

Problem Solving Through programming in C  
Course Code: ONL1001

Looping: For, Nested for loop,  
while

Ms. SHUBHRA  
DWIVEDI  
School - SCOPE  
VIT-AP Amaravati

# Looping

# LOOPS

- You may encounter situations, when a block of code needs to be executed 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.
- Programming languages provide various control structures that allow for more complicated execution paths.
- Loops in programming come into use when we need to repeatedly execute a block of statements. For example: Suppose we want to print “Hello World” 10 times. This can be done in two ways as shown below:

## Iterative Method

[illegible]

# Using Loops

In Loop, the statement needs to be written only once and the loop will be executed 10 times as shown below.

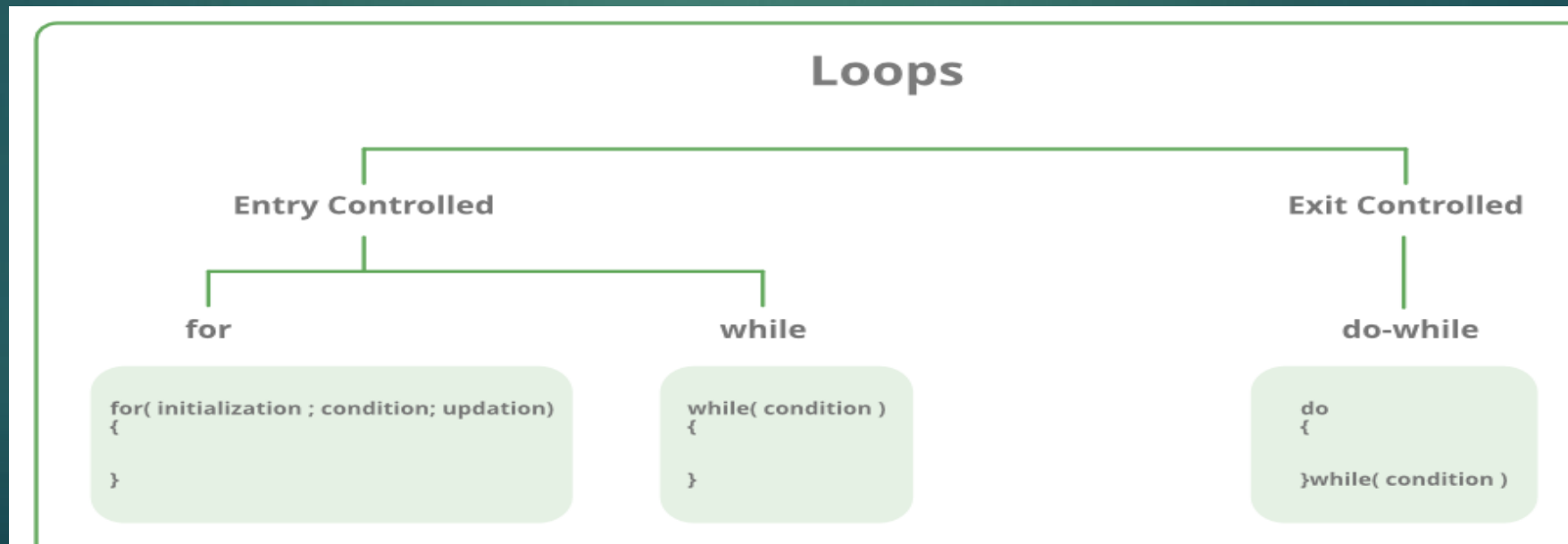
In computer programming, a loop is a sequence of instructions that is repeated until a certain condition is reached.

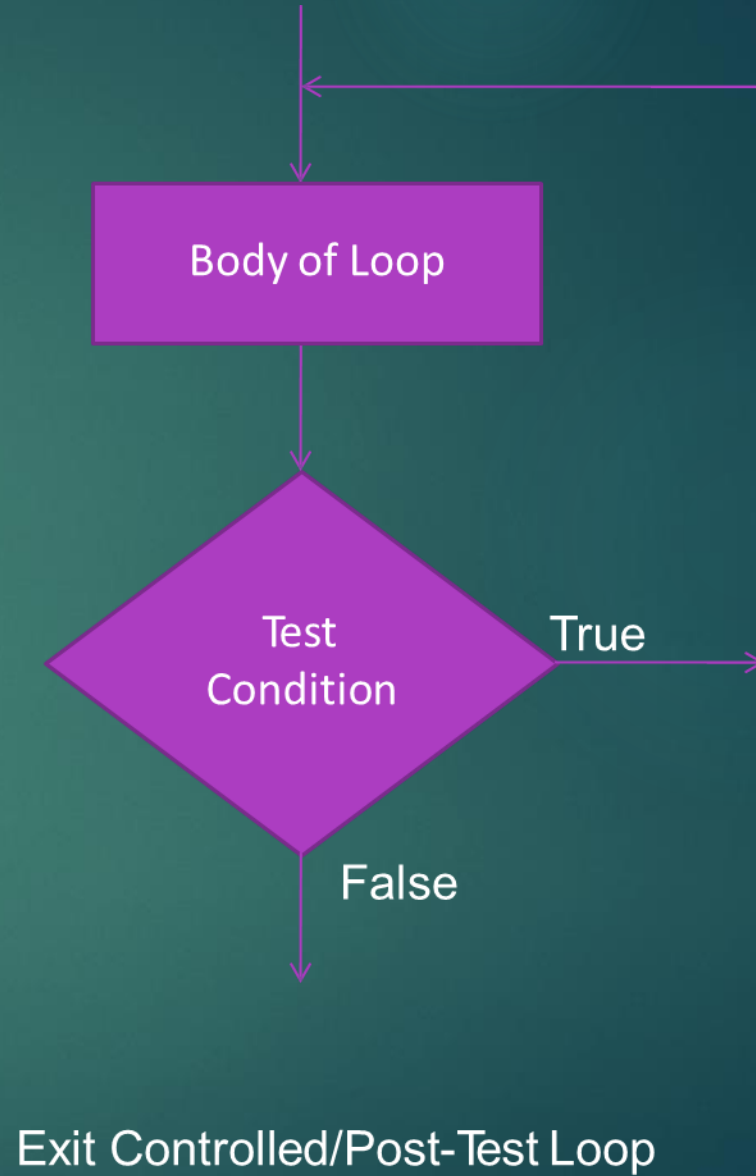
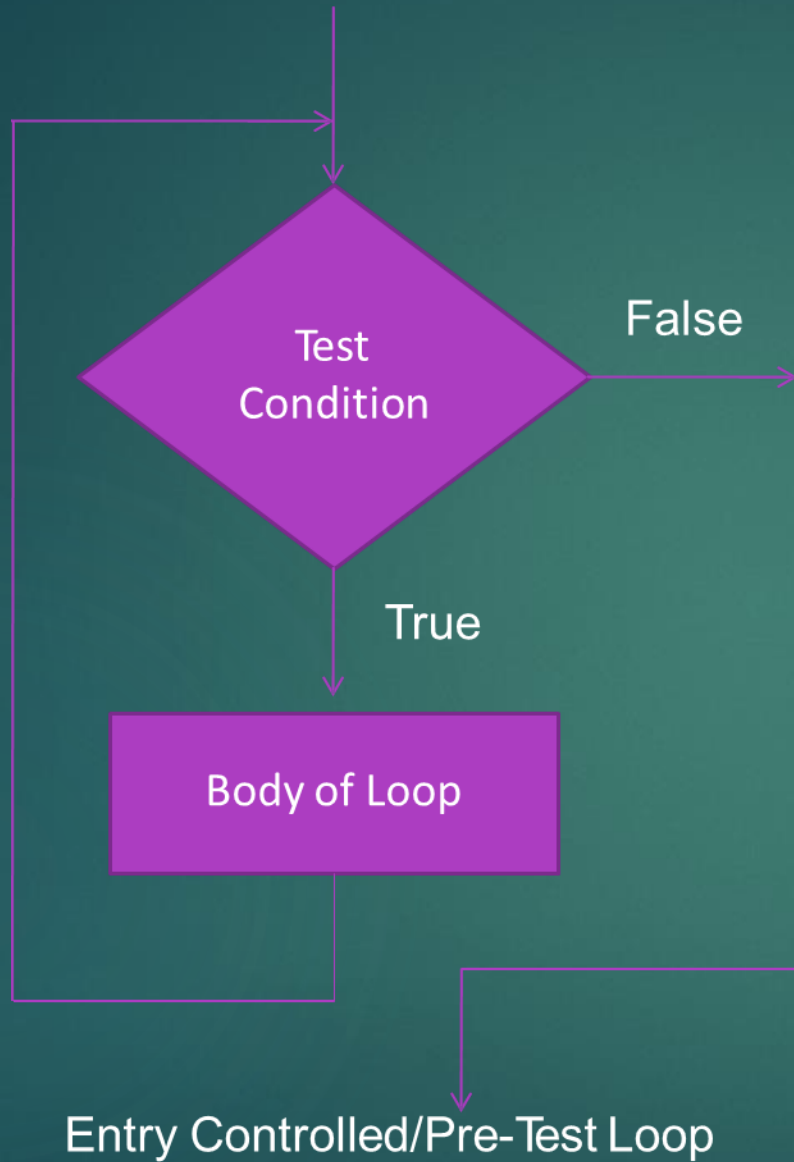
- An operation is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number.
- Counter not Reached: If the counter has not reached the desired number, the next instruction in the sequence returns to the first instruction in the sequence and repeat it.
- Counter reached: If the condition has been reached, the next instruction “falls through” to the next sequential instruction or branches outside the loop.

There are mainly two types of loops:

**Entry Controlled loops:** In this type of loops the test condition is tested before entering the loop body. **For Loop** and **While Loop** are entry controlled loops.

**Exit Controlled Loops:** In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore, the loop body will execute atleast once, irrespective of whether the test condition is true or false. **do – while loop is exit controlled loop.**







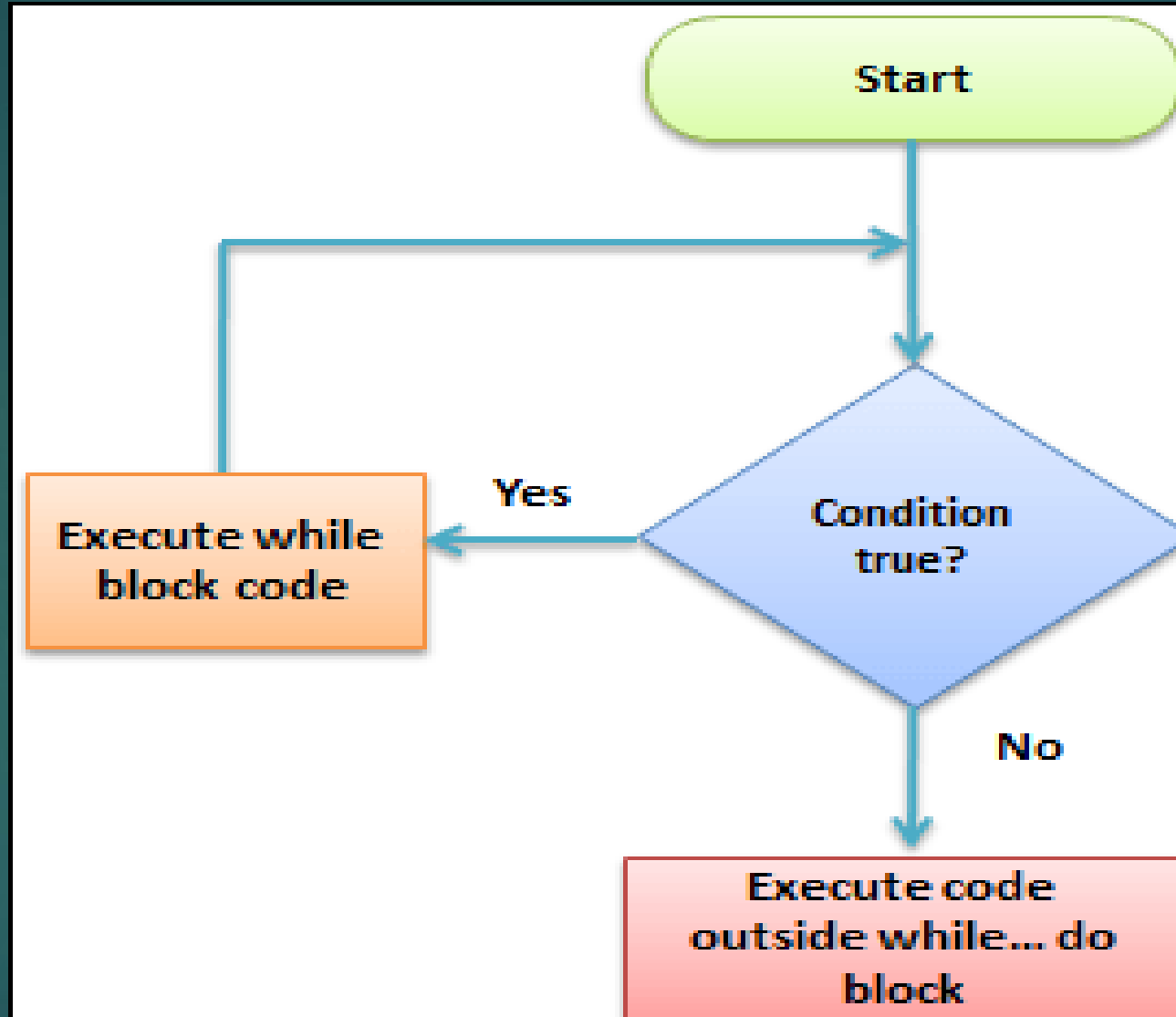
Loop Consists of:

- Body of the loop
- Control Statement


## STEPS IN LOOPING

- Initialization of condition variable
- Test the control statement
- Execution of body of the loop
- Updating the condition variable

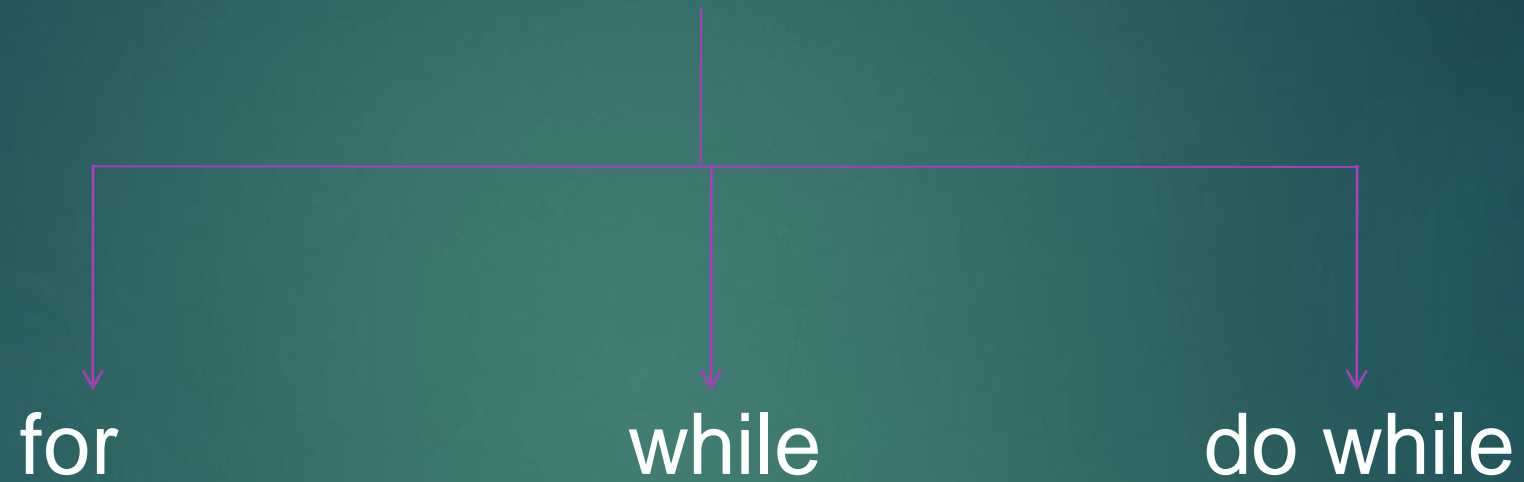




Sample Loop

- 
- As per the above diagram, if the Test Condition is true, then the loop is executed, and if it is false then the execution breaks out of the loop. After the loop is successfully executed the execution again starts from the Loop entry and again checks for the Test condition, and this keeps on repeating.
  - The sequence of statements to be executed is kept inside the curly braces { } known as the **Loop body**. After every execution of the loop body, **condition is verified**, and if it is found to be true the loop body is executed again. When the condition check returns false, the loop body is not executed, and execution breaks out of the loop.


# COMMON LOOPS



# For Loop

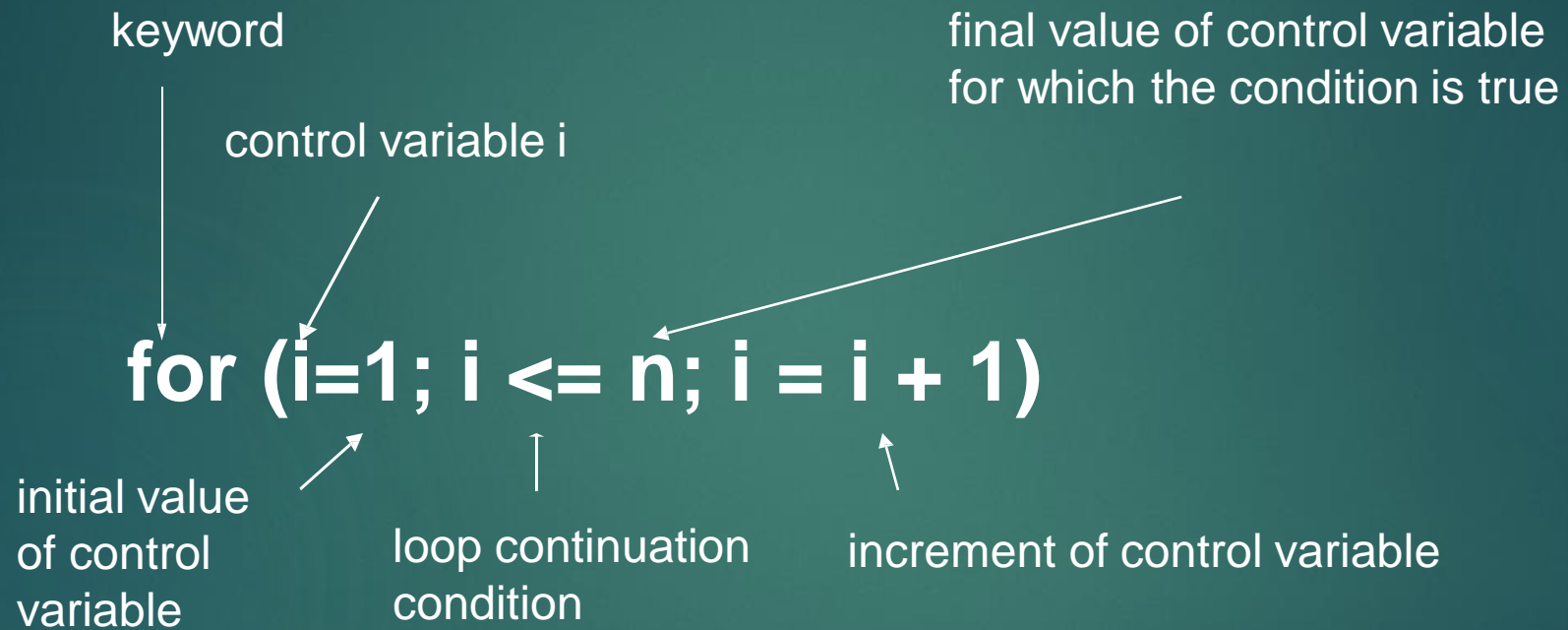
Syntax:

```
for (initialization expr; test expr; update expr)
{
    // body of the loop
    // statements we want to execute
}
```



In for loop, a loop variable is used to control the loop. First initialize this loop variable to some value, then check whether this variable is less than or greater than counter value. If statement is true, then loop body is executed and loop variable gets updated . Steps are repeated till exit condition comes.

1. **Initialization Expression:** In this expression we have to initialize the loop counter to some value. for example: `int i=1;`
2. **Test Expression:** In this expression we have to test the condition. If the condition evaluates to true then we will execute the body of loop and go to update expression otherwise we will exit from the for loop. For example: `i <= 10;`
3. **Update Expression:** After executing loop body this expression increments/decrements the loop variable by some value. for example: `i++;`



A diagram illustrating the syntax of a `for` loop. The code `for (i=1; i <= n; i = i + 1)` is centered on a dark teal background. Six white arrows point from descriptive text labels to specific parts of the code: 

- An arrow from "keyword" points to the word `for`.
- An arrow from "control variable i" points to the variable `i` in the initialization `i=1`.
- An arrow from "initial value of control variable" points to the value `1` in the initialization `i=1`.
- An arrow from "loop continuation condition" points to the comparison `i <= n`.
- An arrow from "increment of control variable" points to the increment expression `i = i + 1`.
- An arrow from "final value of control variable for which the condition is true" points to the variable `n` in the condition `i <= n`.

```
for (i=1; i <= n; i = i + 1)
```

keyword

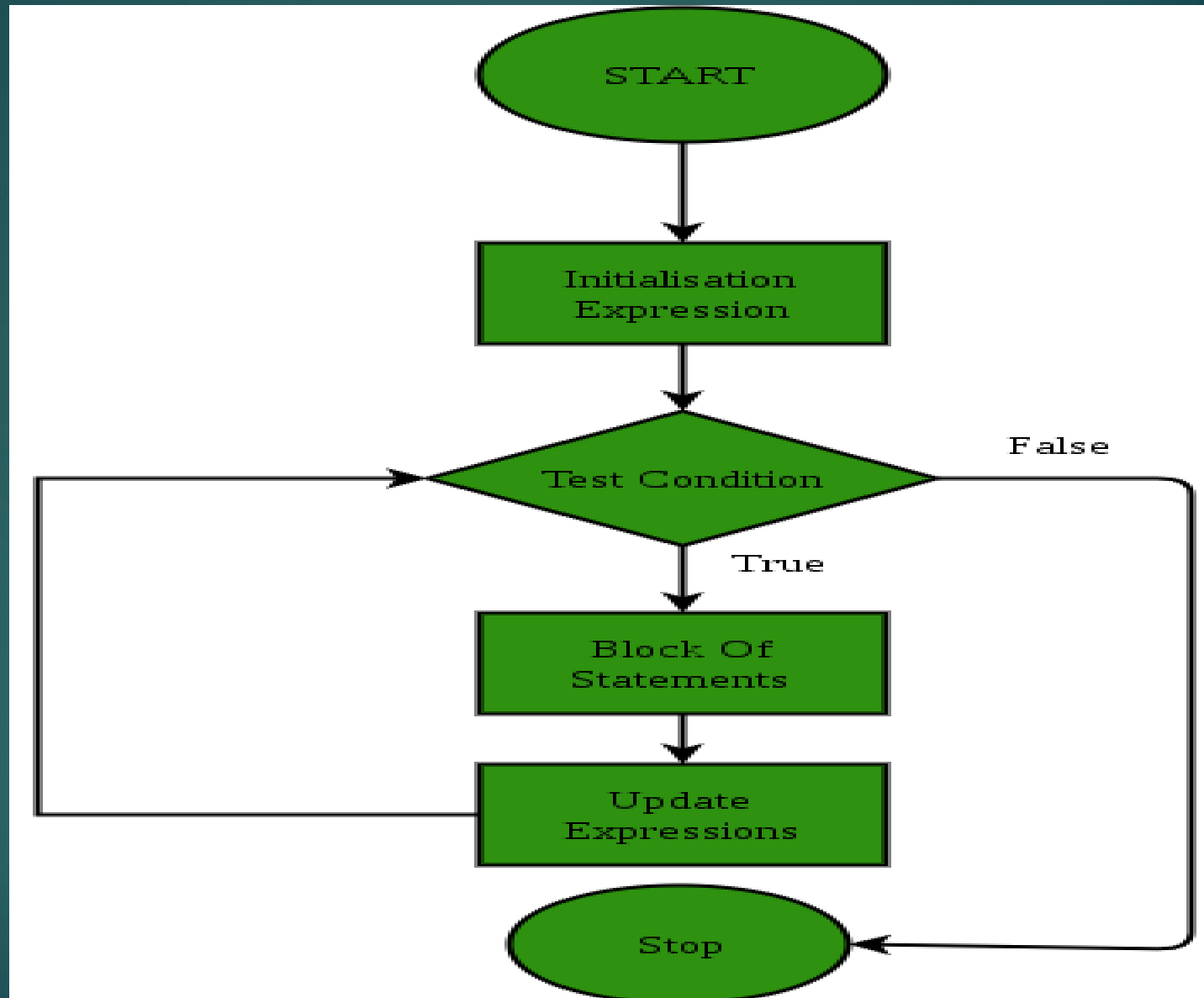
control variable i

initial value of control variable

loop continuation condition

increment of control variable

final value of control variable for which the condition is true



# EXAMPLES

- Vary the control variable from 1 to 100 in increments of 1  
`for (i = 1; i <= 100; i++)`
- Vary the control variable from 100 to 1 in increments of -1  
`for (i = 100; i >= 1; i--)`
- Vary the control variable from 5 to 55 in increments of 5  
`for (i = 5; i <= 55; i+=5)`



// C program to illustrate for loop

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int i=0;
```

```
    for (i = 1; i <= 10; i++)
```

```
    {
```

```
        printf( "Hello World\n");
```

```
    }
```

```
    return 0;
```

```
}
```

Output:

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

## Example: Program to print first 10 natural numbers

```
#include<stdio.h>


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

Output:  
1 2 3 4 5 6 7 8 9 10

```
#include <stdio.h>
int main()
{
    int a ;
    int b ;
    for ( a = 1 , b = 5 ; a <= 5 , b >= 1 ; a ++ , b -- )
    {
        printf ( "a = %d\t b = %d\n" , a , b ) ;
    }
    return 0;
}
```

Output

a = 1	b = 5
a = 2	b = 4
a = 3	b = 3
a = 4	b = 2
a = 5	b = 1



```
#include <stdio.h>
int main()
{
    int sum=0;
    for(int i = 0; i<=10; i++){
        sum = sum+i;
        if(i==5){
            goto addition;
        }
    }

    addition:
    printf("%d", sum);

    return 0;
}
```

Output:15

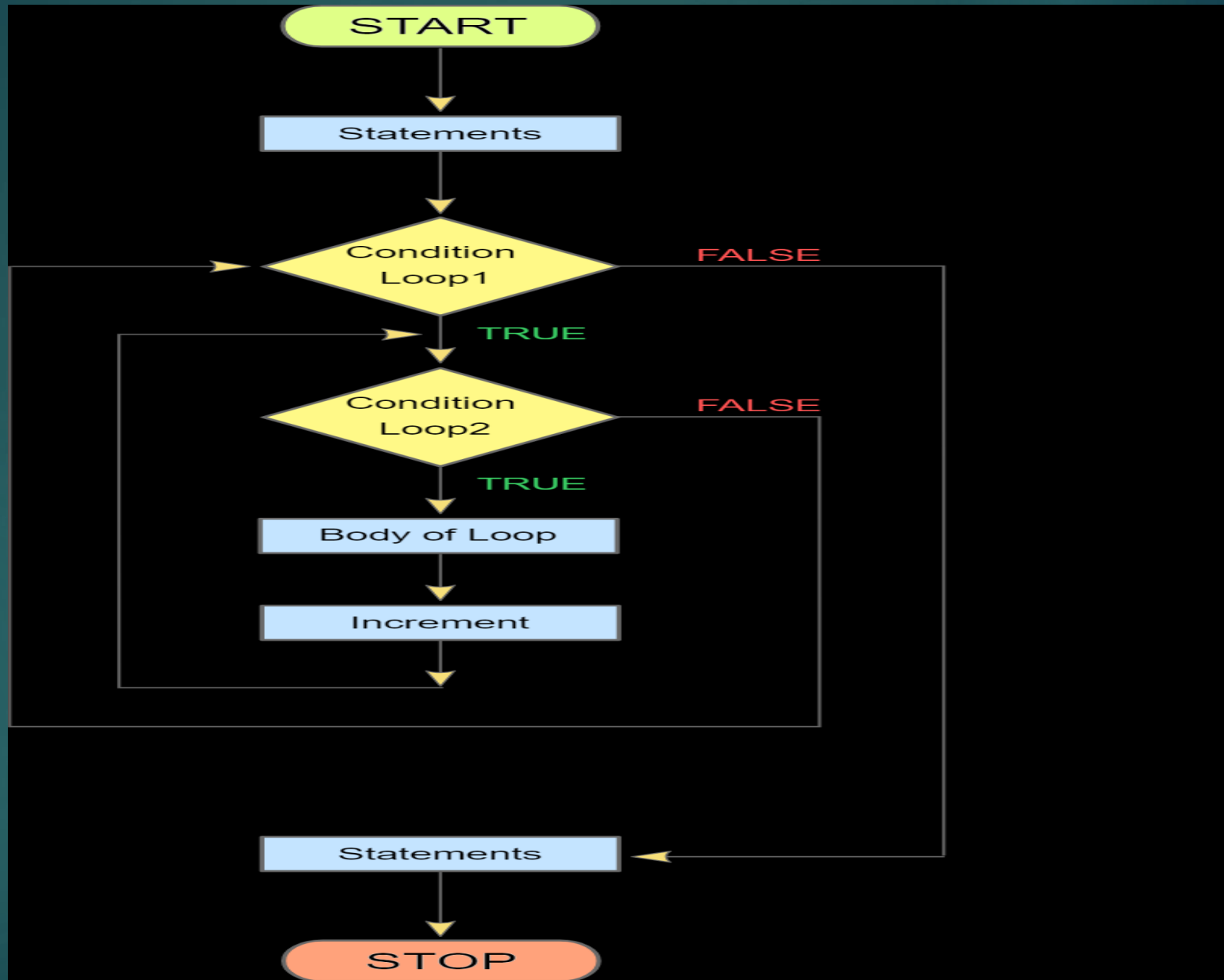
Explanation: In this example, we have a label addition and when the value of i (inside loop) is equal to 5 then we are jumping to this label using goto. This is reason the sum is displaying the sum of numbers till 5 even though the loop is set to run from 0 to 10


# NESTED FOR LOOPS

- Nested loop means a loop statement inside another loop statement. That is why nested loops are also called as “loop inside loop”.
- Executed from the inside out
  - Each loop is like a layer and has its own counter variable, its own loop expression and its own loop body
  - In a nested loop, for each value of the outermost counter variable, the complete inner loop will be executed once

# General form

```
for(initialization; condition; increment/decrement)
{
    for(initialization; condition; increment/decrement)
    {
        statement ;
    }
}
```





```
#include <stdio.h>
int main()
{
    int a, b;
    for(a = 1; a <= 5; a++)
    {
        for(b = 1; b <= 5; b++)
        {
            printf("%d ", b);
        }
        printf("\n");
    }
    return 0;
}
```


output

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

Note:

The variable a and b is initialized to 1 for the first time when the program execution starts in the for loop. The variable a belongs to outer for loop and a variable b belongs to inner for loop. for a = 1, inner for loop will be executed 5 times. Thus, total number of iterations is 25.





```
#include <stdio.h>
int main()
{
    for (int i=0; i<2; i++)
    {
        for (int j=0; j<4; j++)
        {
            printf("%d, %d\n",i ,j);
        }
    }
    return 0;
}
```

Output:

```
0, 0
0, 1
0, 2
0, 3
1, 0
1, 1
1, 2
1, 3
```

```

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

    for(i = 12;i<=14;i++)
    { /*outer loop*/

        printf("Table of %d\n",i);

        for(j = 1;j<=10;j++)
        { /*inner loop*/

            printf("%d*%d\t=\t%d\n",i,j,i*j);
        }
    }
    return 0;
}

```

Table of 12

12*1	=	12
12*2	=	24
12*3	=	36
12*4	=	48
12*5	=	60
12*6	=	72
12*7	=	84
12*8	=	96
12*9	=	108
12*10	=	120

Table of 13

13*1	=	13
13*2	=	26
13*3	=	39
13*4	=	52
13*5	=	65
13*6	=	78
13*7	=	91
13*8	=	104
13*9	=	117
13*10	=	130

Table of 14

14*1	=	14
14*2	=	28
14*3	=	42
14*4	=	56
14*5	=	70
14*6	=	84
14*7	=	98
14*8	=	112
14*9	=	126
14*10	=	140

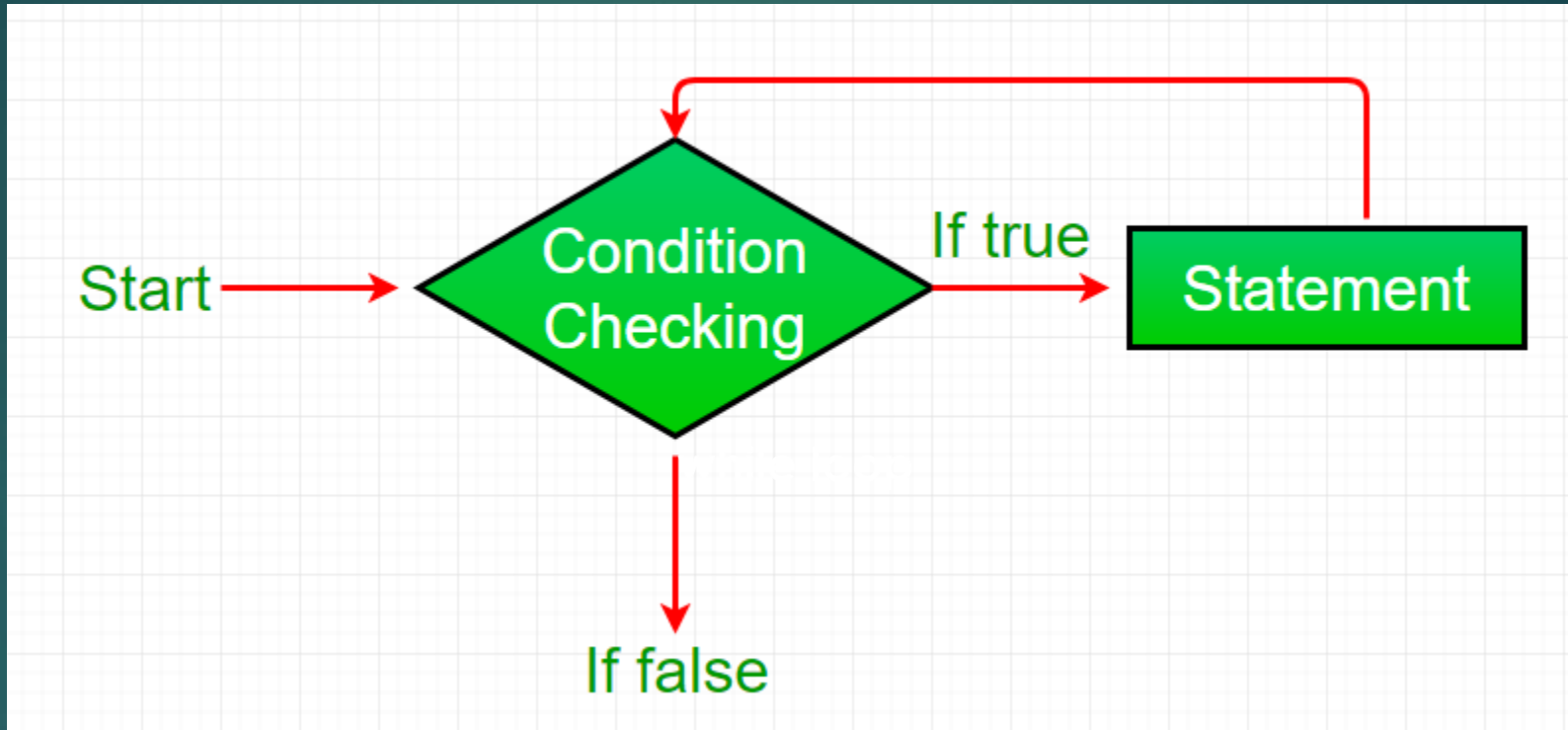
- At the initial execution of the first, 'i' is 12 and thus, 'Table of 12' gets printed.
- Now, the next statement is `for(j = 1;j<=10;j++)`. So, executing this statement:
- 'j' is 1 and  $12*1 = 12$  will be printed.
- Now, the second iteration of the inner loop will take place. 'i' is still 12 but 'j' will be increased to 2. So,  $12*2 = 24$  will be printed.
- Coming to the last iteration of the inner loop, 'i' is still 12, and 'j' will be 10, so  $12*10 = 120$  will be printed.
- Now, coming out of the outer loop, there is no statement after the inner loop, so 'i' will be increased to 13 (`i++`) for the next iteration of the outer loop and 'Table of 13' will be printed and things will be repeated with the value of 'j' equal to 13.

# while loop:


A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

## Syntax :

```
while (boolean condition)
{
    loop statements...
}
```



while loop

- 
- while loop checks whether the condition written in '( )' is true or not.
  - If the condition is found true, then statements written in the body of the while loop i.e., inside the braces { } are executed. Then, again the condition is checked, and if found true then statements in the body of the while loop are executed again.
  - This process continues until the condition becomes false.

Example 1: while loop  
// Print numbers from 1 to 5

```
#include <stdio.h>
int main()
{
    int i = 1;

    while (i <= 5)
    {
        printf("%d\n", i);
        ++i;
    }

    return 0;
}
```

Output

1  
2  
3  
4  
5



Here, we have initialized  $i$  to 1.

When  $i$  is 1, the test expression  $i \leq 5$  is true. Hence, the body of the while loop is executed. This prints 1 on the screen and the value of  $i$  is increased to 2.

Now,  $i$  is 2, the test expression  $i \leq 5$  is again true. The body of the while loop is executed again. This prints 2 on the screen and the value of  $i$  is increased to 3.

This process goes on until  $i$  becomes 6. When  $i$  is 6, the test expression  $i \leq 5$  will be false and the loop terminates.



Example2:

```
#include <stdio.h>
```

```
int main()  
{
```

```
    int i = 5;
```

```
    while (i < 10) {  
        printf("GFG\n");  
        i++;  
    }
```

```
    return 0;  
}
```

Output:

GFG  
GFG  
GFG  
GFG  
GFG