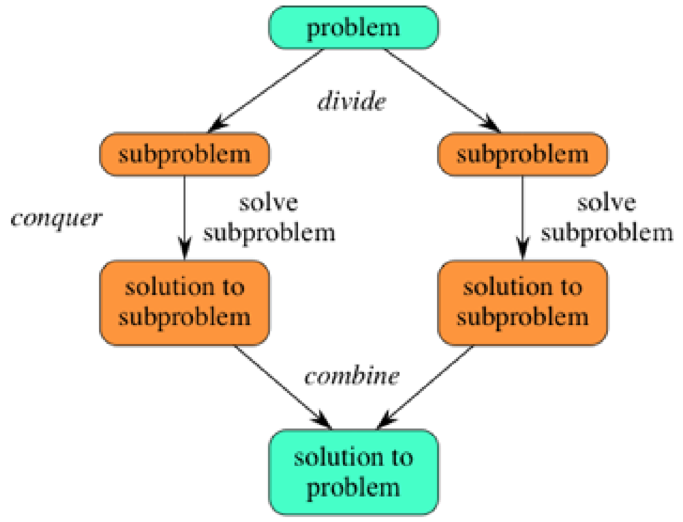
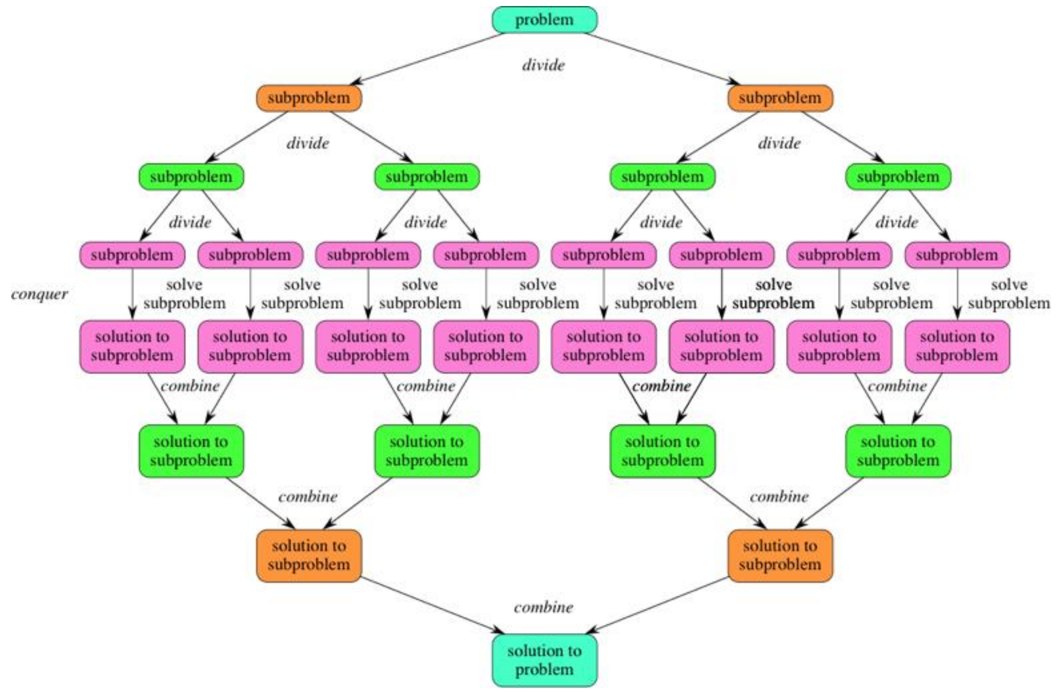
Algorithm

To implement algorithm you need to think in details for solving problem as you can.
Its mean use the **Divide-and-conquer** technique :

1.Divide the problem into a number of subproblems that are smaller instances of the same problem.
2.Conquer the subproblems by solving them recursively. If they are small enough, solve the subproblems as base cases.
3.Combine the solutions to the subproblems into the solution for the original problem.





Sequences

Sequences are the main logical structure of algorithms. When creating algorithms, the instructions are presented in a specific correct order.

A **sequence** can contain any number of instructions but each instruction must be run in the order they are presented. No instruction can be skipped.

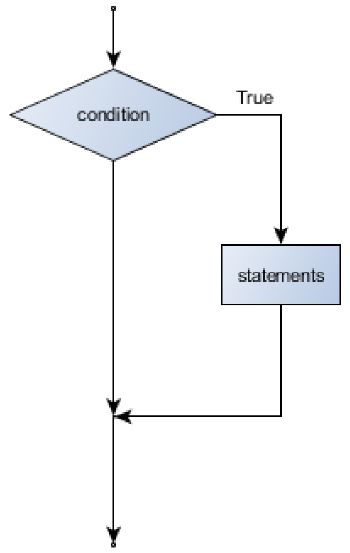
Ex1 : for sequence

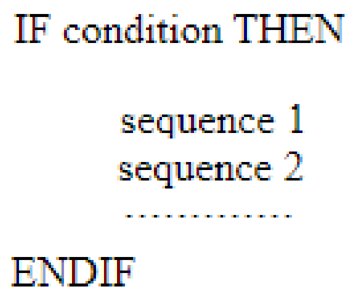
Write an Algorithm to add two numbers entered by the user.

```
Step 1: Start
Step 2: Declare variables num1, num2 and sum.
Step 3: Read values num1 and num2.
Step 4: Add num1 and num2 and assign the result to sum.
Step 5: Display sum
Step 6: Stop
```

Branching (one way branch)

In order to write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly. Conditional statements give us this ability. The simplest form is the **if** statement, which has the general form





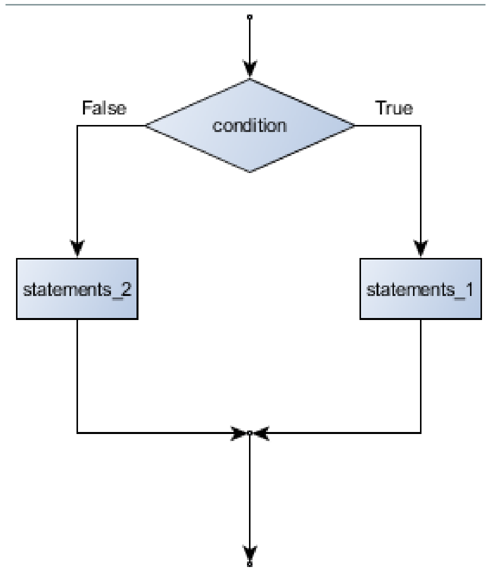
```
IF condition THEN

        sequence 1
        sequence 2
        .............

ENDIF
```

Branching (two way branch)

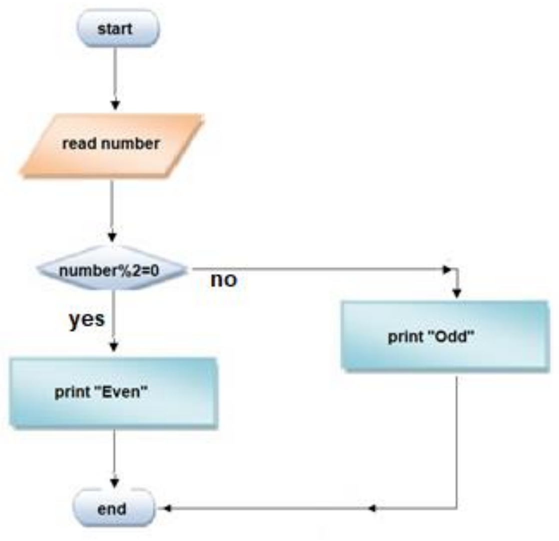
In order to write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly. Conditional statements give us this ability. The simplest form is the **if else** statement, which has the general form



```
IF condition THEN
      sequence 1
      sequence 2
      ..............
ELSE
     sequence 1
     sequence 2
     ..............
ENDIF
```

example

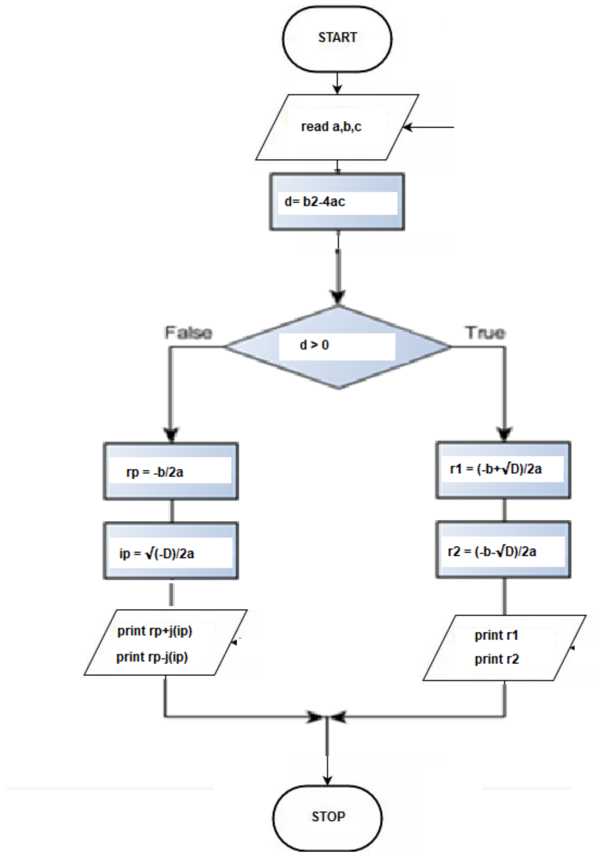
Write an algorithm – flowchart – pseudocode to check whether a given number is even or odd.



```
Read number
If (number%2=0) then
        print ("Even")
Else
        print("Odd")
endif
```

example

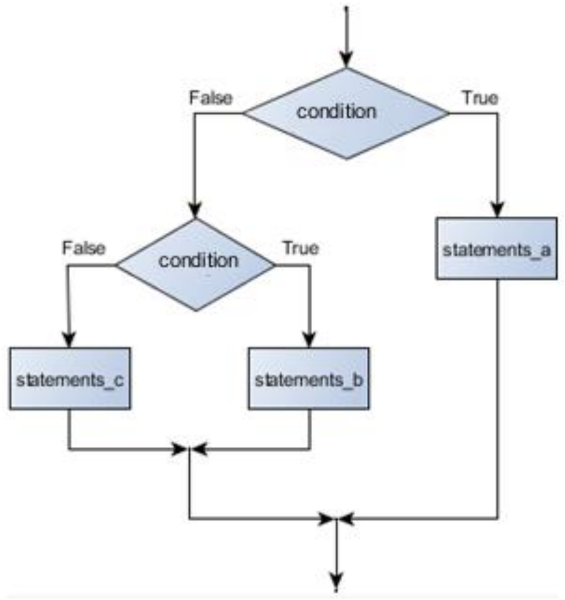
Write an algorithm – flowchart – pseudocode to find Roots of a quadratic equation ax2 + bx + c = 0

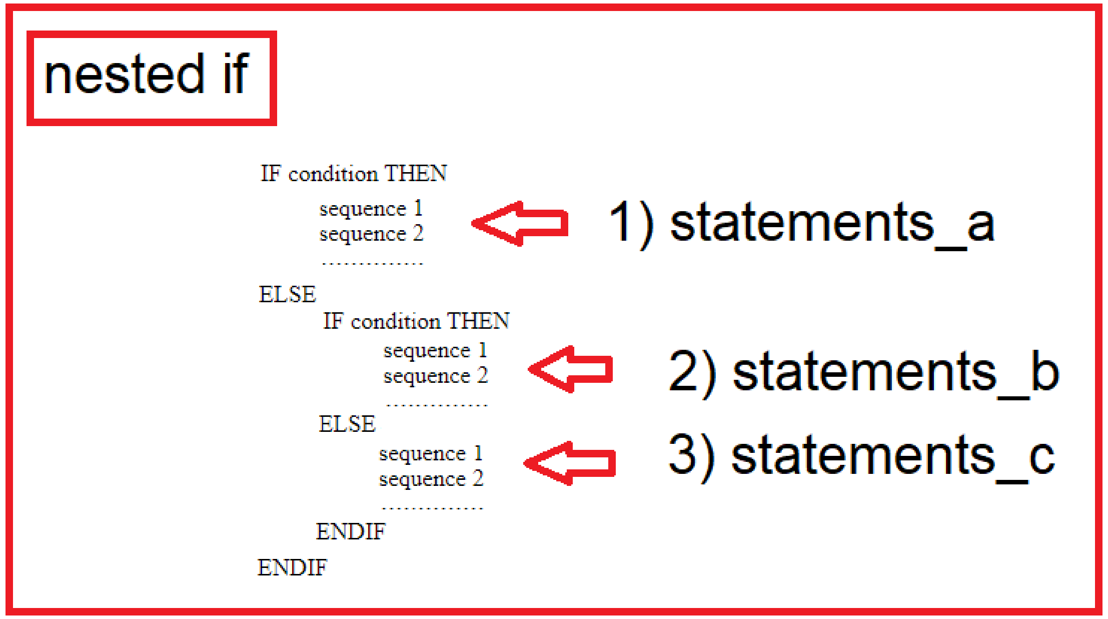


```
Step 1: Start
Step 2: Declare variables a, b, c, D, x1, x2, rp and ip;
Step 2.1 read a,b,c
Step 3: Calculate discriminant
        D ← b2-4ac
Step 4: If D ≥ 0
            r1 ← (-b+√D)/2a
            r2 ← (-b-√D)/2a
            Display r1 and r2 as roots.
       Else
            Calculate real part and imaginary part
            rp ← -b/2a
            ip ← √(-D)/2a
            Display rp+j(ip) and rp-j(ip) as roots
Step 5: Stop
```

Branching (multi way branch)

In order to write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly. Conditional statements give us this ability. The simplest form is the **if else** statement, which has the general form

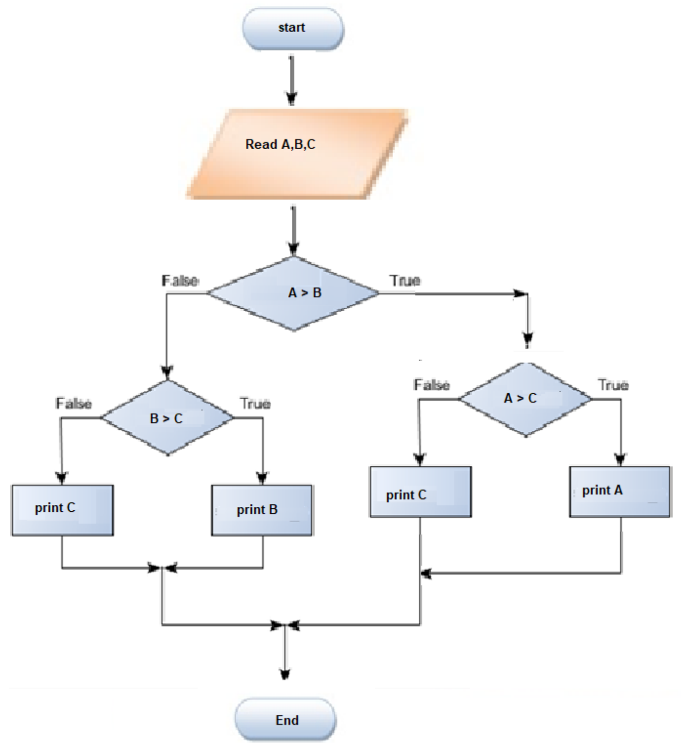



nested if
1) statements_a
2) statements_b
3) statements_c

```
IF condition THEN
    sequence 1
    sequence 2
    ..............
ELSE
    IF condition THEN
        sequence 1
        sequence 2
        ..............
    ELSE
        sequence 1
        sequence 2
        ..............
    ENDIF
ENDIF
```

Example(multi way branch)

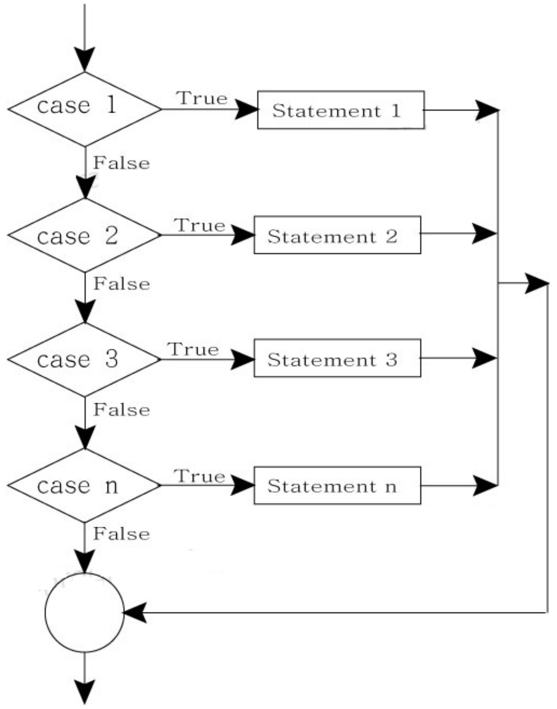
Write an algorithm – flowchart – pseudocode to Find the largest number among three different numbers



```
Step 1: Start
Step 2: Declare variables a,b and c.
Step 3: Read variables a,b and c.
Step 4:
    If a > b
        If a > c
            Display a is the largest number.
        Else
            Display c is the largest number.
        END IF
    Else
        If b > c
            Display b is the largest number.
        Else
            Display c is the greatest number.
        END IF
    END IF
Step 5: Stop
```

Branching (multi way branch)

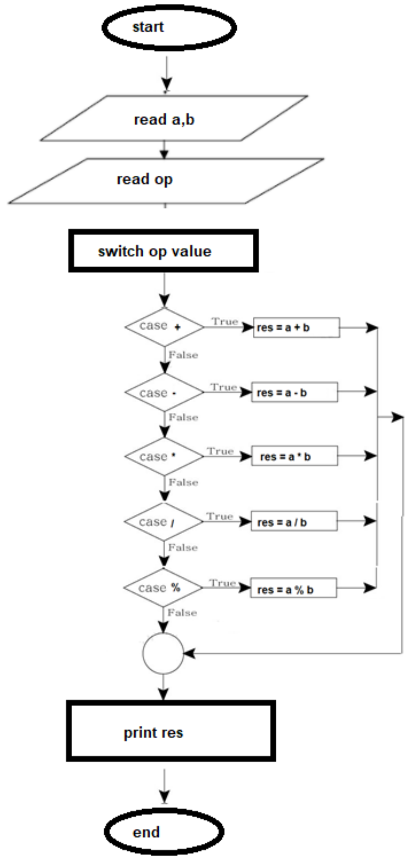
In order to write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly. [Conditional statements](#) give us this ability. The simplest form is the **if else** statement, which has the general form



```
CASE Day OF
   1 : PRINT "Sunday"
   2 : PRINT "Monday"
   3 : PRINT "Tuesday"
   4 : PRINT "Wednesday"
   5 : PRINT "Thursday"
   6 : PRINT "Friday"
   7 : PRINT "Saturday"
   OTHERWISE PRINT "Error"
ENDCASE
```

Example

Write algorithm – flowchart – pseudocode that Input two numeric values and depend on operator as one char that calculate the value of a + b if operate is ‘+’, a - b if operate is ‘-’, and so on for *,%,/ Parameters a and b are type int and op is type char.



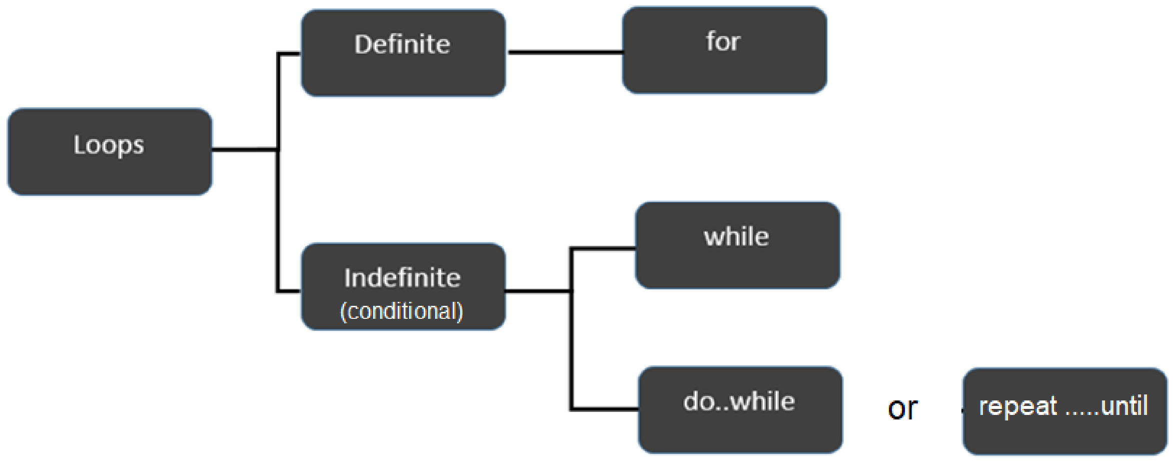
```
Step 1: read a,b
Step 2: read op
Step 3:
Case (op)
    Case ‘+’: res = a + b
    Case ‘-’ : res = a – b
    Case ‘*’ : res = a * b
    Case ‘/’ : res = a / b
    Case ‘%’: res = a % b
End Case
```

looping

Almost all the programming languages provide a concept called **loop**, which helps in executing one or more statements up to a desired number of times.

After executing all the statements given in the loop body, the computer goes back to the given condition and checked again, and the loop is executed again if the condition holds true. This process repeats till the given condition remains true.

To conclude, a loop statement allows us to execute a statement or group of statements multiple times.



Looping (For)

The For Loop is a loop where the program tells the compiler to run a specific code FOR a specified number of times.

This loop allows using four parts, first is the counter initialization, next is the condition to check it , next there is an increment/decrement operation to change the counter variable and finally the body of loop .

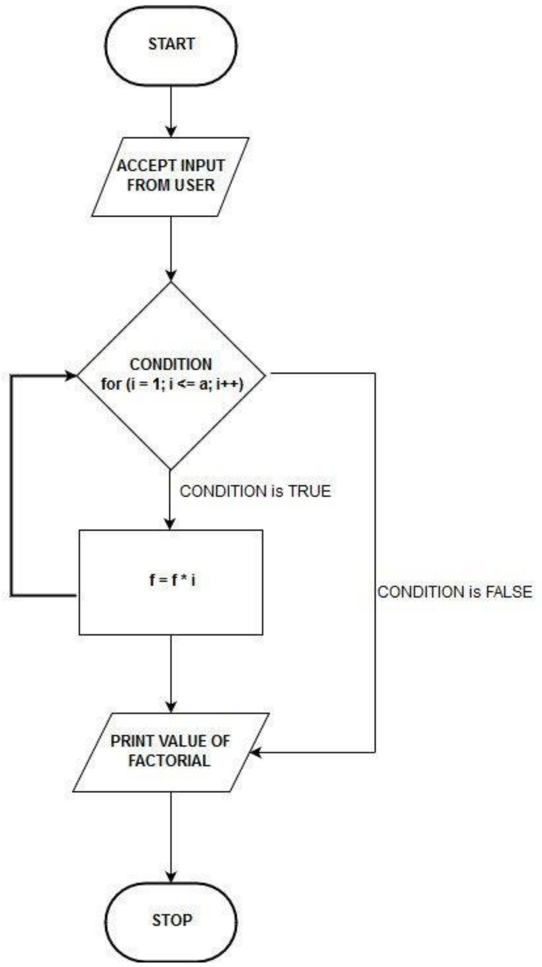
```
for (counter initialization; condition to check ; increment/decrement )
{
  statement(s);
}
```

```
for (i=1; i<=5; i=i+1)
{
  print(i,“hello”)
}
```

```
1,Hello
2,Hello
3,Hello
4,Hello
5,Hello
```

Example

Write algorithm – flowchart – pseudocode to calculate the Factorial of a number entered by the user.



```
Step 1: Start
Step 2: Declare variables n, factorial and i.
Step 3: Read value of n
Step 4: Initialize variables
        factorial ← 1
        i ← 1
Step 5: Repeat the steps until i = n
    5.1: factorial ← factorial*i
    5.2: i ← i+1
Step 6: Display factorial
Step 7: Stop
```

Looping (while)

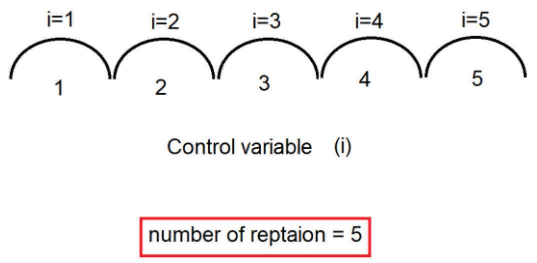
a **while** loop is a control flow **statement** that allows code to be executed repeatedly based on a given Boolean condition.

The while loop evaluates the test expression inside the parenthesis ().

If the test expression is true, statements inside the body of while loop are executed. Then, the test expression is evaluated again.

The process goes on until the test expression is evaluated to false.

If the test expression is false, the loop terminates (ends).



```
while(i<=5)
{
   statement 1;
   statement 2;
   statement 3;
}
```

no initialization

```
i=1;
while(i<=5)
{
   statement 1;
   statement 2;
   statement 3;
}
```

no update

```
i=1;
while(i<=5)
{
   i=i+1;
   statement 1;
   statement 2;
   statement 3;
}
```

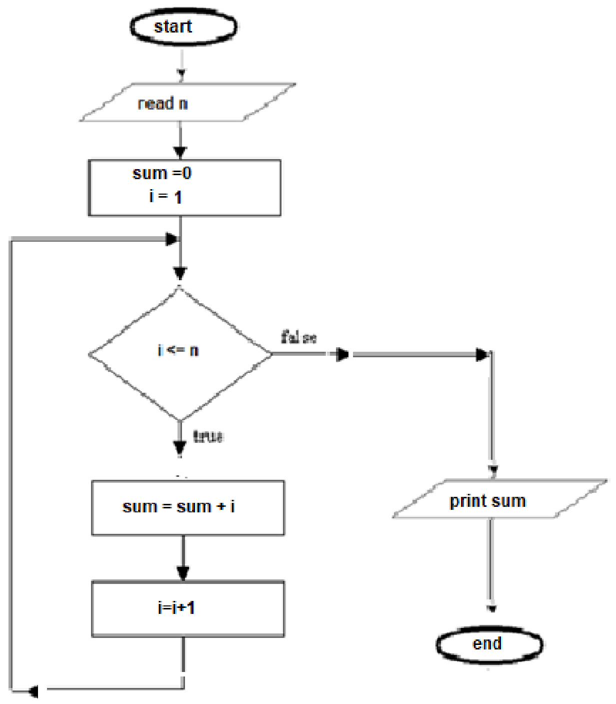
wrong update place

```
i=1;
while(i<=5)
{
   statement 1;
   statement 2;
   statement 3;
   i=i+1;
}
```

correct

example

Write algorithm – flowchart – pseudocode to calculate sum of digits in all numbers from 1 to n Given a number n.

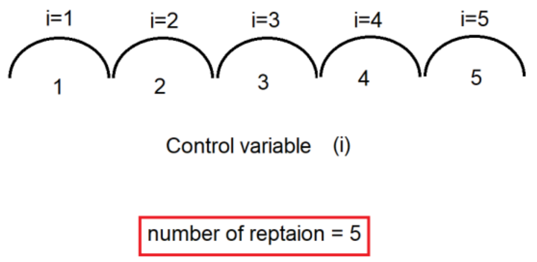


```
Read n
Sum = 0
i =1
While ( i<=n)
        sum = sum+i
        i=i+1
End while
Print sum
```

Looping (do …while)

Now that we are working with loops, let's explore "loop control variables".

The do….while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.



```
do {
  statement 1;
  statement 2;
}
while (i <= 5);
```

✗

```
i=1;
do {
   statement 1;
   statement 2;
}
while (i <= 5);
```

✗

```
i=1;
do {
   i=i+1;
   statement 1;
   statement 2;
}
while (i <= 5);
```

✗

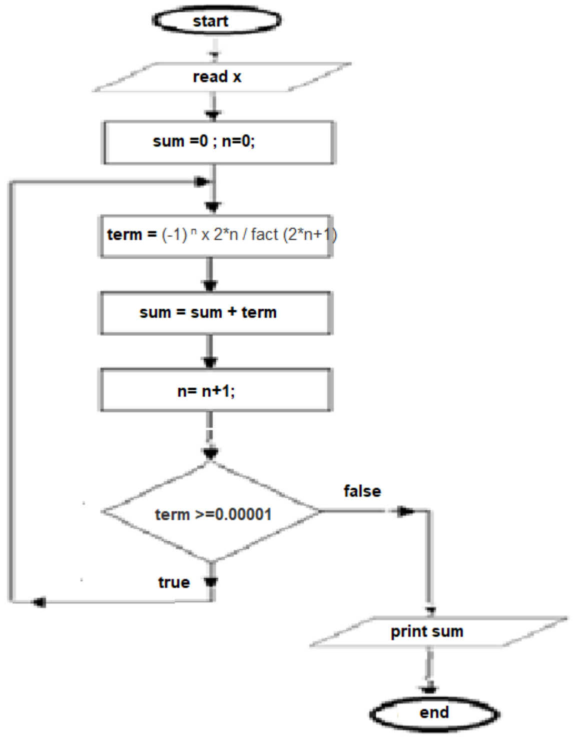
```
i=1;
do {
   statement 1;
   statement 2;
   i=i+1;
}
while (i <= 5);
```

✓

example

Write algorithm – flowchart – pseudocode to calculate sin(x) using the formula until the term less that 0.00001.

$$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

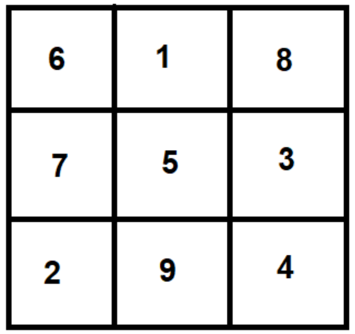


```
Read x
Sum = 0
 n=0
do
            term = $(-1)^n x^{(2*n+1)}$ / fact (2*n+1)
            Sum = Sum +term;
            n++;
While ( term >=0.00001)
Print "sin of ",x,"=",sum
```

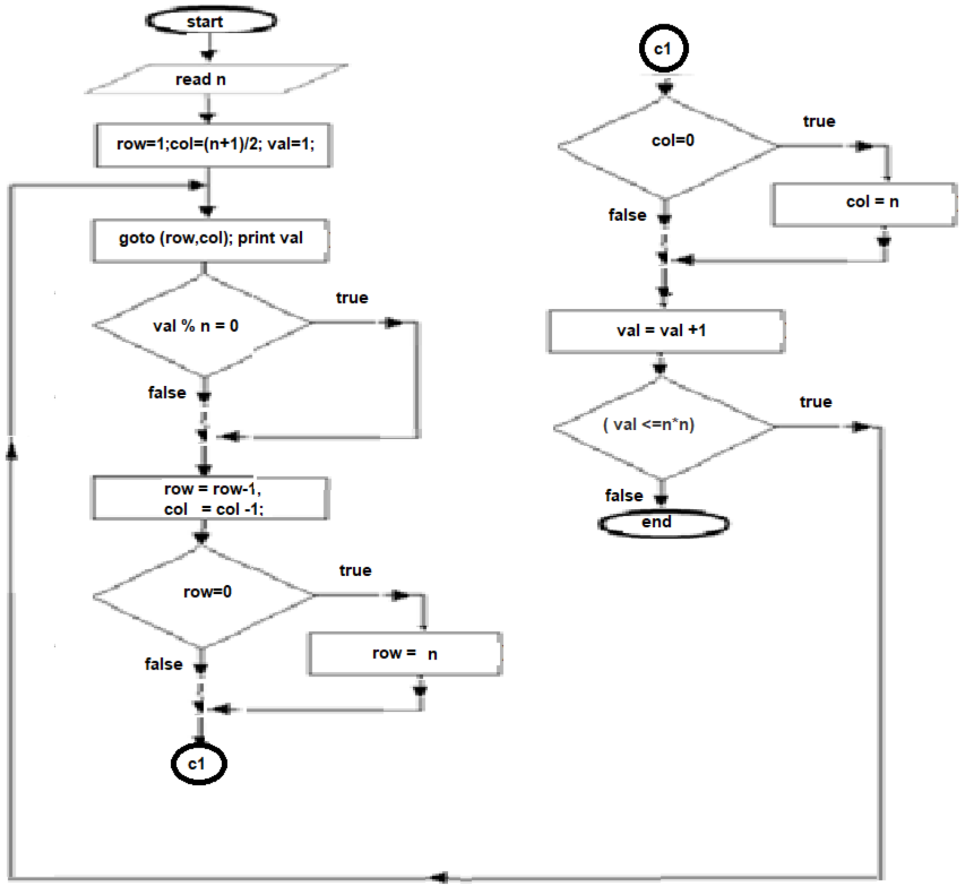
Hint : use the concept of function to build the function Fact(int) the to calculate the factorial needed in each term .

General Example

Write algorithm – flowchart – pseudocode to A magic square is a square array of numbers consisting of the distinct positive integers 1, 2, ..., n^2 arranged such that the sum of the n numbers in any horizontal, vertical, or main diagonal line is always the same number



6	1	8
7	5	3
2	9	4



```
Read n
Row=1;col = (n+1)/2 ; val =1;
do
        goto (row,col)
        print val
        if ( val % n ==0 ) then
                row=row+1
        else
                row=row-1;
                col = col -1
        endif
        if (row=0) then
                row=n
        endif
        if (col=0) then
                col=n;
        endif
        val = val +1;
While ( val <=n*n)
```