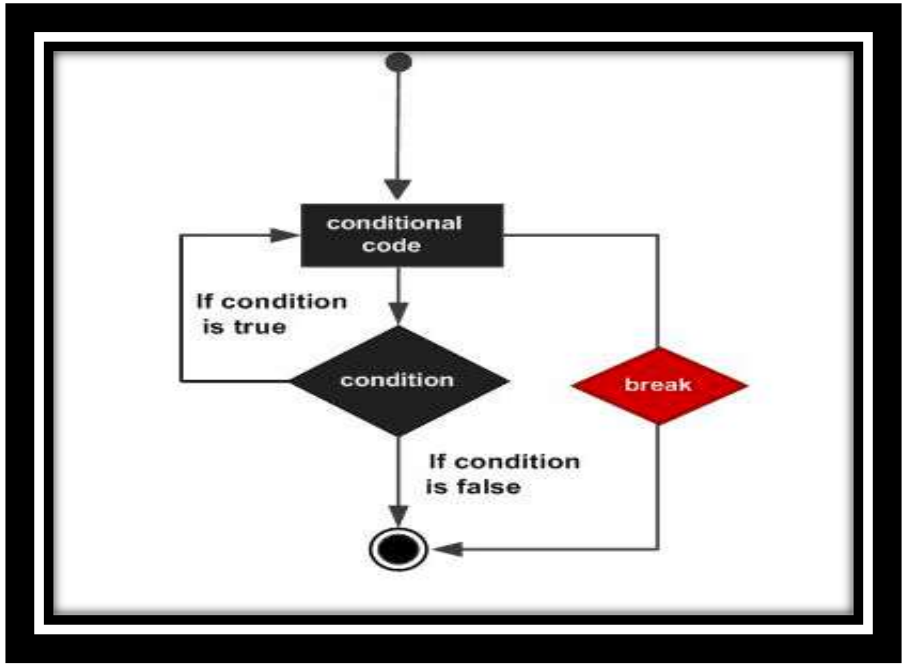
break

The **break** statement in C programming has the following two usages –

•When a **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

•It can be used to terminate a case in the **switch** statement .

[If you are using nested loops, the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.]




```
#include <stdio.h>
int main ()
 {
        /* local variable definition */
         int a = 10;
         /* while loop execution */
         while( a < 20 )
         {
              printf("value of a: %d\n", a);
              a++;
              if( a > 15)
               {
                   /* terminate the loop
                    using break statement */
                         break;
               }
           }
           return 0;
}
```

Thank YOU