

A close-up, slightly blurred photograph of a laptop. The screen shows a code editor with syntax-highlighted code, likely HTML or JavaScript, with line numbers visible on the left. The keyboard is in the foreground, partially out of focus. A dark semi-transparent banner is overlaid on the lower half of the image, containing the text 'AMIT LEARNING' and 'SESSION 2 CONTROL FLOW'. The 'AMIT' logo is in the bottom right corner.

AMIT LEARNING

SESSION 2 CONTROL FLOW

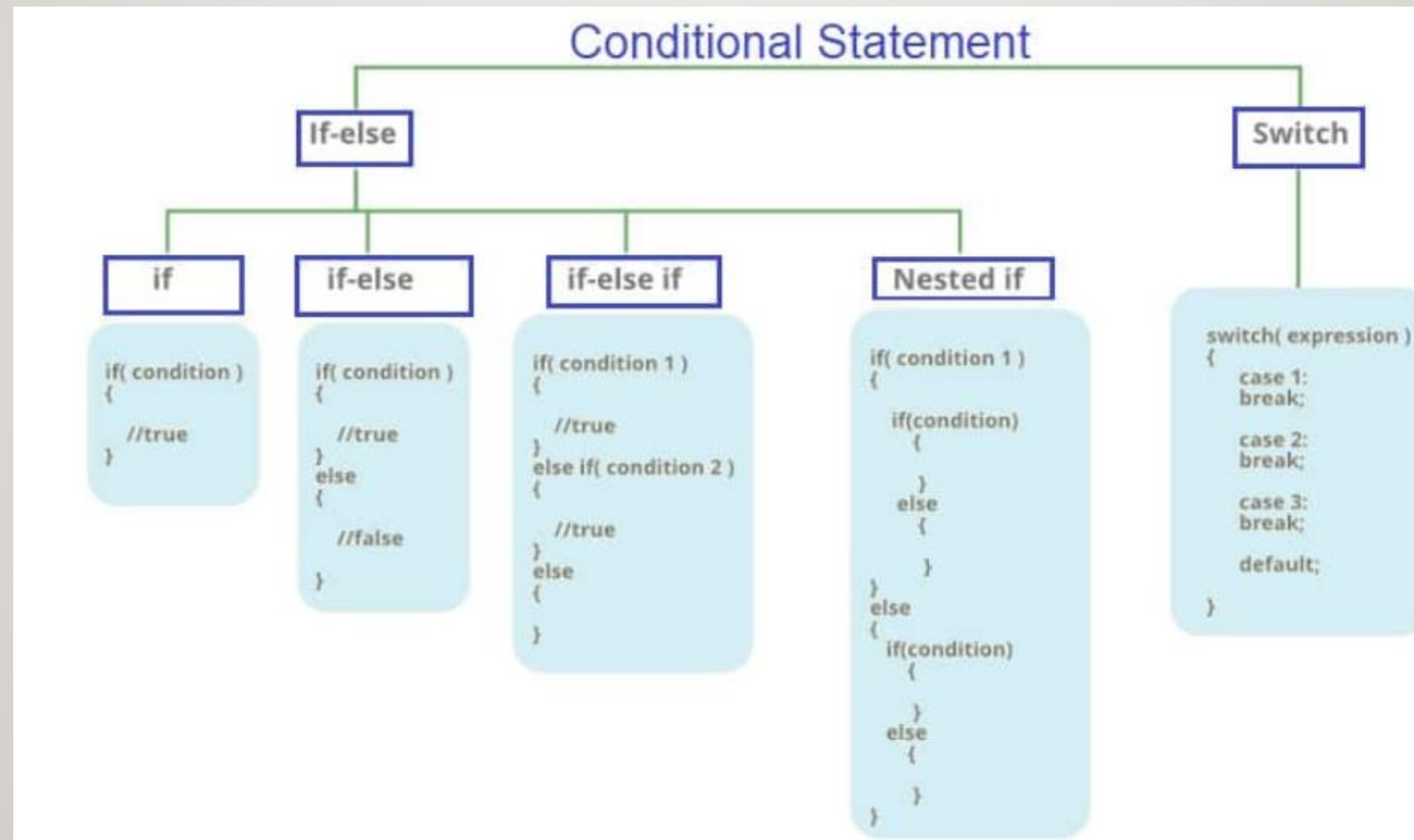
CONTENTS

1	WHAT IS Conditional statement	1
1.1	If statement	1
1.2	If else statement	1
1.3	Nested if statements	1
1.4	Ifelse if statements	1
1.5	Switch statements	1
2	LOOP IN C	2
2.1	Iterative method	2
2.2	for loop .	2
2.3	While loop.	2
2.4	Do....while loop	2
2.5	Infinite loop	
2.6	Jump Statement :Break, Continue ,go to	

1 What is conditional statement

1 What is conditional statement

- There come situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next.



1

What is conditional statement

1.1 If statement

- if statement is the most simple decision making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

- **Syntax:**

```
if(condition)
{
    // Statements to execute if
    // condition is true
}
```

- **Example:**

```
if(condition)
    statement1;
    statement2;

// Here if the condition is true, if block
// will consider only statement1 to be inside
// its block.
```

1 What is conditional statement

1.1 If statement

- if statement is the most simple decision making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

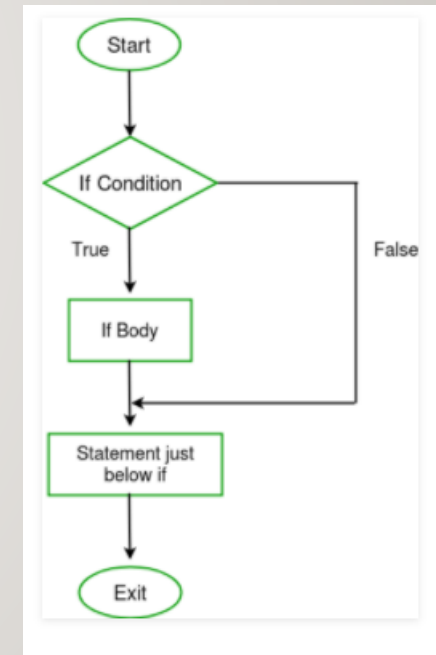
- **Syntax:**

```
if(condition)
{
    // Statements to execute if
    // condition is true
}
```

- **Example:**

```
if(condition)
    statement1;
    statement2;

// Here if the condition is true, if block
// will consider only statement1 to be inside
// its block.
```

Flowchart

1

What is conditional statement

1.1 If statement

CODE

```
// C program to illustrate If statement
#include <stdio.h>

int main() {
    int i = 10;

    if (i > 15)
    {
        printf("10 is less than 15");
    }

    printf("I am Not in if");
}
```

OUTPUT

I am Not in if

DESCRIPTION

- In this example check condition if condition true the statement will executed .if condition is false the if will not executed

1 What is conditional statement

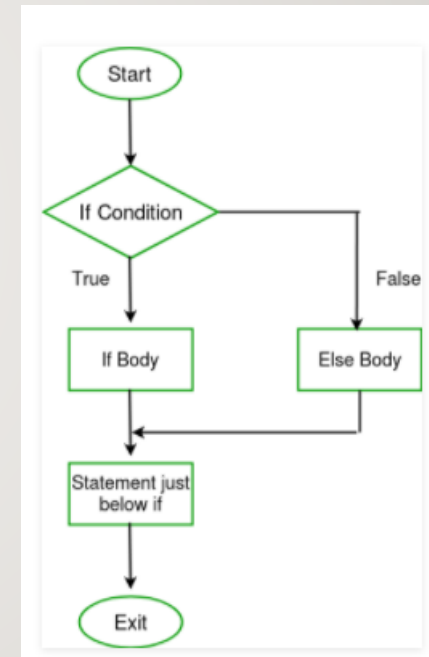
1.1 If statement

➤ The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false.

➤ Syntax:

```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```

Flowchart



1

What is conditional statement

1.2 If ...else statement

CODE

```
// C program to illustrate If statement
#include <stdio.h>

int main() {
    int i = 20;

    if (i < 15)
        printf("i is smaller than 15");
    else
        printf("i is greater than 15");

    return 0;
}
```

OUTPUT

i is greater than 15

DESCRIPTION

- In this example check condition if condition true the statement will executed .if condition is false the else will executed

1 What is conditional statement

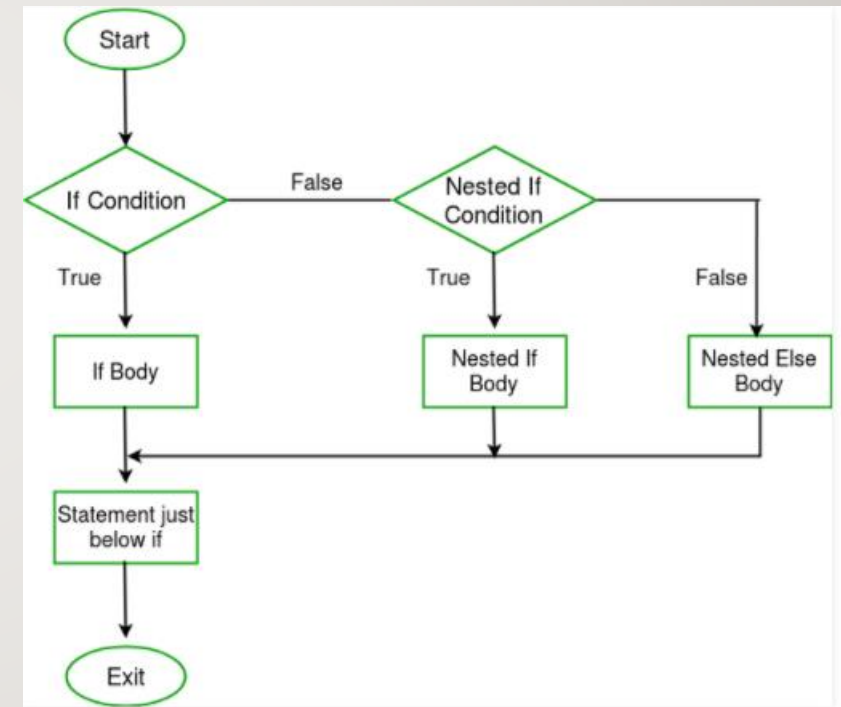
1.3 Nested -if statement

A nested if in C is an if statement that is the target of another if statement. Nested if statements means an if statement inside another if statement.

➤ Syntax:

```
if (condition1)
{
    // Executes when condition1 is true
    if (condition2)
    {
        // Executes when condition2 is true
    }
}
```

Flowchart



1

What is conditional statement

1.3 Nested -if statement

CODE

```
// C program to illustrate nested-if statement
#include <stdio.h>

int main() {
    int i = 10;

    if (i == 10)
    {
        // First if statement
        if (i < 15)
            printf("i is smaller than 15\n");

        // Nested - if statement
        // Will only be executed if statement above
        // is true
        if (i < 12)
            printf("i is smaller than 12 too\n");
        else
            printf("i is greater than 15");
    }

    return 0;
}
```

OUTPUT

```
i is smaller than 15
i is smaller than 12 too
```

DESCRIPTION

- In this example if the first condition true the nested if will executed .if first condition is false will never executed any code.

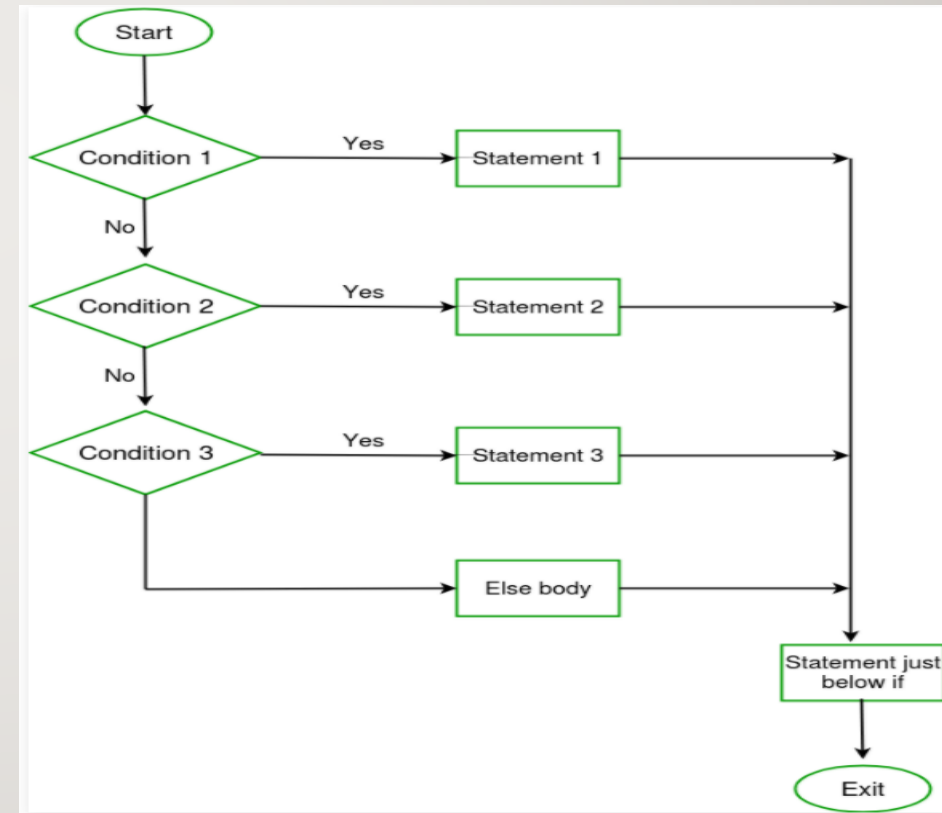
1 What is conditional statement

1.4 Ifelse if statements

- Here, a user can decide among multiple options. The C if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed
- Syntax:

```
if (condition)
    statement;
else if (condition)
    statement;
.
.
else
    statement;
```

Flowchart



1

What is conditional statement

1.4 Ifelse if statements

CODE

```
// C program to illustrate nested-if statement
#include <stdio.h>

int main() {
    int i = 20;

    if (i == 10)
        printf("i is 10");
    else if (i == 15)
        printf("i is 15");
    else if (i == 20)
        printf("i is 20");
    else
        printf("i is not present");
}
```

OUTPUT

```
i is 20
```

DESCRIPTION

- In this example if the first condition true and rest bypassed .

1

What is conditional statement

1.5 Switch statements

- Switch case statements are a substitute for long if statements that compare a variable to several integral values.
- The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
- Switch is a control statement that allows a value to change control of execution.
- Duplicate case values are not allowed.
- The default statement is optional. Even if the switch case statement do not have a default statement, it would run without any problem.
- The break statement is used inside the switch to terminate a statement sequence. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.
- Nesting of switch statements are allowed, which means you can have switch statements inside another switch. However nested switch statements should be avoided as it makes program more complex and less readable

1 What is conditional statement

1.5 Switch statements

➤ Syntax:

```
switch (n)
{
    case 1: // code to be executed if n = 1;
        break;
    case 2: // code to be executed if n = 2;
        break;
    default: // code to be executed if n doesn't match any cases
}
```

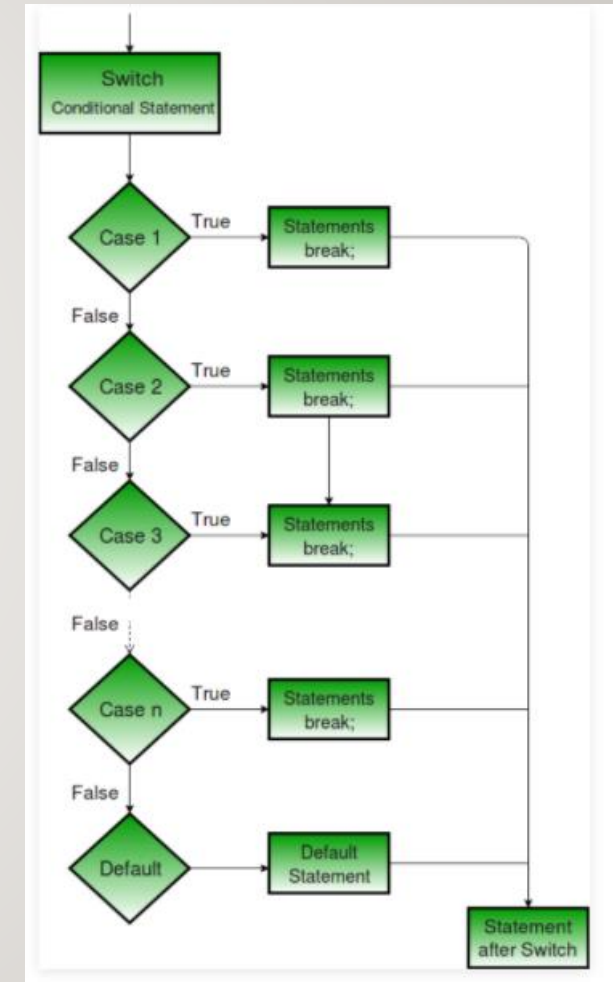
- The expression provided like `n` in the switch should result in a **constant value** otherwise it would not be valid.

Valid expressions for switch: `switch(a*b+c*d)` or `(a||b)` or `(1+2+23)`.

- The expression used in switch must be integral type (int, char and enum). Any other type of expression is not allowed.

- **The statements written above cases are never executed** After the switch statement, the control transfers to the matching case, the statements written before case are not executed.

Flow control



1

What is conditional statement

1.5 Switch statements

CODE

```
int main()
{
    int x = 2;
    switch (x)
    {
        case 1: printf("Choice is 1");
                break;
        case 2: printf("Choice is 2");
                break;
        case 3: printf("Choice is 3");
                break;
        default: printf("Choice other than 1, 2 and 3");
                break;
    }
    return 0;
}
```

OUTPUT

Choice is 2

DESCRIPTION

- In this example x is equal 2 so will jump to case 2 and print statement.

1

What is conditional statement

1.5 Nested Switch statements

CODE

```
int main () {  
  
    /* local variable definition */  
    int a = 100;  
    int b = 200;  
  
    switch(a) {  
  
        case 100:  
            printf("This is part of outer switch\n", a );  
  
            switch(b) {  
                case 200:  
                    printf("This is part of inner switch\n", a );  
            }  
    }  
  
    printf("Exact value of a is : %d\n", a );  
    printf("Exact value of b is : %d\n", b );  
  
    return 0;  
}
```

OUTPUT

```
This is part of outer switch  
This is part of inner switch  
Exact value of a is : 100  
Exact value of b is : 200
```

DESCRIPTION

- In this example switch a and b will be executed .

1

What is conditional statement

1.6 Lab

➤ Program to Assign grades to a student using Nested If Else:

Percentage	Grade
90 and above	A
80 to 89	B
60 to 79	C
33 - 59	D
below 33	F

1 What is conditional statement

1.6 Lab

➤ Program to Assign grades to a student using Nested If Else:

```
// Nested if else
if (percentage >= 90)
{
    grade = 'A';
}
else
{
    if (percentage >= 80 && percentage <= 89)
    {
        grade = 'B';
    }
    else
    {
        if (percentage >= 60 && percentage <= 79)
        {
            grade = 'C';
        }
        else
        {
            if (percentage >= 33 && percentage <= 59)
            {
                grade = 'D';
            }
            else
            {
                grade = 'F';
            }
        }
    }
}

printf("%c",grade);
}
```

- 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:
An iterative method to do this is to write the `printf()` statement 10 times.

2

Iterative method

2.1 Iterative method

CODE

```
int main()
{
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");

    return 0;
}
```

OUTPUT

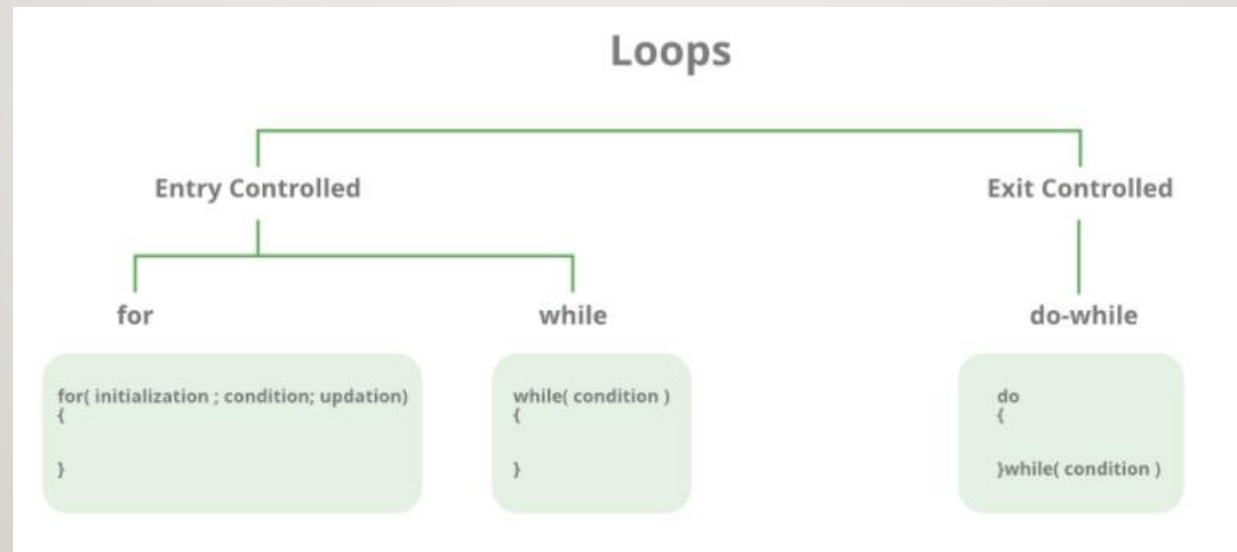
```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

DESCRIPTION

- In this example hello world print 10 times

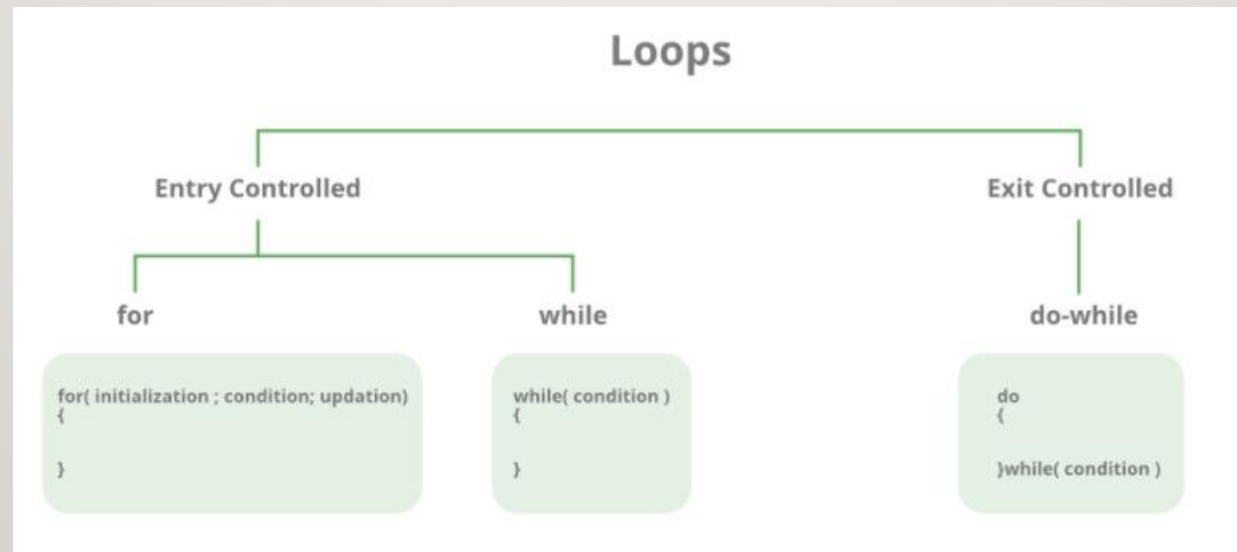
➤ There are mainly two types of loops:

- **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 at least once, irrespective of whether the test condition is true or false. do – while loop is exit controlled 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 at least once, irrespective of whether the test condition is true or false. do – while loop is exit controlled loop.

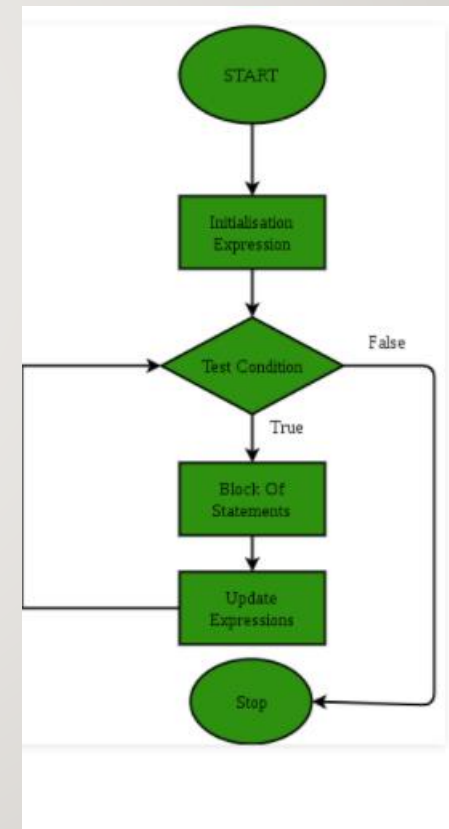


➤ A for loop is a repetition control structure which allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.

➤ **Syntax:**

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

Flowchart



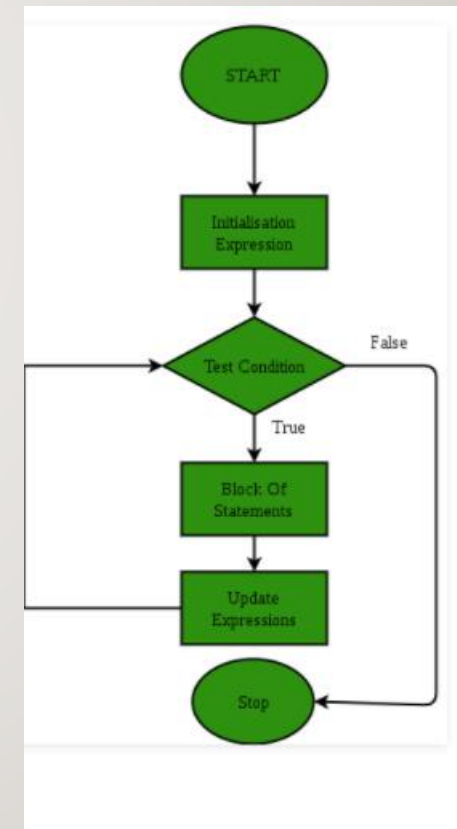
- A for loop is a repetition control structure which allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.

- **Syntax:**

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

- **Initialization Expression:** In this expression we have to initialize the loop counter to some value. for example: `int i=1;`
- **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;`
- **Update Expression:** After executing loop body this expression increments/decrements the loop variable by some value. for example: `i++;`

Flowchart



2

For loop

2.1 For loop

CODE

```
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
```

DESCRIPTION

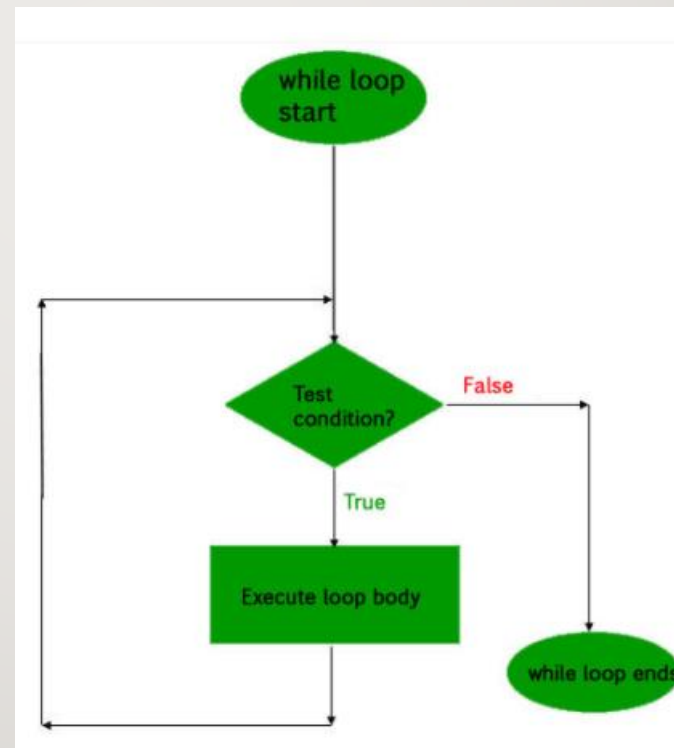
- In this example hello world print 10 times

- While studying **for loop** we have seen that the number of iterations is known beforehand, i.e. the number of times the loop body is needed to be executed is known to us. while loops are used in situations where we do not know the exact number of iterations of loop beforehand.

- **Syntax:**

```
initialization expression;  
while (test_expression)  
{  
    // statements  
  
    update_expression;  
}
```

Flowchart



2

while loop

2.3 while loop

CODE

```
int main()
{
    // initialization expression
    int i = 1;

    // test expression
    while (i < 6)
    {
        printf( "Hello World\n");

        // update expression
        i++;
    }

    return 0;
}
```

OUTPUT

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

DESCRIPTION

- In this example hello world print 6 times

2

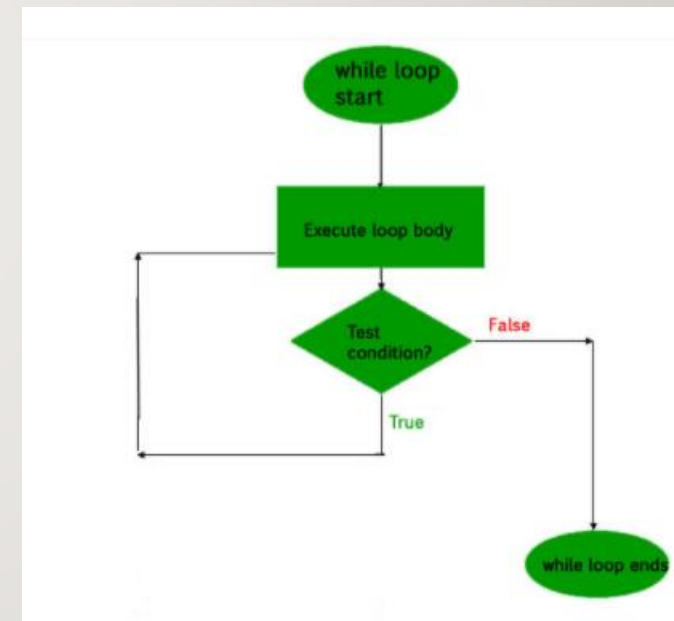
While loop

2.4 While loop

- In do while loops also the loop execution is terminated on the basis of test condition. The main difference between do while loop and while loop is in do while loop the condition is tested at the end of loop body. In do while loop the loop body will execute at least once irrespective of test condition. Notice the semi – colon(“;”) in the end of loop.

- **Syntax:**

```
initialization expression;  
do  
{  
    // statements  
  
    update_expression;  
} while (test_expression);
```

flowchart

2

Do....while loop

2.4 Do...while loop

CODE

```
int main()
{
    int i = 2; // Initialization expression

    do
    {
        // loop body
        printf( "Hello World\n");

        // update expression
        i++;

    } while (i < 1); // test expression

    return 0;
}
```

OUTPUT

Hello World

DESCRIPTION

- In this example hello world print 1 times

- An infinite loop (sometimes called an endless loop) is a piece of coding that lacks a functional exit so that it repeats indefinitely. An infinite loop occurs when a condition always evaluates to true. Usually, this is an error
- we put the semicolon after the condition of the while loop which leads to the infinite loop. Due to this semicolon, the internal body of the while loop will not execute.
- We should check the logical conditions carefully. Sometimes by mistake, we place the assignment operator (=) instead of a relational operator (==).

```
for(;;)
{
    // body of the for loop.
}
```

```
infinite_loop;
// body statements.
goto infinite_loop;
```

```
while(1)
{
    // body of the loop..
}
```

```
int main()
{
    int i=1;
    while(i<=10);
    {
        printf("%d", i);
        i++;
    }
    return 0;
}
```

```
#define infinite for(;;)
int main()
{
    infinite
    {
        printf("hello");
    }

    return 0;
}
```

```
int main()
{
    char ch='n';
    while(ch='y')
    {
        printf("hello");
    }
    return 0;
}
```

2

Infinite loop

2.5 Infinite loop

- We use the wrong loop condition which causes the loop to be executed indefinitely.
- We should be very careful when we are using the floating-point value inside the loop as we cannot underestimate the floating-point errors.
- We should be careful when we are using the **break** keyword in the nested loop because it will terminate the execution of the nearest loop, not the entire loop.

```
#include <stdio.h>

int main()
{
    for(int i=1;i>=1;i++)
    {
        printf("hello");
    }
    return 0;
}
```

```
#include <stdio.h>

int main()
{
    while(1)
    {
        for(int i=1;i<=10;i++)
        {
            if(i%2==0)
            {
                break;
            }
        }
    }
    return 0;
}
```

```
#include <stdio.h>

int main()
{
    float x = 3.0;
    while (x != 4.0) {
        printf("x = %f\n", x);
        x += 0.1;
    }
    return 0;
}
```

2 Jump Statement :Break, Continue ,go to, return

2.6 Jump Statement :Break, Continue ,go to, return

➤ These statements are used in for unconditional flow of control through out the functions in a program.They support four type of jump statements:

1. Break.
2. Continue.
3. Go to .
4. Return.

1.Break:This loop control statement is used to terminate the loop.As soon as the break statement is encountered from within a loop, the loop iterations stops there and control returns from the loop immediately to the first statement after the loop.

➤ **Syntax:**

```
break;
```

```
while (testExpression) {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
}
```

```
do {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
} while (testExpression);
```

```
for (init; testExpression; update) {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
}
```

2 Jump Statement :Break, Continue ,go to, return

2.6 Jump Statement :Break, Continue ,go to, return

2. Continue: it forces to execute the next iteration of the loop.

➤ **Syntax:**

```
continue;
```

```
while (testExpression) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```

```
do {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
} while (testExpression);
```

```
for (init; testExpression; update) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```

2 Jump Statement :Break, Continue ,go to, return

2.6 Jump Statement :Break, Continue ,go to, return

3.The go to statement in C also referred to as unconditional jump statement can be used to jump from one point to another within a function.

➤ Syntax:

Syntax1		Syntax2

goto label;		label:
.		.
.		.
.		.
label:		goto label;

2 Jump Statement :Break, Continue ,go to, return

2.6 :Break

CODE

```
#include<stdio.h>
int main()
{
    int i;
    for(i=0;i<5;i++)
    {
        if(i==2)
            break;
        else
            printf("%d\n",i);
    }
    return 0;
}
```

OUTPUT



```
0
1
```

DESCRIPTION

- In this example break will terminate from loop when i equal 2


2 Jump Statement :Break, Continue ,go to, return

2.6 Continue

CODE

```
int main()  
{  
  
    int i;  
    for(i=0;i<5;i++)  
    {  
        if(i==2)  
            continue;  
        else  
            printf("%d\n",i);  
    }  
    return 0;  
}
```

OUTPUT



```
0  
1  
3  
4
```

DESCRIPTION

- In this example continue will skip one iteration from loop when i equal 2

2 Jump Statement :Break, Continue ,go to, return

2.6 goto

CODE

```
int main()
{
    int n = 1;
label:
    printf("%d ",n);
    n++;
    if (n <= 10)
        goto label;
return 0;
}
```

OUTPUT

```
1 2 3 4 5 6 7 8 9 10
Process returned 0 (0x0)   execution time : 0.060 s
Press any key to continue.
```

DESCRIPTION

- In this example goto will print 1 to 10