

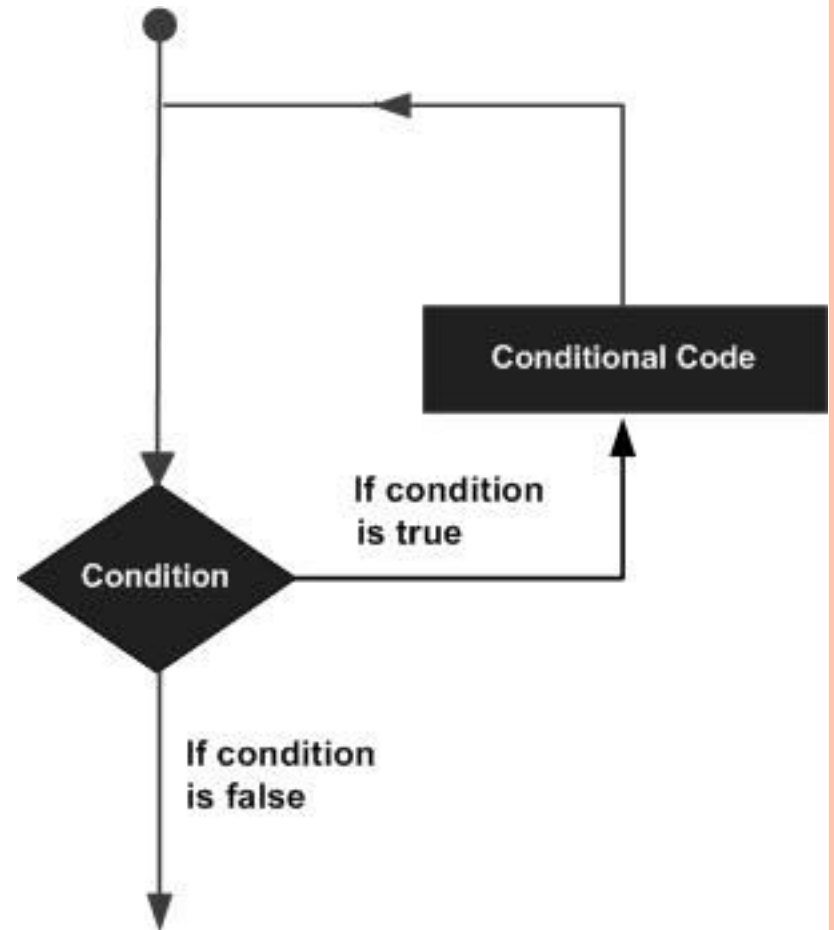


C – LOOP

Dr. Sheak Rashed Haider Noori
Associate Professor & Associate Head
Department of Computer Science

C – LOOP

- 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 and following is the general form of a loop statement in most of the programming languages:



TYPES OF LOOP

C programming language provides the following types of loop to handle looping requirements.

Statement	Description
1. for loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
2. while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
3. do...while loop	Like a while statement, except that it tests the condition at the end of the loop body
4. nested loop	You can use one or more loop inside any another while, for or do..while loop.


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:**

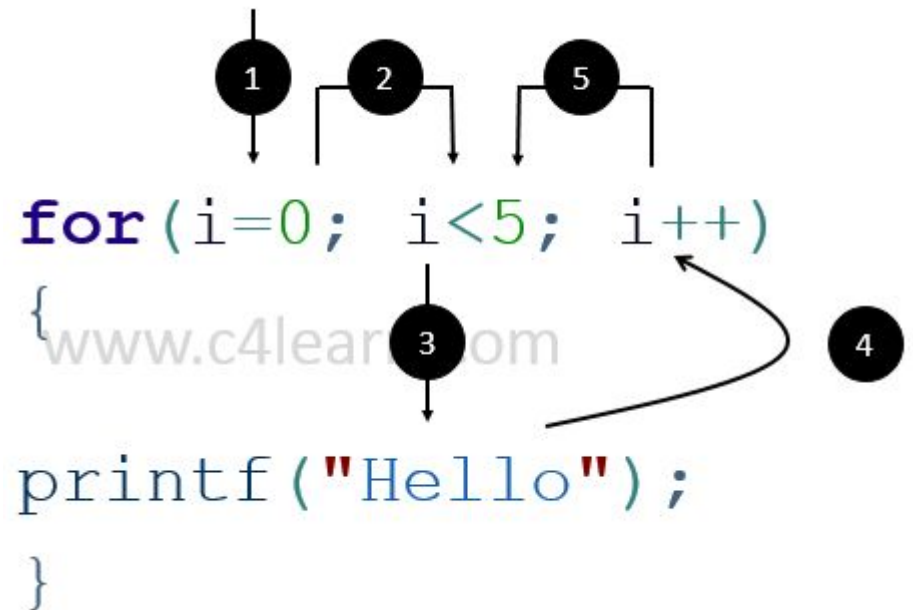
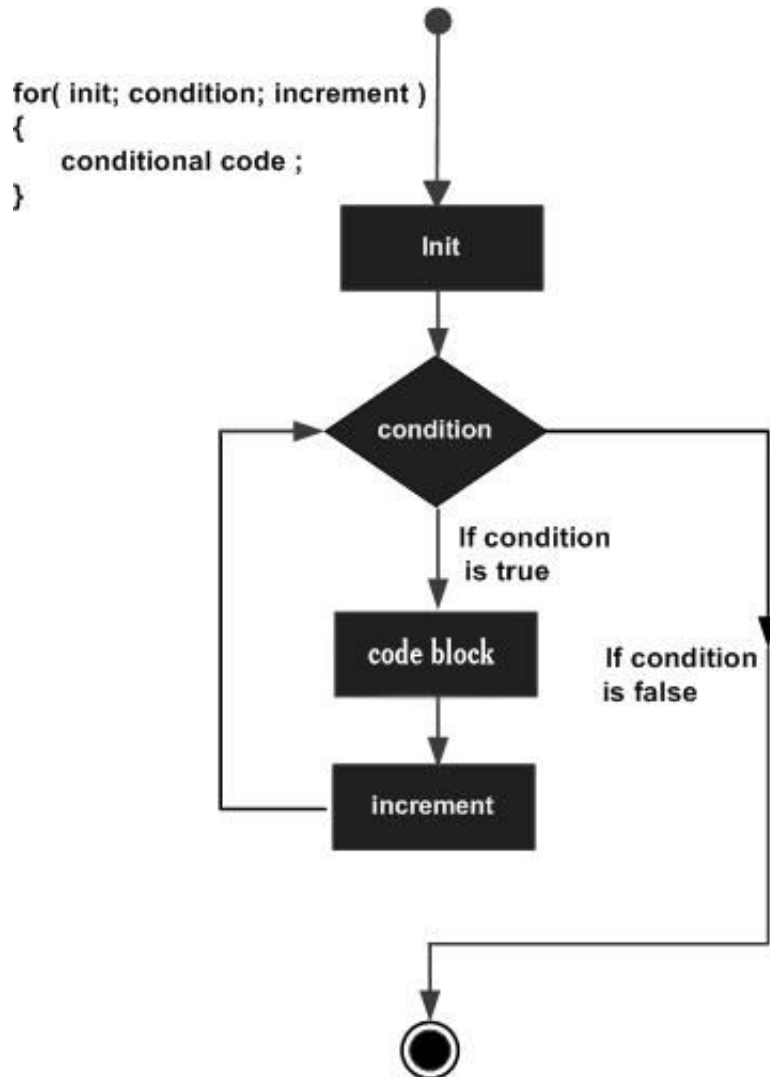
```
for ( init; condition; increment )  
{  
    statement(s);  
}
```



FLOW OF CONTROL IN A 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.
 - 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.
- 

FLOW DIAGRAM : FOR LOOP



CODE EXAMPLE

```
#include <stdio.h>

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

    return 0;
}
```

Output:


```
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
```



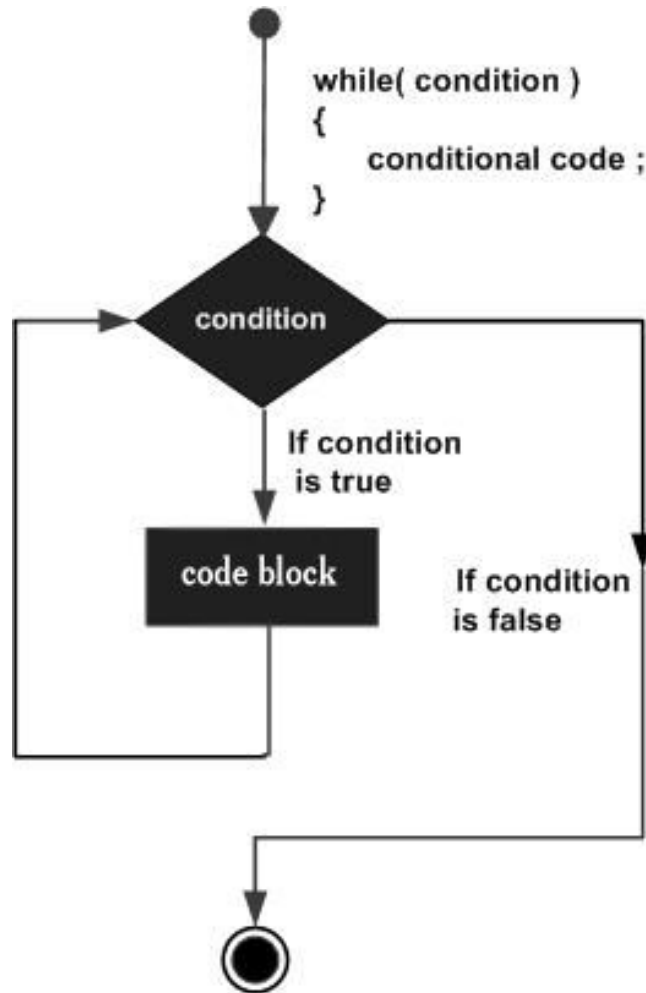
2. WHILE LOOP

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

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

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

FLOW DIAGRAM: WHILE LOOP



Here, key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

CODE EXAMPLE : IF ELSE

```
#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;
}
```

Output:

```
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
```



3. DO..WHILE LOOP

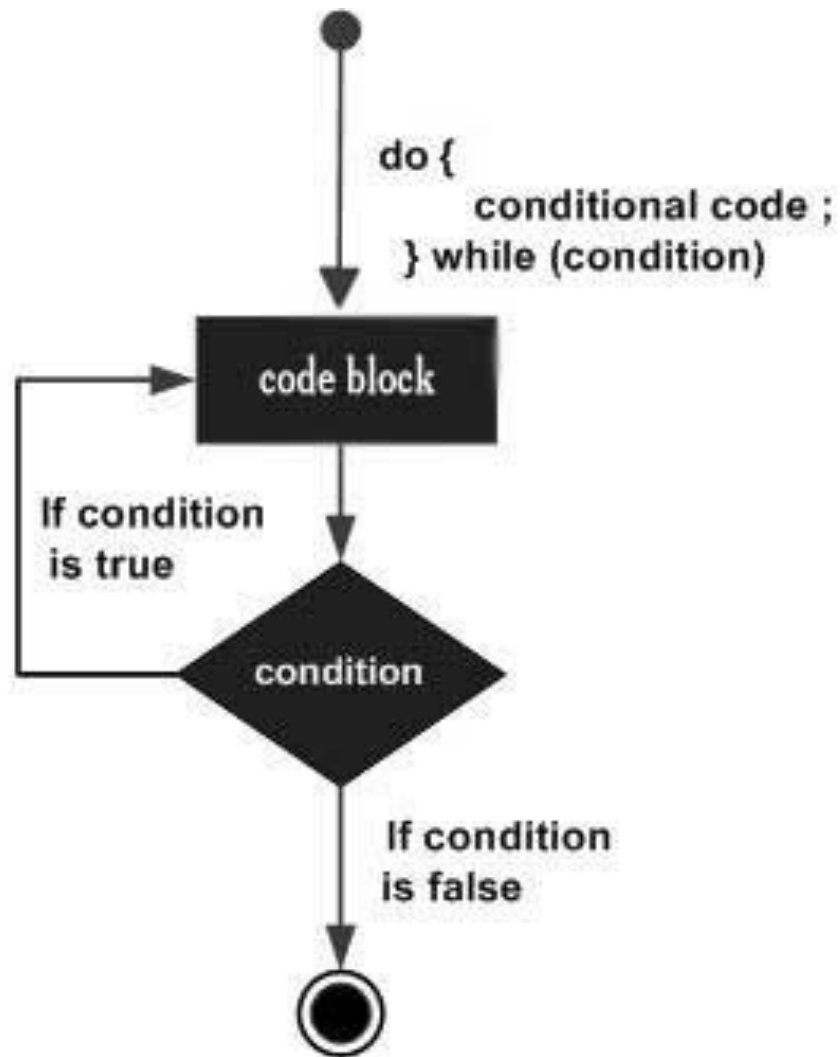
- Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop checks its condition at the bottom of the loop.
- A **do...while** loop is similar to a **while** loop, except that a **do...while** loop is guaranteed to execute at least one time.

- **Syntax:**

```
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.
- If the condition is true, the flow of control jumps back up to `do`, and the `statement(s)` in the loop execute again. This process repeats until the given condition becomes false.

FLOW DIAGRAM: DO..WHILE LOOP



CODE EXAMPLE: DO...WHILE

```
#include <stdio.h>

int main ()
{
    /* local variable definition */
    int a = 10;

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

    return 0;
}
```

Output:

```
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
```



NESTED LOOPS IN C

- It is allow to use one loop inside another loop.
- The syntax for a **nested for loop** statement

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

- The syntax for a **nested while loop** statement

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



NESTED LOOPS IN C

- The syntax for a **nested do...while loop**

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

- A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example, a for loop can be inside a while loop or vice versa.



CODE EXAMPLE NESTED FOR LOOP

```
#include <stdio.h>

void main ()
{
    int i, j;

    for(i=0; i<5; i++) /*outer loop*/
    {
        printf("\nI is %d\n", i);
        for(j=0; j < 5; j++) /*inner loop*/
        {
            printf("J=%d,", j);
        }
    }
}
```

```
I is 0
J=0, J=1, J=2, J=3, J=4,
I is 1
J=0, J=1, J=2, J=3, J=4,
I is 2
J=0, J=1, J=2, J=3, J=4,
I is 3
J=0, J=1, J=2, J=3, J=4,
I is 4
J=0, J=1, J=2, J=3, J=4,
```



CODE EXAMPLE NESTED FOR LOOP

```
#include <stdio.h>

void main ()
{
    int i, j;
    for(i=0; i<5; i++) /*outer loop*/
    {
        for(j=0; j<5; j++) /*inner loop*/
        {
            printf("*");
        }
        printf("\n");
    }
}
```

* * * * *

* * * * *

* * * * *

* * * * *

* * * * *



LOOP CONTROL STATEMENTS:

Loop control statements change execution from its normal sequence. C supports the following control statements.

Control Statement	Description
break statement	Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

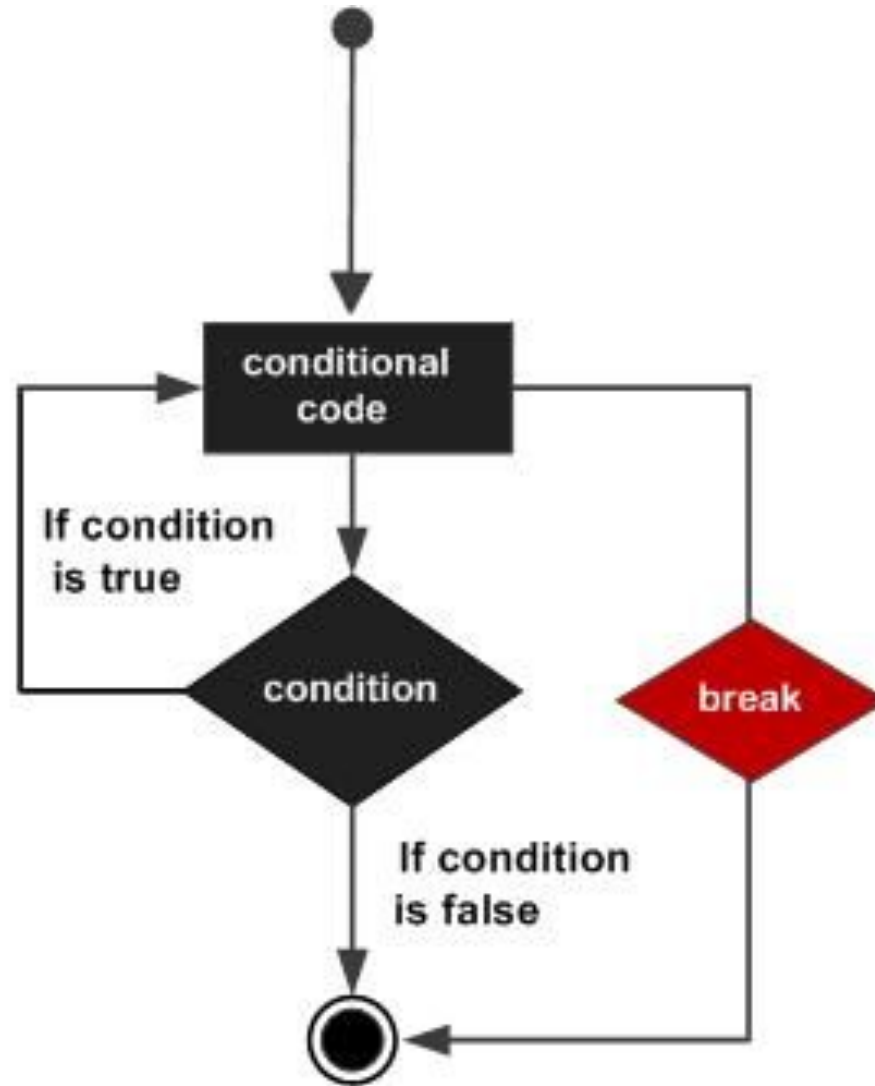


BREAK STATEMENT

- The **break** statement in C programming language has the following two usages:
 - When the **break** statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
 - It can be used to terminate a case in the **switch** statement (we already saw).
- If you are using nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.
- **Syntax:**
break;



FLOW DIAGRAM: BREAK STATEMENT



CODE EXAMPLE: BREAK

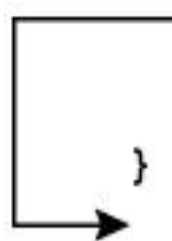
```
#include <stdio.h>

int main ()
{
    int i;
    for(i=0;i<6;i++)
    {
        if(i==3)
        {
            /*terminate the loop using break statement*/
            break;
        }
        printf("Value of i: %d\n", i);
    }
}
```

Value of i: 0
Value of i: 1
Value of i: 2

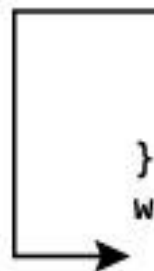
BREAK

```
while (test expression) {  
    statement/s  
    if (test expression) {  
        break;  
    }  
    statement/s  
}
```



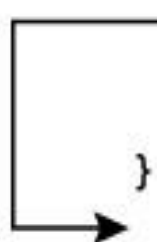
A flowchart illustrating the execution of a while loop. It starts with a loop header 'while (test expression) {'. The body contains 'statement/s', an 'if (test expression) {' block with a 'break;' statement, and another 'statement/s'. A line connects the 'break;' statement to the closing brace '}' of the while loop, indicating an exit path. Another line connects the closing brace '}' back to the start of the loop body, indicating a continuation path.

```
do {  
    statement/s  
    if (test expression) {  
        break;  
    }  
    statement/s  
} while (test expression);
```



A flowchart illustrating the execution of a do-while loop. It starts with a loop header 'do {'. The body contains 'statement/s', an 'if (test expression) {' block with a 'break;' statement, and another 'statement/s'. A line connects the 'break;' statement to the closing brace '}' of the do block, indicating an exit path. Another line connects the closing brace '}' to the 'while (test expression);' condition, which then loops back to the start of the do block.

```
for (initial expression; test expression; update expression) {  
    statement/s  
    if (test expression) {  
        break;  
    }  
    statements/  
}
```



A flowchart illustrating the execution of a for loop. It starts with a loop header 'for (initial expression; test expression; update expression) {'. The body contains 'statement/s', an 'if (test expression) {' block with a 'break;' statement, and another 'statements/'. A line connects the 'break;' statement to the closing brace '}' of the for loop, indicating an exit path. Another line connects the closing brace '}' back to the start of the loop body, indicating a continuation path.

NOTE: The break statment may also be used inside body of else statement.

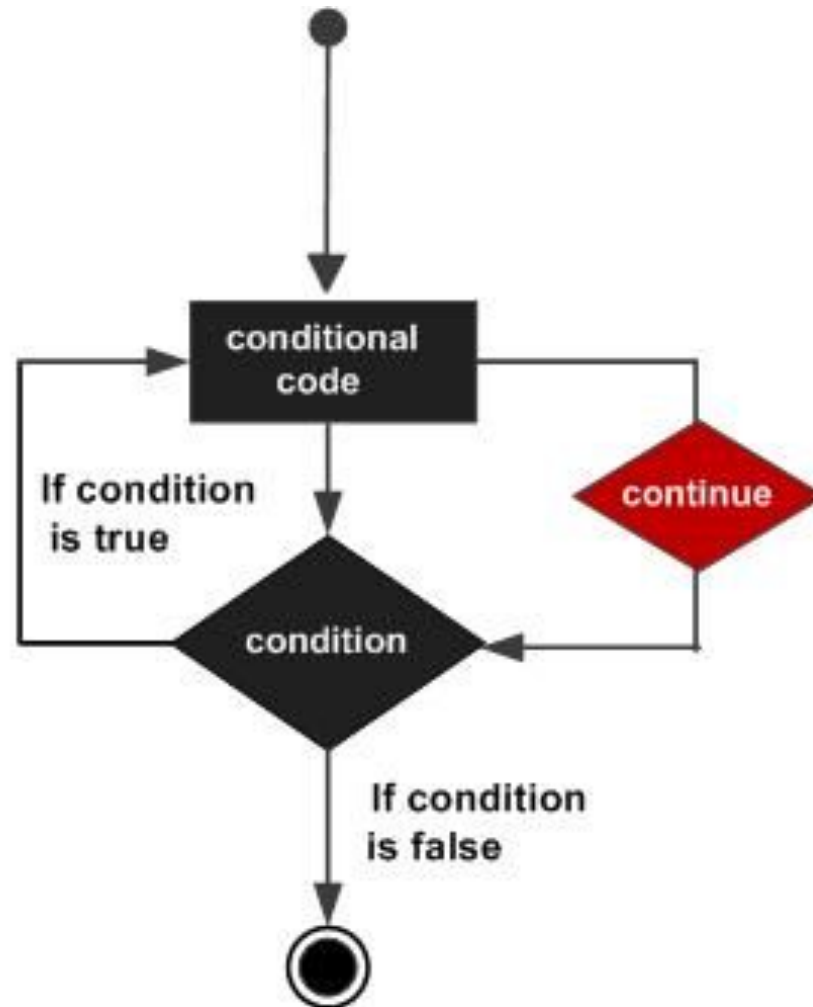


CONTINUE STATEMENT

- The **continue** statement works somewhat like the **break** statement. Instead of forcing termination, however, **continue** forces the next iteration of the loop to take place, skipping any code in between.
- For the **for** loop, **continue** statement causes the conditional test and increment portions of the loop to execute. For the **while** and **do...while** loops, **continue** statement causes the program control passes to the conditional tests.
- **Syntax:**
continue;



FLOW DIAGRAM: CONTINUE



CODE EXAMPLE: CONTINUE

```
#include <stdio.h>

int main ()
{
    int i;
    for(i=0;i<6;i++)
    {
        if( i == 3)
        {
            /* skip the iteration */
            continue;
        }
        printf("value of i: %d\n", i);
    }
    return 0;
}
```

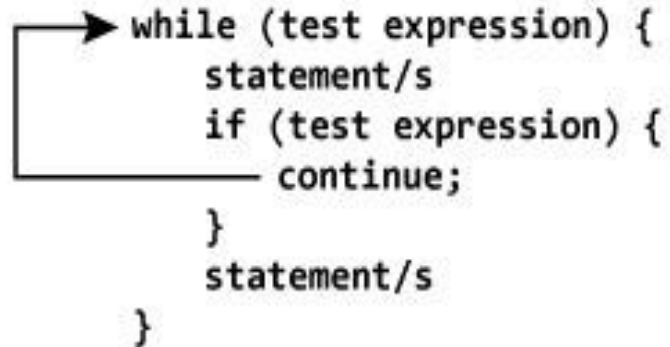
value of i: 0
value of i: 1
value of i: 2
value of i: 4
value of i: 5

Notice
that 3 is
not there!

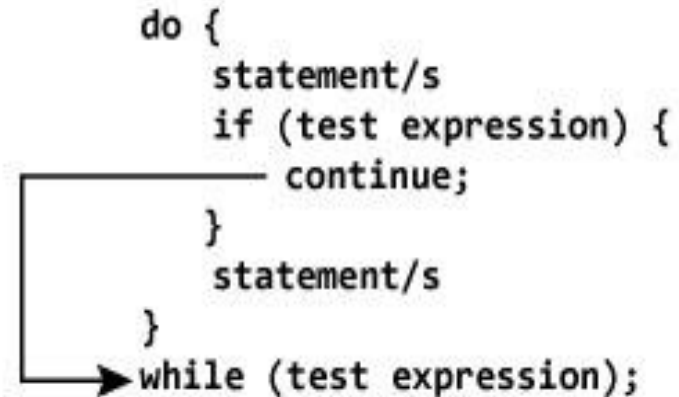


CONTINUE

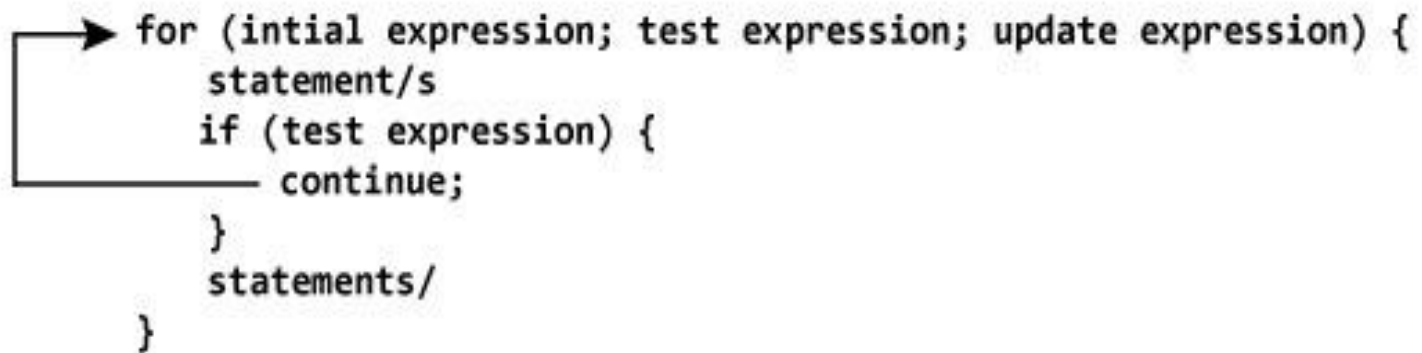
```
while (test expression) {  
    statement/s  
    if (test expression) {  
        continue;  
    }  
    statement/s  
}
```



```
do {  
    statement/s  
    if (test expression) {  
        continue;  
    }  
    statement/s  
} while (test expression);
```



```
for (initial expression; test expression; update expression) {  
    statement/s  
    if (test expression) {  
        continue;  
    }  
    statements/  
}
```



NOTE: The continue statment may also be used inside body of else statement.

TEST: CONVERT THE FOLLOWING FLOW CHART IN A C PROGRAM

