

Name of Teacher: Miss Radhika M. Patil

Class: B.Sc. Computer Science (Entire)- II **Semester :** 3

Course Title: Object Oriented Programming using C++

Control Statements:

Control statements are used to control the flow of execution of program. This execution order depends on supply data values and conditional logic.

C++ contains following types of control statements:

1. Conditional

- a) if
- b) if else
- c) Switch

2. Unconditional

- a) break
- b) continue
- c) goto

3. Looping

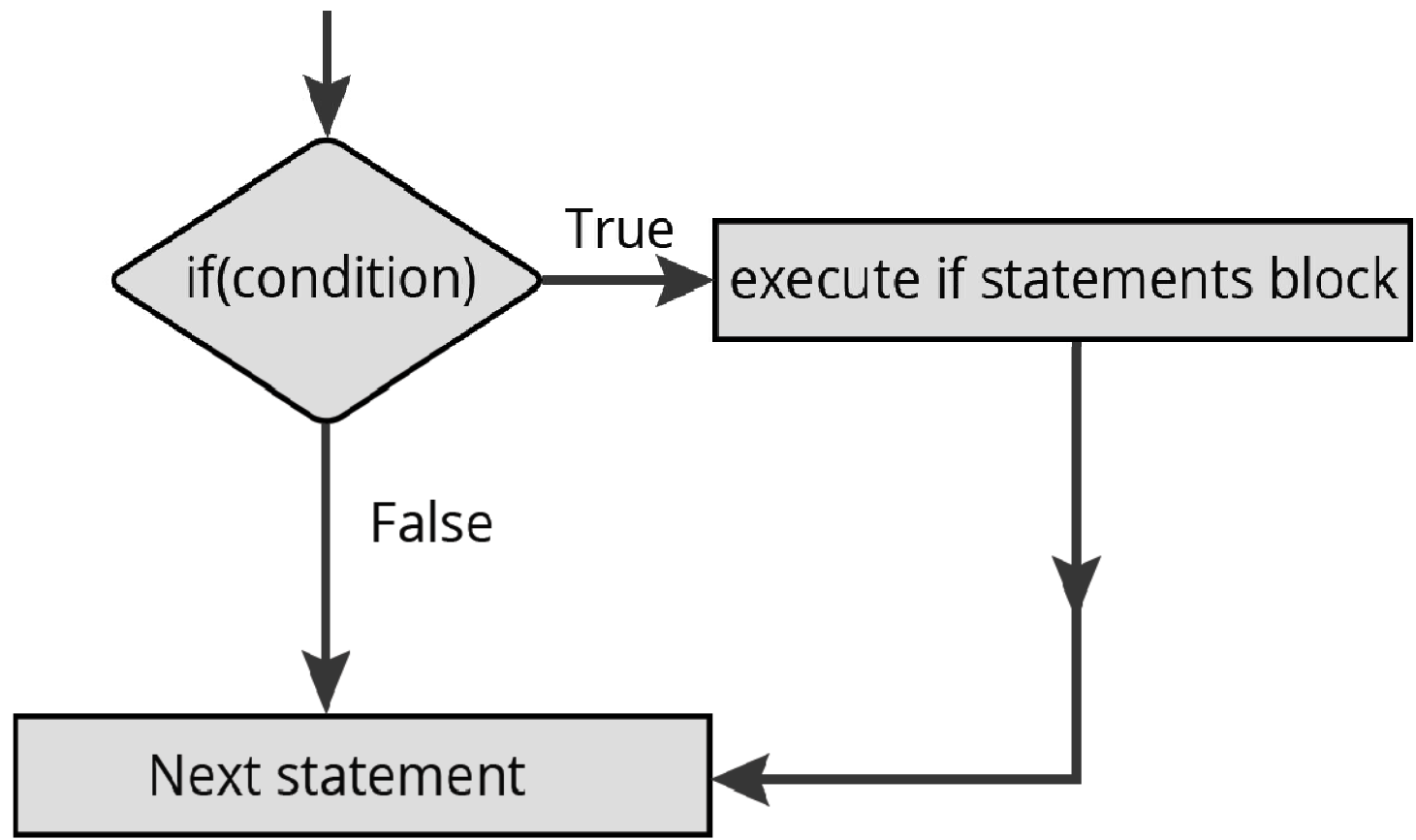
- a) while
- b) do while
- c) for loop

If statement

This control statement is used to execute single statement or block of code when given condition is true and if condition is false then it skips if block and rest code of program will be executed.

Syntax:

```
if (conditional expression)
{
// statements;
}
```

Program

```
void main()
{
    int num;
    cout << "Enter an integer: ";
    cin >> num;
    if ( num > 0)
    {
        cout << "Number is positive " ;
    }
    getch();
}
```

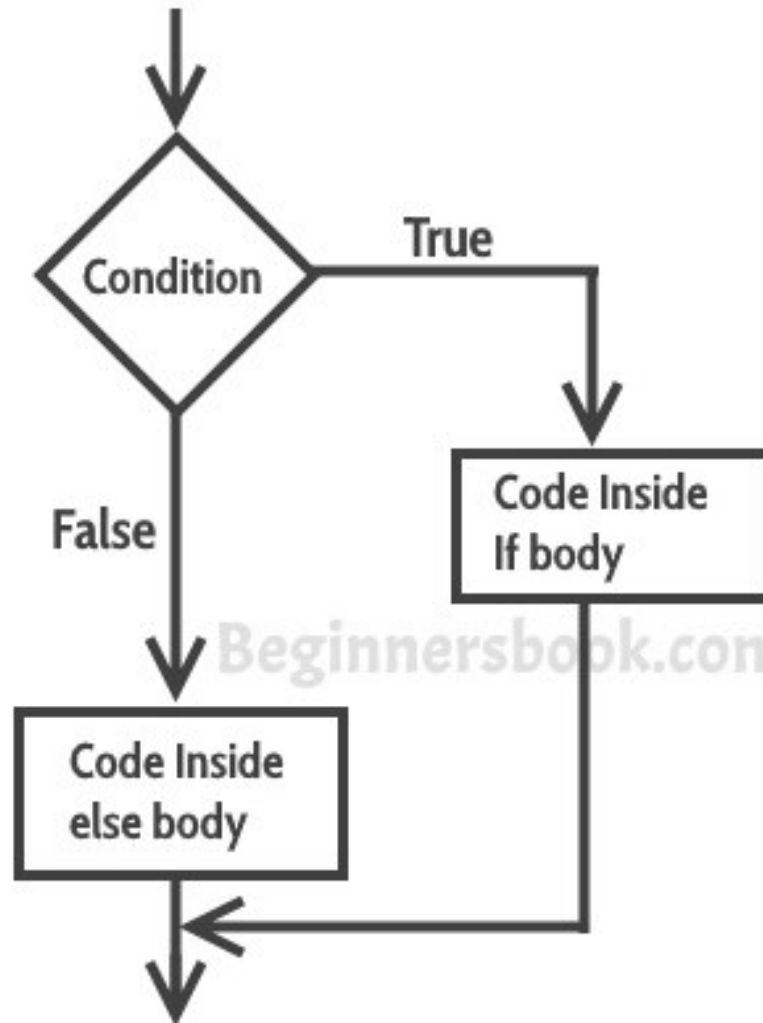
if...else statement

It is an extension of if statement. If condition is true then statements in if block will be executed . Otherwise statements in else block will be executed.

In either case, either true or false block will be executed but not both.

Syntax

```
if (condition)
{
// statements to be executed if condition is true ;
}
else
{
// statement to be executed if condition is false;
}
```

Beginnersbook.com

Example

```
void main()
{
    int number;
    cout << "Enter an integer: ";
    cin >> num;
    if ( num > 0)
    {
        cout << "You entered a positive integer: " << num << endl;
    }
    else
    {
        cout << "You entered a negative integer: " << num << endl;
    }
    getch();
}
```


Example:

```
void main( )
{
    int x,y;
    x=15;
    y=18;
    if (x > y )
    {
        cout << "x is greater than y";
    }
    else
    {
        cout << "y is greater than x";
    }
}
```

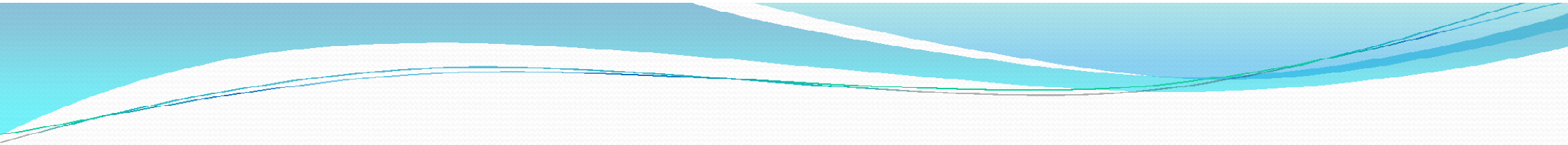
Output:
y is greater than x

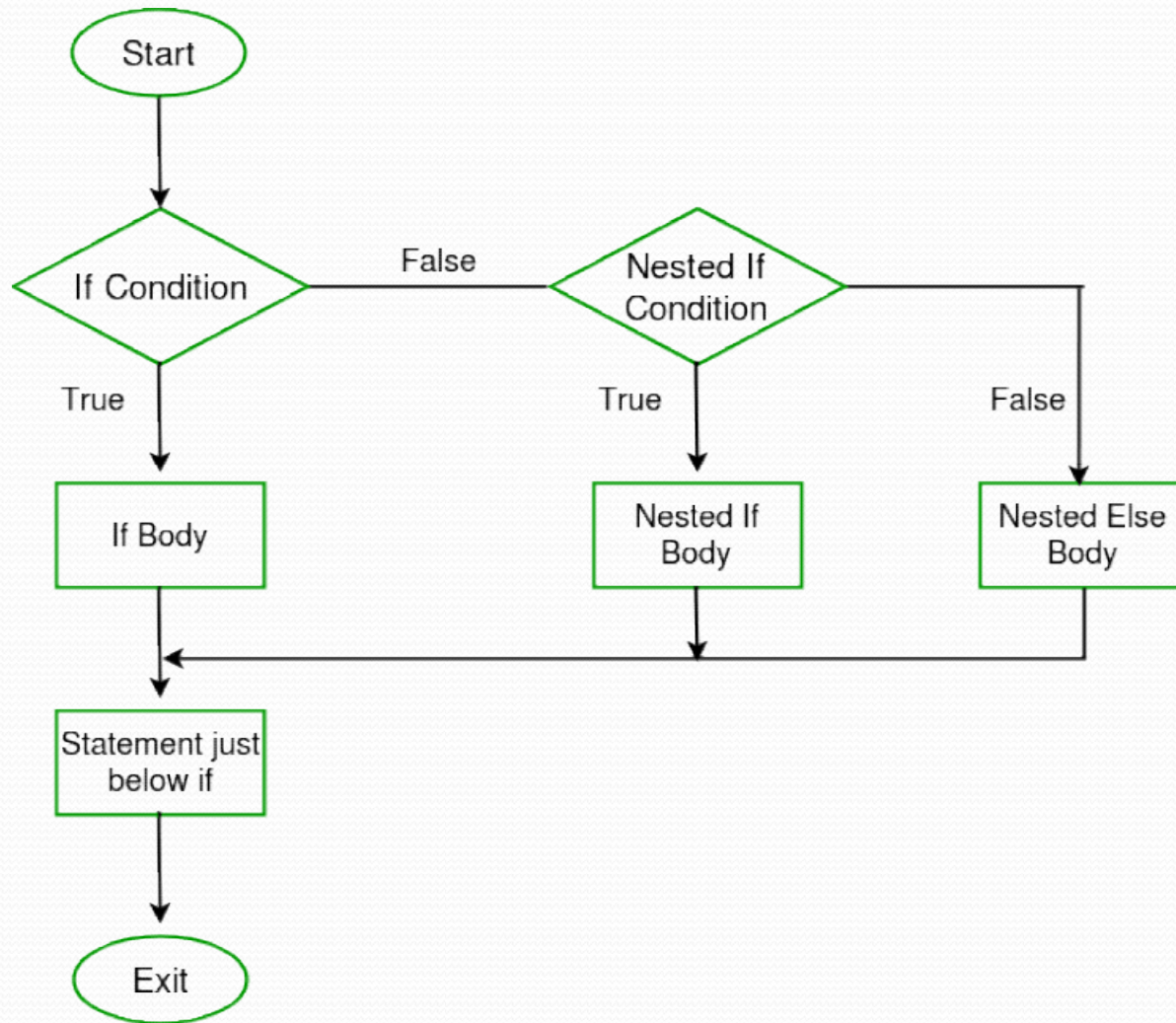
Nested if....else statement

When we need to check more than one condition then we use multiple if else statements i.e. nested if else statement

Syntax:

```
if(expression)
{
    if(expression1)
    {
        statement-block1;
    }
    else
    {
        statement-block2;
    }
}
else
{
    statement-block3;
}
```

- 
- if 'expression' is **True**, then the control enters into the main if block .Inside the if block expression₁ is evaluated ,if it is true the statement-block₁ will be executed otherwise statement-block₃ will be executed.
 - Otherwise statement-block₃ will be executed.



Example

```
void main( )
{
    int a,b,c;
    cout << "enter 3 number";
    cin >> a >> b >> c;
    if(a > b)
    {
        if( a > c)
        {
            cout << "a is greatest";
        }
        else
        {
            cout << "c is greatest";
        }
    }
    else
    {
        if( b> c)
        {
            cout << "b is greatest";
        }
        else
        {
            cout << "c is greatest";
        }
    }
}
```


if-else-if ladder in C++

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 else-if ladder is bypassed. If none of the conditions are true, then the final else statement will be executed.

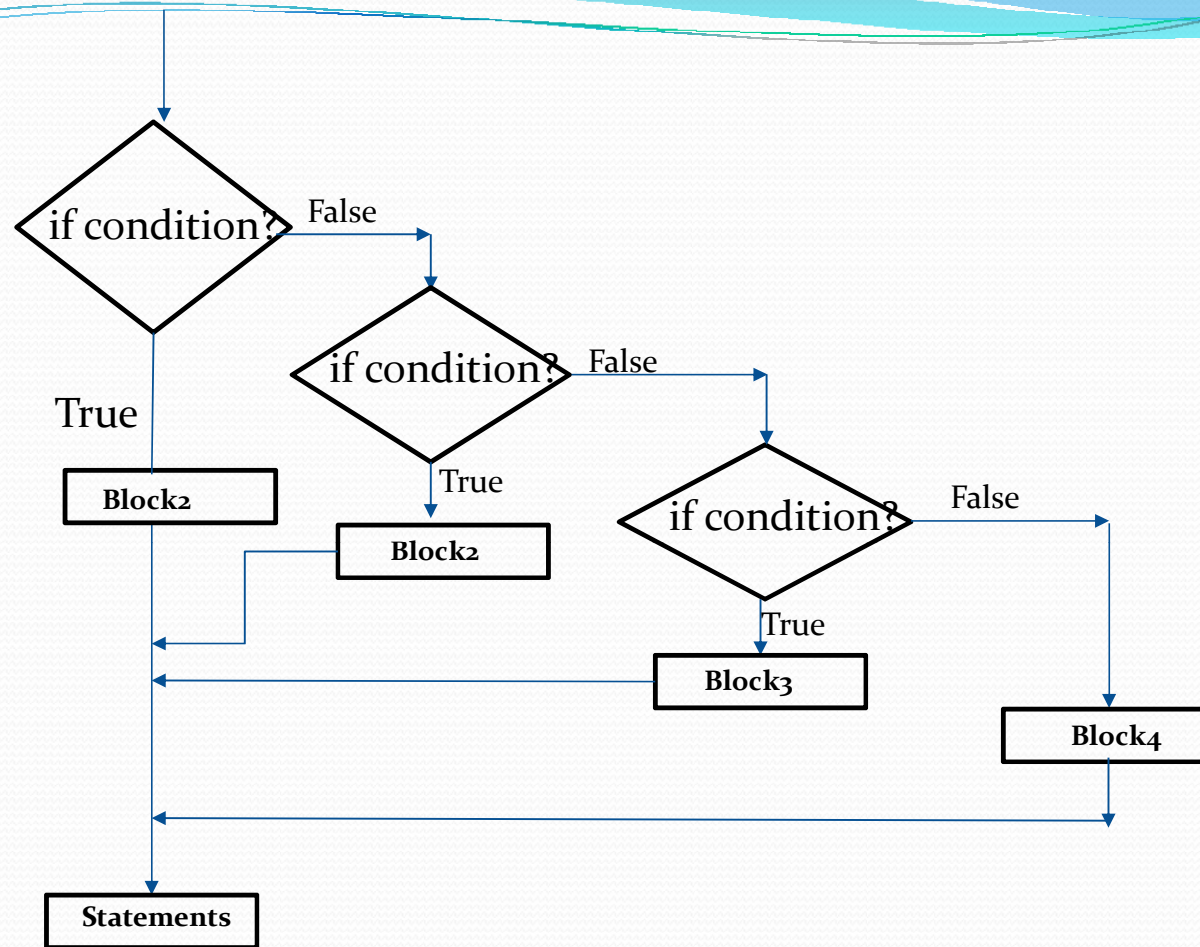
Syntax:

```
if (condition1)  
{  
    //Block 1;  
}  
else if (condition2)  
{  
    //Block2;  
}  
.  
.  
else  
{  
    //Block3;  
}
```




Here,

- If condition1 evaluates to true, the code block 1 is executed.
- If condition1 evaluates to false, then condition2 is evaluated.
- If condition2 is true, the code block 2 is executed.
- If condition2 is false, the code block 3 is executed.




```
void main()
{
    int number;
    cout << "Enter an integer: ";
    cin >> number;
    if (number > 0)
    {
        cout << "You entered a positive integer: " << number << endl;
    }
    else if (number < 0)
    {
        cout << "You entered a negative integer: " << number << endl;
    }
    else
    {
        cout << "You entered 0." << endl;
    }
    cout << "This line is always printed.";
    getch();
}
```


C++ Switch Statements

Use the switch statement to select one of many code blocks to be executed.

Syntax

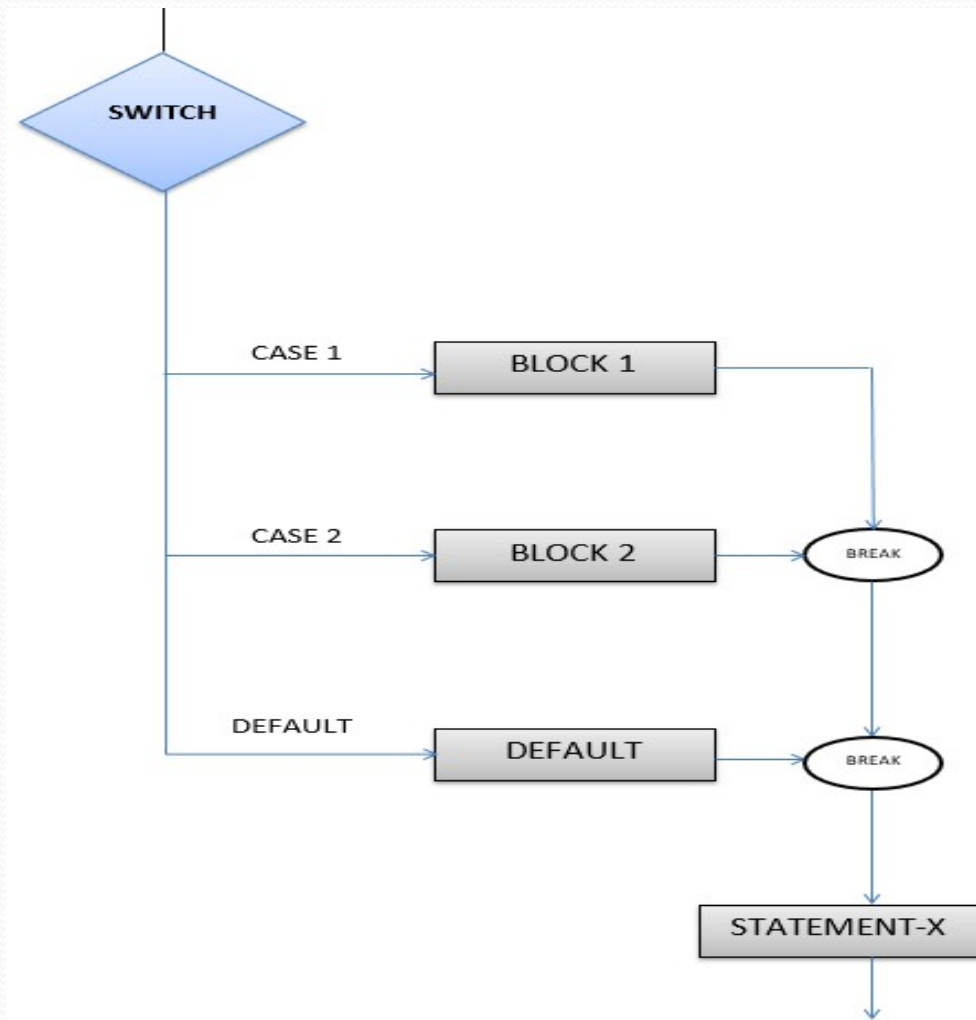
```
switch (expression)
{
    case constant1:
        // code to be executed if expression is equal to constant1;
        break;
    case constant2:
        // code to be executed if expression is equal to constant2;
        break;
    .
    .
    .
    default:
        // code to be executed if expression doesn't match any constant
}
```



How does the switch statement work?

The expression is evaluated once and compared with the values of each case label.

- If there is a match, the corresponding code after the matching label is executed.
- For example, if the value of the variable is equal to constant2,
then code after case constant2: is executed until the break statement is encountered.
- If there is no match, the code after default: is executed.




```
void main()
{
    int day = 4;
    switch (day)
    {
        case 1:
            cout << "Monday";
            break;
        case 2:
            cout << "Tuesday";
            break;
        case 3:
            cout << "Wednesday";
            break;
        case 4:
            cout << "Thursday";
            break;
        case 5:
            cout << "Friday";
            break;
        case 6:
            cout << "Saturday";
            break;
    }
}
```

```
case 7:
    cout << "Sunday";
    break;
default: cout<<"\n Invalid choice";
}
getch();
}
```


Looping Statement

- Loop statements are used to repeat execution of statements or blocks.
- Repeating the execution of same code fragment several times is called **iteration**.
- Basic logic behind iteration is that a sequence of statements is repeated until a certain condition is satisfied.
- When this condition is false then iteration terminates.

Following are various looping or iterative statements used in C++.

- a) while loop
- b) do while loop
- c) for loop

a) while loop

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

Syntax:

initialization of loop control variable;

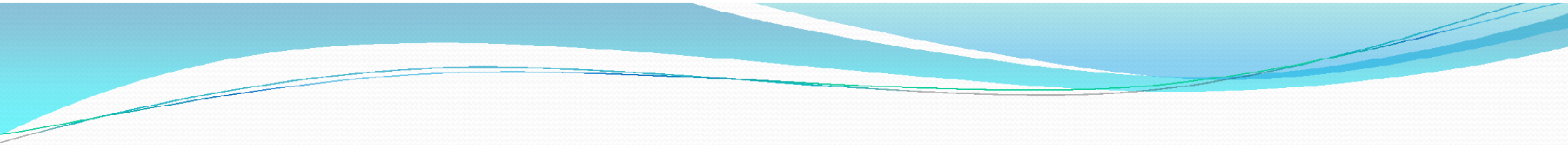
while (condition)

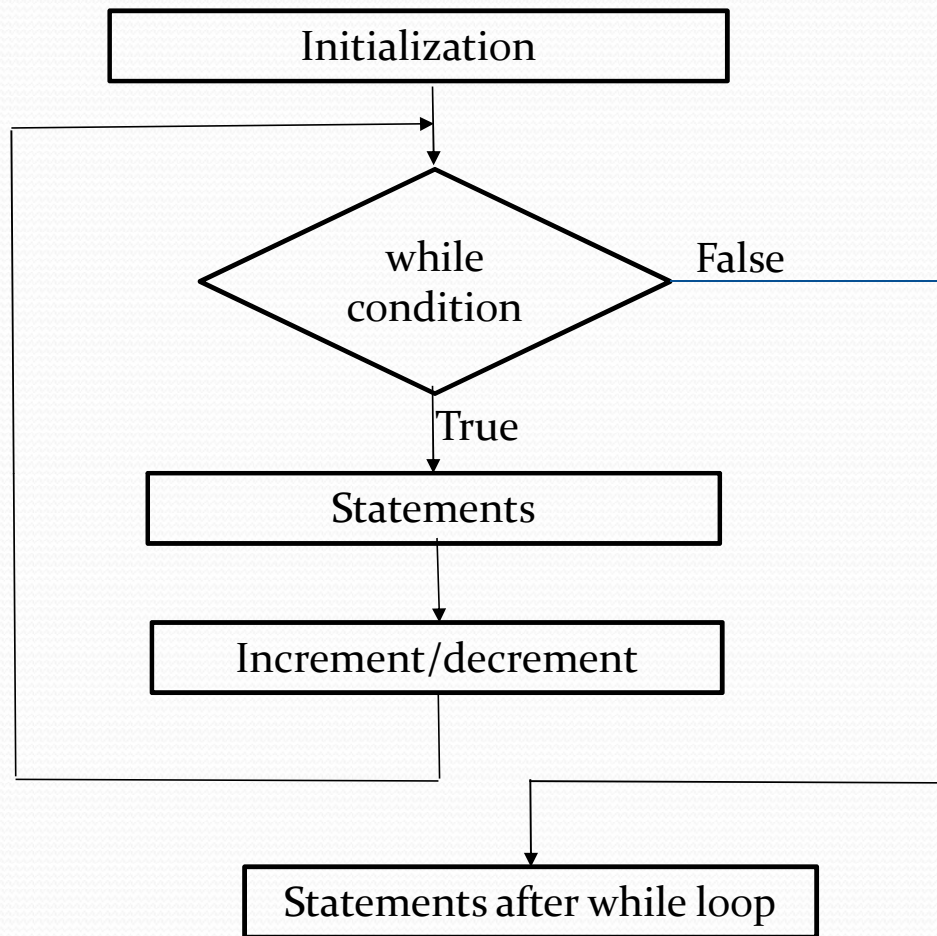
{

statements;

increment/decrement;

}

- 
- Condition may be any expression.
 - Loop iterates when condition is true, when condition becomes false then control passes to the line immediately after the loop.
 - Key point of while loop is that loop might not ever run when condition is tested and if the result is false then the loop body will be skipped and first statement after while loop will be executed.
 - There must be increment or decrement statement inside the while block that should make the evaluation of result. Otherwise infinite loop is created.



Program:

Program to print 1 to 10 numbers using while loop.

```
void main()
{
    int i;
    i=1;
    while (i<=10)
    {
        cout<<i<<" "<<endl;
        i++;
    }
    cout<<" Loop terminated";
    getch();
}
```



b) do while loop

- do while loop works like while where group of statements is executed as long as condition is true.
- If condition becomes false then flow of execution goes out of do while loop.
- Main difference between while and do while loop is that **in do while** loop, statements are executed **before evaluating conditional expression** where as **in while** loop, **condition is checked first** and then statements are executed.



Syntax:

initialization of loop control variable;

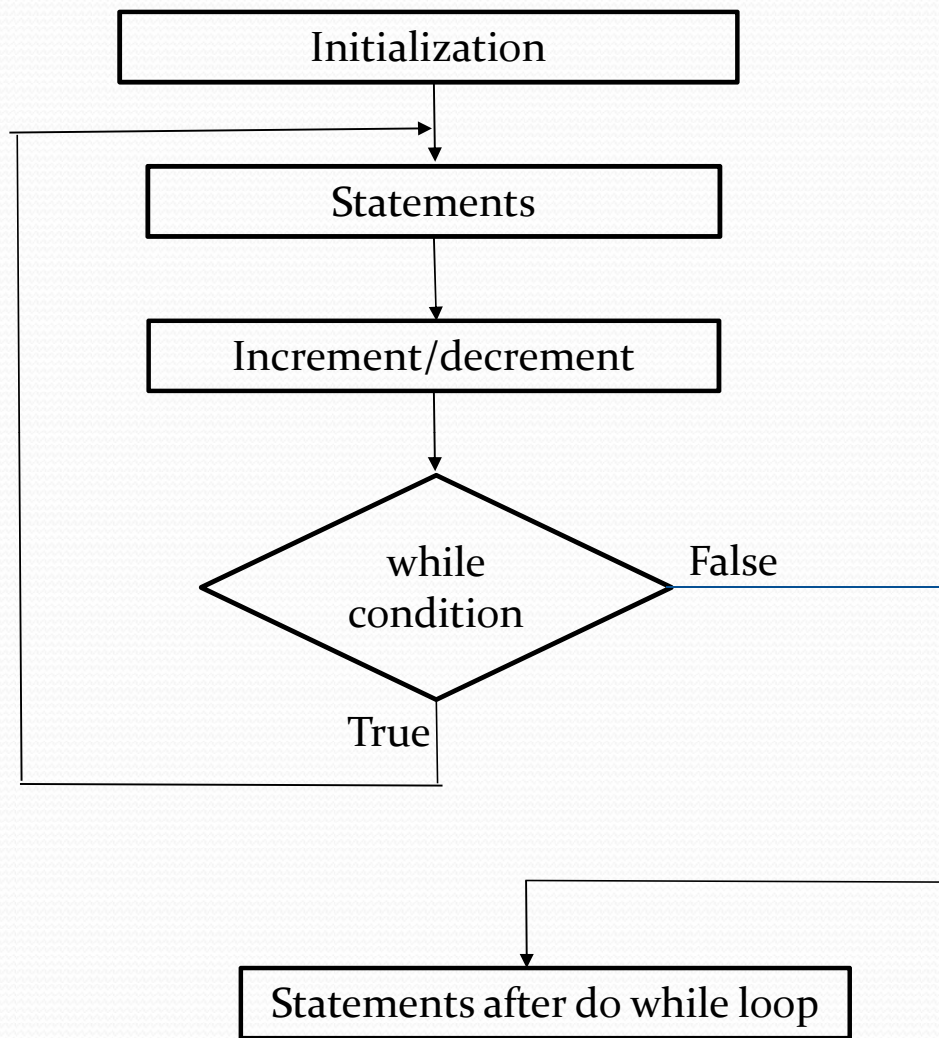
do

{

 statements;

 increment/decrement;

} while (condition);



Program:

Program to print 1 to 10 numbers using do while loop.

```
void main()
{
    int i=1;
    do
    {
        cout<<i<<" ";
        i++;
    }while (i<=10);

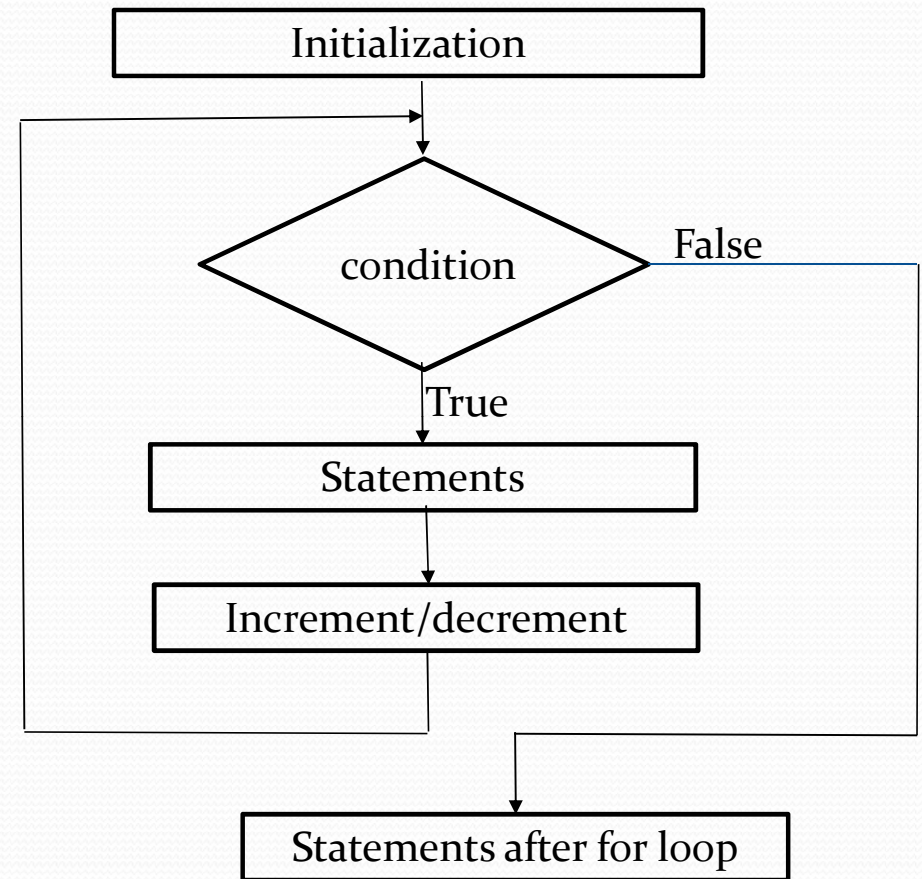
    cout<<"\n Loop terminated";
    getch();
}
```

c) for loop

for loop is entry controlled loop.

Syntax:

```
for (initialization; condition; increment/decrement)
{
    Block of statements;
}
```





Execution of for loop

- Initialization of loop control variable is done first using assignment statements such as $i=0$, variable i is known as loop control variable.
- Value of loop control variable is tested using test condition. If condition is true, body of loop is executed otherwise loop is terminated and execution continues with statements that immediately follow the loop.
- When body of the loop is executed, control is transferred back to the for statement after evaluating last statement in the loop. Now the control variable is incremented or decremented and **new value of control variable is again tested to see whether it satisfies loop condition.**

Program:

Program to print 1 to 10 numbers using for loop.

```
void main()
{
    int i;
    for ( i=1; i<=10; i++)
    {
        cout<<i<<" ";
    }
    cout<<"\n Loop terminated";
    getch();
}
```




- **Nested for loop**

- **Nested loop** means a **loop statement** inside another loop statement.
- That's why nested loops are also called as “**loop inside loop**“
- C++ allows at least 256 levels of nesting.

Syntax for Nested For loop:

```
for ( initialization; condition; increment )
{
    for ( initialization; condition; increment )
    {
        // statement of inner for loop;
    }
    // statement of outer for loop;
}
```

Nested for loop Program:

Program to display 7 days of 3 weeks

```
void main()
{
    int weeks = 3, days_in_week = 7;

    for (int i = 1; i <= weeks; i++)
    {
        cout << "Week: " << i << endl;
        for (int j = 1; j <= days_in_week; ++j)
        {
            cout << "    Day:" << j << endl;
        }
    }

    getch();
}
```




THANK YOU...