SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)
COIMBATORE-35

DEPARTMENT OF INFORMATION TECHNOLOGY

23CST101 Problem Solving and C Programming

UNIT I INTRODUCTION TO PROBLEM SOLVING TECHNIQUES

Fundamentals - Computer Hardware – Computer Software - Algorithms - Building blocks of algorithms (statements, state, control flow, functions) - Notation (pseudo code, flow chart, and programming language) -Problem formulation - Algorithmic problem solving - Simple strategies for developing algorithms (iteration, recursion). Illustrative problems.

PROBLEM SOLVING

Problem solving is the systematic approach to define the problem and creating number of solutions.

The problem-solving process starts with the problem specifications and ends with a Correct program.

PROBLEM SOLVING TECHNIQUES

Problem solving technique is a set of techniques that helps in providing logic for solving a problem.

Problem Solving Techniques

Problem solving can be expressed in the form of

1. Algorithms.
2. Flowcharts.
3. Pseudo codes.
4. programs

ALGORITHM

It is defined as a sequence of instructions that describe a method for solving a problem. In other words it is a step by step procedure for solving a problem.

Properties of Algorithms

- Should be written in simple English
- Each and every instruction should be precise and unambiguous.
- Instructions in an algorithm should not be repeated infinitely.

- Algorithm should conclude after a finite number of steps.
- Should have an end point
- Derived results should be obtained only after the algorithm terminates.

Qualities of a good algorithm

The following are the primary factors that are often used to judge the quality of the algorithms.

- Time – To execute a program, the computer system takes some amount of time. The lesser is the time required, the better is the algorithm.
- Memory – To execute a program, computer system takes some amount of memory space. The lesser is the memory required, the better is the algorithm.
- Accuracy – Multiple algorithms may provide suitable or correct solutions to a given problem, some of these may provide more accurate results than others, and such algorithms may be suitable.

Example

Write an algorithm to print „Good Morning"

Step 1: Start

Step 2: Print "Good Morning"

Step 3: Stop

BUILDING BLOCKS OF ALGORITHMS (statements, state, control flow, functions)

Algorithms can be constructed from basic building blocks namely, sequence, selection and iteration.

Statements: - Statement is a single action in a computer.

In a computer statements might include some of the following actions

Ø input data-information given to the program

Ø process data-perform operation on a given input

Ø output data-processed result

State - Transition from one process to another process under specified condition with in a time is called state.

Control flow:

The process of executing the individual statements in a given order is called control flow.

The control can be executed in three ways

1. sequence
2. selection
3. iteration

Sequence:

All the instructions are executed one after another is called sequence execution.

Example:

Add two numbers:

Step 1: Start

Step 2: get a,b

Step 3: calculate c=a+b

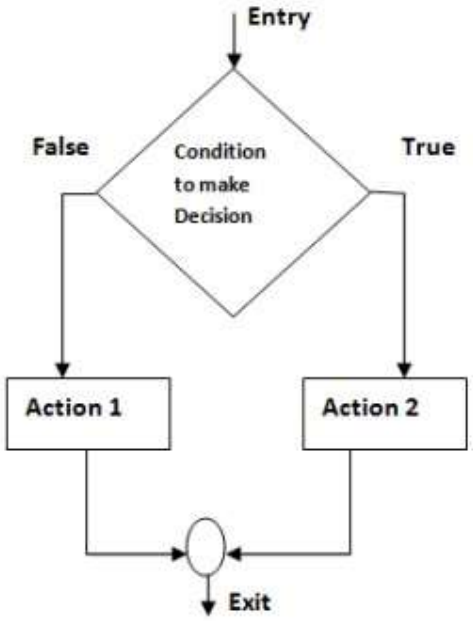
Step 4: Display c

Step 5: Stop

Selection:

A selection statement causes the program control to be transferred to a specific part of the program based upon the condition.

If the conditional test is true, one part of the program will be executed, otherwise it will execute the other part of the program.



Example

Write an algorithm to check whether he is eligible to vote?

Step 1: Start

Step 2: Get age

Step 3: if age >= 18 print "Eligible to vote"

Step 4: else print "Not eligible to vote"

Step 6: Stop

Iteration:

In some programs, certain set of statements are executed again and again based upon conditional test. i.e. executed more than one time. This type of execution is called looping or iteration.

Example

Write an algorithm to print all natural numbers up to n

Step 1: Start

Step 2: get n value.

Step 3: initialize i=1

Step 4: if (i<=n) go to step 5 else go to step 7

Step 5: Print i value and increment i value by 1

Step 6: go to step 4

Step 7: Stop

Functions:

- Function is a sub program which consists of block of code(set of instructions) that performs a particular task.
- For complex problems, the problem is been divided into smaller and simpler tasks during algorithm design.

Benefits of Using Functions

- v Reduction in line of code
- v code reuse
- v Better readability
- v Information hiding
- v Easy to debug and test
- v Improved maintainability

Example:

Algorithm for addition of two numbers using function

Main function()

Step 1: Start

Step 2: Call the function add()

Step 3: Stop

sub function add()

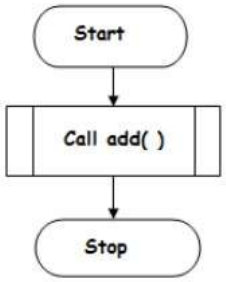
Step 1: Function start

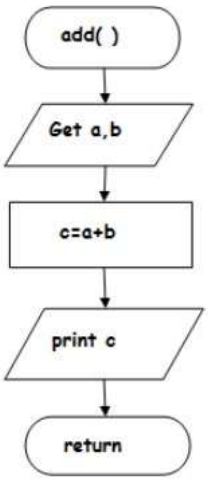
Step 2: Get a, b Values

Step 3: add c=a+b

Step 4: Print c

Step 5: Return





NOTATIONS

FLOW CHART

Flow chart is defined as graphical representation of the logic for problem solving.

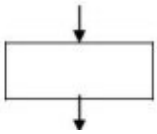
The purpose of flowchart is making the logic of the program clear in a visual representation.

Symbol	Symbol Name	Description
	Flow Lines	Used to connect symbols
	Terminal	Used to start, pause or halt in the program logic
	Input/output	Represents the information entering or leaving the system
	Processing	Represents arithmetic and logical instructions
	Decision	Represents a decision to be made
	Connector	Used to Join different flow lines
	Sub function	used to call function

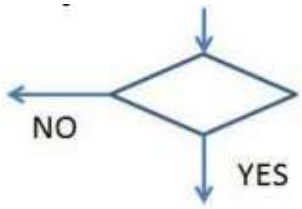
Rules for drawing a flowchart

1. The flowchart should be clear, neat and easy to follow.
2. The flowchart must have a logical start and finish.

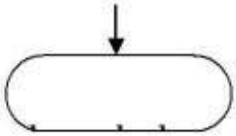
3. Only one flow line should come out from a process symbol.



4. Only one flow line should enter a decision symbol. However, two or three flow lines may leave the decision symbol.



5. Only one flow line is used with a terminal symbol.



6. Within standard symbols, write briefly and precisely.

7. Intersection of flow lines should be avoided.

Advantages of flowchart:

- **Communication: -** Flowcharts are better way of communicating the logic of a system to all concerned.
- **Effective analysis: -** With the help of flowchart, problem can be analyzed in more effective way.
- **Proper documentation: -** Program flowcharts serve as a good program documentation, which is needed for various purposes.
- **Efficient Coding: -** The flowcharts act as a guide or blueprint during the systems analysis and program development phase.
- **Proper Debugging: -** The flowchart helps in debugging process.
- **Efficient Program Maintenance: -** The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part.

Disadvantages of flow chart:

- **Complex logic: -** Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.
- **Alterations and Modifications: -** If alterations are required the flowchart may require re-drawing completely.
- **Reproduction: -** As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.
- **Cost: F**or large application the time and cost of flowchart drawing becomes costly.

PSEUDO CODE:

- Pseudo code consists of short, readable and formally styled English languages used for explain an algorithm.
- It does not include details like variable declaration, subroutines.
- It is easier to understand for the programmer or non programmer to understand the general working of the program, because it is not based on any programming language.
- It gives us the sketch of the program before actual coding.
- It is not a machine readable
- Pseudo code can't be compiled and executed.
- There is no standard syntax for pseudo code.

Guidelines for writing pseudo code:

- Write one statement per line
- Capitalize initial keyword
- Indent to hierarchy
- End multiline structure
- Keep statements language independent

Common keywords used in pseudocode

The following gives common keywords used in pseudocodes.

1. **//:** This keyword used to represent a comment.
2. **BEGIN,END:** Begin is the first statement and end is the last statement.
3. **INPUT, GET, READ**: The keyword is used to inputting data.
4. **COMPUTE, CALCULATE**: used for calculation of the result of the given expression.
5. **ADD, SUBTRACT, INITIALIZE** used for addition, subtraction and initialization**.**
6. **OUTPUT, PRINT, DISPLAY**: It is used to display the output of the program.
7. **IF, ELSE, ENDIF**: used to make decision.

8. **WHILE, ENDWHILE**: used for iterative statements.
9. **FOR, ENDFOR**: Another iterative incremented/decremented tested automatically.

Syntax for if else:

```
IF (condition)THEN
statement
...
ELSE
statement
...
ENDIF
```

Example: Greatest of two numbers

```
BEGIN
READ a,b
IF (a>b) THEN
DISPLAY a is greater
ELSE
DISPLAY b is greater
END IF
END
```

Syntax for For:

```
FOR( start-value to end-value) DO
statement
...
ENDFOR
```

Example: Print n natural numbers

```
BEGIN
GET n
INITIALIZE i=1
FOR (i<=n) DO
PRINT i
i=i+1
ENDFOR
END
```

Syntax for While:

WHILE (condition) DO

statement

...

ENDWHILE

Example: Print n natural numbers

BEGIN

GET n

INITIALIZE i=1

WHILE(i<=n) DO

PRINT i

i=i+1

ENDWHILE

END

Advantages:

- Pseudo is independent of any language; it can be used by most programmers.
- It is easy to translate pseudo code into a programming language.
- It can be easily modified as compared to flowchart.
- Converting a pseudo code to programming language is very easy as compared with converting a flowchart to programming language.

Disadvantages:

- It does not provide visual representation of the program's logic.
- There are no accepted standards for writing pseudo codes.
- It cannot be compiled nor executed.
- For a beginner, It is more difficult to follow the logic or write pseudo code as compared to flowchart.

Example:

Addition of two numbers:

BEGIN

GET a,b

ADD c=a+b

PRINT c

END

Algorithm	Flowchart	Pseudo code
An algorithm is a sequence of instructions used to solve a problem	It is a graphical representation of algorithm	It is a language representation of algorithm.
User needs knowledge to write algorithm.	not need knowledge of program to draw or understand flowchart	Not need knowledge of program language to understand or write a pseudo code.

PROGRAMMING LANGUAGE

A programming language is a set of symbols and rules for instructing a computer to perform specific tasks. The programmers have to follow all the specified rules before writing program using programming language. The user has to communicate with the computer using language which it can understand.

Types of programming language

- Machine language
- Assembly language
- High level language

Machine language:

The computer can understand only machine language which uses 0's and 1's. In machine language the different instructions are formed by taking different combinations of 0's and 1's.

Advantages:

Translation free:

Machine language is the only language which the computer understands. For executing any program written in any programming language, the conversion to machine language is necessary. The program written in machine language can be executed directly on computer. In this case any conversion process is not required.

High speed

The machine language program is translation free. Since the conversion time is saved, the execution of machine language program is extremely fast.

Disadvantage:

Ø It is hard to find errors in a program written in the machine language.

Ø Writhing program in machine language is a time consuming process.

Machine dependent: According to architecture used, the computer differs from each other. So machine language differs from computer to computer. So a program developed for a particular type of computer may not run on other type of computer.

Assembly language:

To overcome the issues in programming language and make the programming process easier, an assembly language is developed which is logically equivalent to machine language but it is easier for people to read, write and understand.

Assembly language is symbolic representation of machine language. Assembly languages are symbolic programming language that uses symbolic notation to represent machine language instructions. They are called low level language because they are so closely related to the machines.

Assembler

Assembler is the program which translates assembly language instruction in to a machine language.

- Easy to understand and use.
- It is easy to locate and correct errors.

Disadvantage

Machine dependent

The assembly language program which can be executed on the machine depends on the architecture of that computer.

Hard to learn

It is machine dependent, so the programmer should have the hardware knowledge to create applications using assembly language.

Less efficient

Ø Execution time of assembly language program is more than machine language program.

Ø Because assembler is needed to convert from assembly language to machine language.

High level language

High level language contains English words and symbols. The specified rules are to be followed while writing program in high level language. The interpreter or compilers are used for converting these programs in to machine readable form.

Translating high level language to machine language

The programs that translate high level language in to machine language are called interpreter or compiler.

Compiler:

A compiler is a program which translates the source code written in a high level language in to object code which is in machine language program. Compiler reads the whole program written in high level language and translates it to machine language. If any error is found it display error message on the screen.

Interpreter

Interpreter translates the high level language program in line by line manner. The interpreter translates a high level language statement in a source program to a machine code and executes it immediately before translating the next statement. When an error is found the execution of the program is halted and error message is displayed on the screen.

Advantages

Readability

High level language is closer to natural language so they are easier to learn and understand

Machine independent

High level language program have the advantage of being portable between machines.

Easy debugging

Easy to find and correct error in high level language

Disadvantages

Less efficient

The translation process increases the execution time of the program. Programs in high level language require more memory and take more execution time to execute.

They are divided into following categories:

1. Interpreted programming languages
2. Functional programming languages
3. Compiled programming languages
4. Procedural programming languages
5. Scripting programming language
6. Markup programming language
7. Concurrent programming language
8. Object oriented programming language

Interpreted programming languages:

An interpreted language is a programming language for which most of its implementation executes instructions directly, without previously compiling a program into machine language instructions. The interpreter executes the program directly translating each statement into a sequence of one or more subroutines already compiled into machine code.

Examples:

Pascal

Python

Functional programming language:

Functional programming language defines every computation as a mathematical evaluation. They focus on the programming languages are bound to mathematical calculations

Examples:

Clean

Haskell

Compiled Programming language:

A compiled programming is a programming language whose implementation are typically compilers and not interpreters.

It will produce a machine code from source code.

Examples:

C

C++

C#

JAVA

Procedural programming language:

Procedural (imperative) programming implies specifying the steps that the programs should take to reach to an intended state.

A procedure is a group of statements that can be referred through a procedure call. Procedures help in the reuse of code. Procedural programming makes the programs structured and easily traceable for program flow.

Examples:

Hyper talk

MATLAB

Scripting language:

Scripting language are programming languages that control an application. Scripts can execute independent of any other application. They are mostly embedded in the application that they control and are used to automate frequently executed tasks like communicating with external program.

Examples:

Apple script

VB script

Markup languages:

A markup language is an artificial language that uses annotations to text that define hoe the text is to be displayed.

Examples:

HTML

XML

Concurrent programming language:

Concurrent programming is a computer programming technique that provides for the execution of operation concurrently, either with in a single computer or across a number of systems.

Examples:

Joule

Limbo

Object oriented programming language:

Object oriented programming is a programming paradigm based on the concept of objects which may contain data in the form of procedures often known as methods.

Examples:

Lava

Moto

ALGORITHMIC PROBLEM SOLVING:

Algorithmic problem solving is solving problem that require the formulation of an algorithm for the solution.

Understanding the Problem

- It is the process of finding the input of the problem that the algorithm solves.
- It is very important to specify exactly the set of inputs the algorithm needs to handle.

- A correct algorithm is not one that works most of the time, but one that works correctly for *all* legitimate inputs.

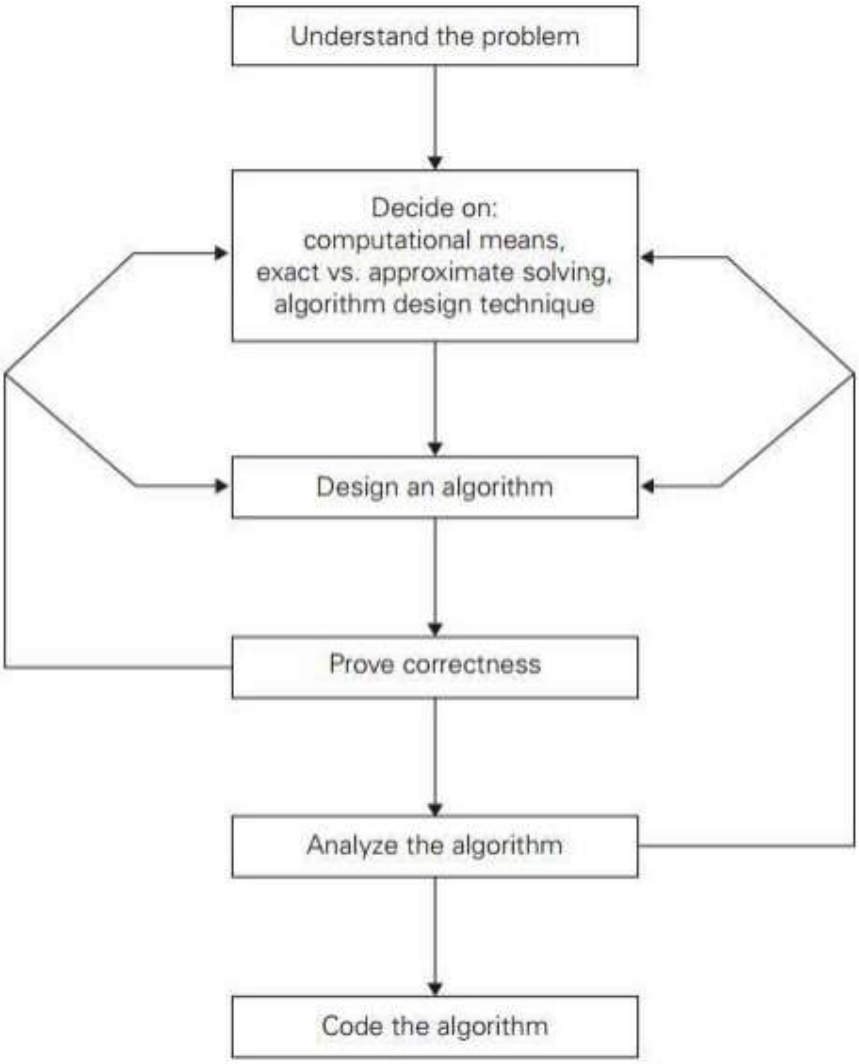


FIGURE 1.2 Algorithm design and analysis process.

Ascertaining the Capabilities of the Computational Device

- If the instructions are executed one after another, it is called sequential algorithm.
- If the instructions are executed concurrently, it is called parallel algorithm.

Choosing between Exact and Approximate Problem Solving

- The next principal decision is to choose between solving the problem exactly or solving it approximately.

- Based on this, the algorithms are classified as exact algorithm and approximation algorithm.

Deciding a data structure:

- Data structure plays a vital role in designing and analysis the algorithms.
- Some of the algorithm design techniques also depend on the structuring data specifying a problem's instance
- Algorithm+ Data structure=programs.

Algorithm Design Techniques

- An ***algorithm design technique*** (or "strategy" or "paradigm") is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.
- Learning these techniques is of utmost importance for the following reasons.
- First, they provide guidance for designing algorithms for new problems,
- Second, algorithms are the cornerstone of computer science

Methods of Specifying an Algorithm

- ***Pseudocode*** is a mixture of a natural language and programming language-like constructs. Pseudocode is usually more precise than natural language, and its usage often yields more succinct algorithm descriptions.
- In the earlier days of computing, the dominant vehicle for specifying algorithms was a ***flowchart***, a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps.
- **Programming language** can be fed into an electronic computer directly. Instead, it needs to be converted into a computer program written in a particular computer language. We can look at such a program as yet another way of specifying the algorithm, although it is preferable to consider it as the algorithm's implementation.

Proving an Algorithm's Correctness

- Once an algorithm has been specified, you have to prove its ***correctness***. That is, you have to prove that the algorithm yields a required result for every legitimate input in a finite amount of time.
- A common technique for proving correctness is to use mathematical induction because an algorithm's iterations provide a natural sequence of steps needed for such proofs.
- It might be worth mentioning that although tracing the algorithm's performance for a few specific inputs can be a very worthwhile activity, it cannot prove the algorithm's

correctness conclusively. But in order to show that an algorithm is incorrect, you need just one instance of its input for which the algorithm fails.

Analysing an Algorithm

1. Efficiency.

- *Time efficiency,* indicating how fast the algorithm runs,
- *Space efficiency*, indicating how much extra memory it uses.

2. simplicity.

- An algorithm should be precisely defined and investigated with mathematical expressions.
- Simpler algorithms are easier to understand and easier to program.
- Simple algorithms usually contain fewer bugs.

Coding an Algorithm

- Most algorithms are destined to be ultimately implemented as computer programs. Programming an algorithm presents both a peril and an opportunity.
- A working program provides an additional opportunity in allowing an empirical analysis of the underlying algorithm. Such an analysis is based on timing the program on several inputs and then analyzing the results obtained.

SIMPLE STRATEGIES FOR DEVELOPING ALGORITHMS:

- iterations
- Recursions

1. Iterations:

A sequence of statements is executed until a specified condition is true is called iterations.

- for loop
- While loop

Syntax for For:

FOR(*start-value to end-value*) DO

Statement

...

ENDFOR

Example: Print n natural numbers

BEGIN

GET n

INITIALIZE i=1

FOR (i<=n) DO

PRINT i

i=i+1

ENDFOR

END

Syntax for While:

WHILE (condition) DO

Statement

...

ENDWHILE

Example: Print n natural numbers

BEGIN

GET n

INITIALIZE i=1

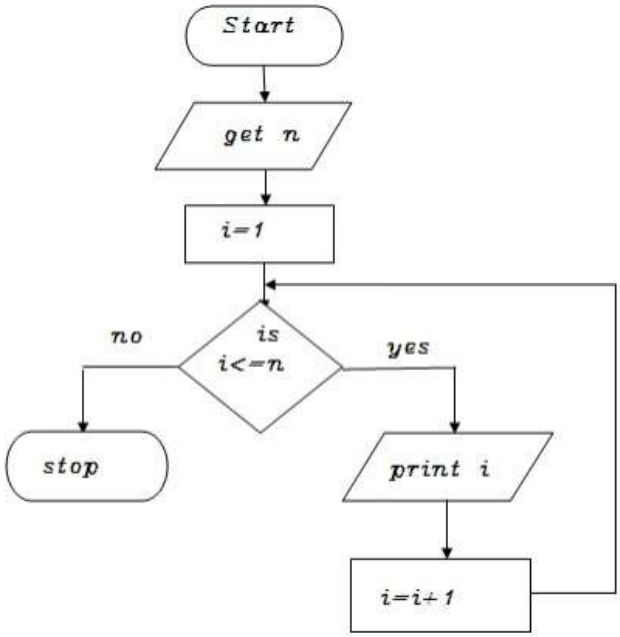
WHILE(i<=n) DO

PRINT i

i=i+1

ENDWHILE

END



Recursions:

- A function that calls itself is known as recursion.
- Recursion is a process by which a function calls itself repeatedly until some specified condition has been satisfied.

Algorithm for factorial of n numbers using recursion:

Main function:

Step1: Start

Step2: Get n

Step3: call factorial(n)

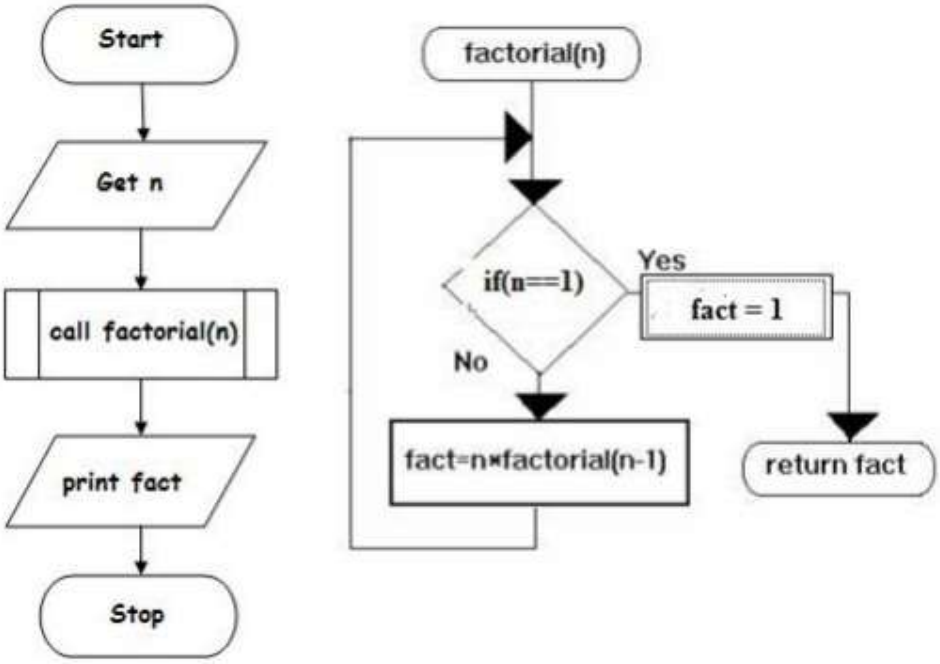
Step4: print fact

Step5: Stop

Sub function factorial(n):

Step1: if(n==1) then fact=1 return fact

Step2: else fact=n*factorial(n-1) and return fact



Pseudo code for factorial using recursion:

Main function:

BEGIN

GET n

CALL factorial(n)

PRINT fact

BIN

Sub function factorial(n):

IF(n==1) THEN

    fact=1

    RETURN fact

ELSE

    RETURN fact=n*factorial(n-1)

More examples:

Write an algorithm to find area of a rectangle

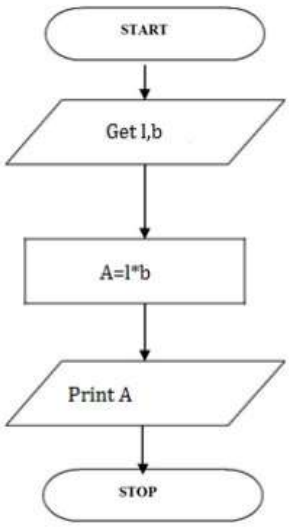
Step 1: Start

Step 2: get l,b values

Step 3: Calculate A=l*b

Step 4: Display A

Step 5: Stop



BEGIN

READ l,b

CALCULATE A=l*b

DISPLAY A

END

Write an algorithm for Calculating area and circumference of circle

Step 1: Start

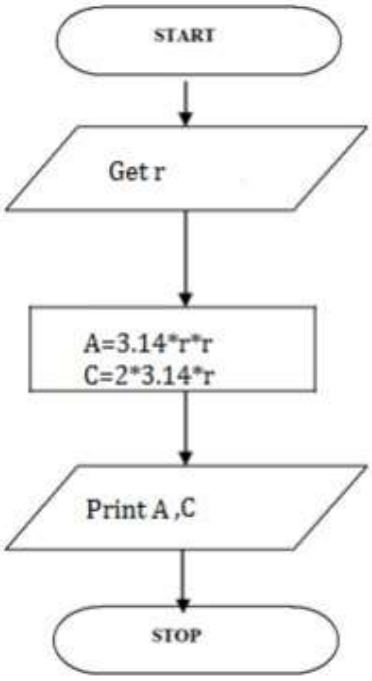
Step 2: get r value

Step 3: Calculate A=3.14*r*r

Step 4: Calculate C=2.3.14*r

Step 5: Display A,C

Step 6: Stop



BEGIN

READ r

CALCULATE A and C

A=3.14*r*r

C=2*3.14*r

DISPLAY A

END

Write an algorithm for Calculating simple interest

Step 1: Start

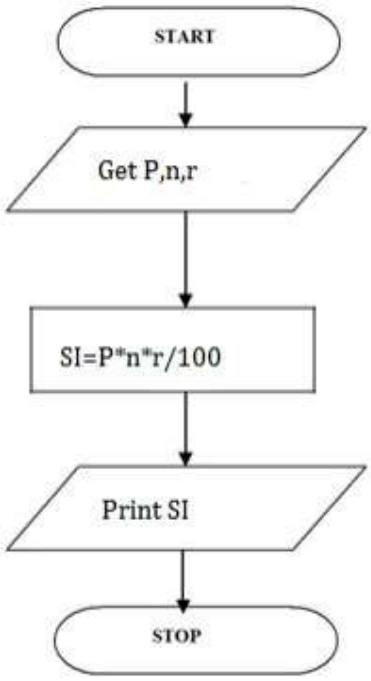
Step 2: get P, n, r value

Step3:Calculate

SI=(p*n*r)/100

Step 4: Display S

Step 5: Stop



BEGIN

READ P, n, r

CALCULATE S

SI=(p*n*r)/100

DISPLAY SI

END

Write an algorithm for Calculating engineering cutoff

Step 1: Start

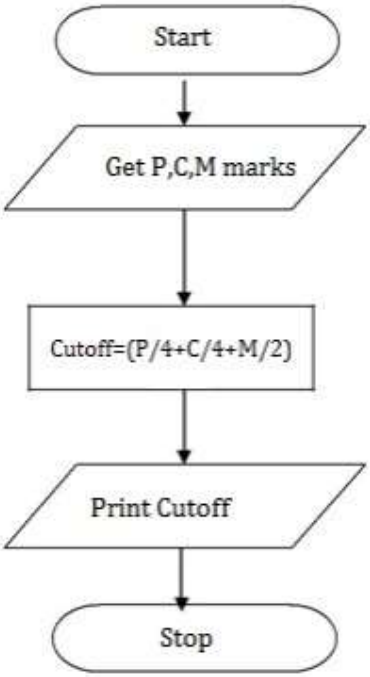
Step2: get P,C,M value

Step3:calculate

Cutoff= (P/4+C/4+M/2)

Step 4: Display Cutoff

Step 5: Stop



BEGIN

READ P,C,M

CALCULATE

Cutoff= (P/4+C/4+M/2)

DISPLAY Cutoff

END

To check greatest of two numbers

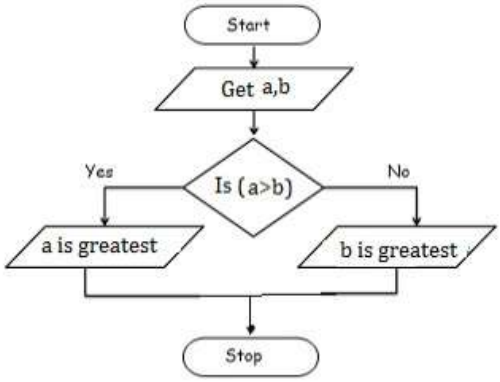
Step 1: Start

Step 2: get a,b value

Step 3: check if(a>b) print a is greater

Step 4: else b is greater

Step 5: Stop



BEGIN

READ a,b

IF (a>b) THEN

DISPLAY a is greater

ELSE

DISPLAY b is greater

END IF

END

To check leap year or not

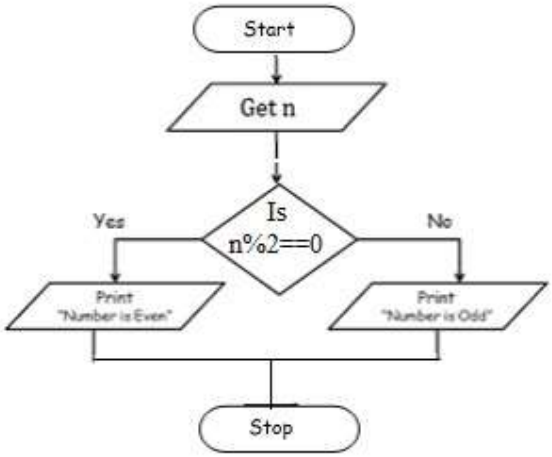
Step 1: Start

Step 2: get y

Step 3: if(y%4==0) print leap year

Step 4: else print not leap year

Step 5: Stop



BEGIN

READ y

IF (y%4==0) THEN

DISPLAY leap year

ELSE

DISPLAY not leap year

END IF

END

To check positive or negative number

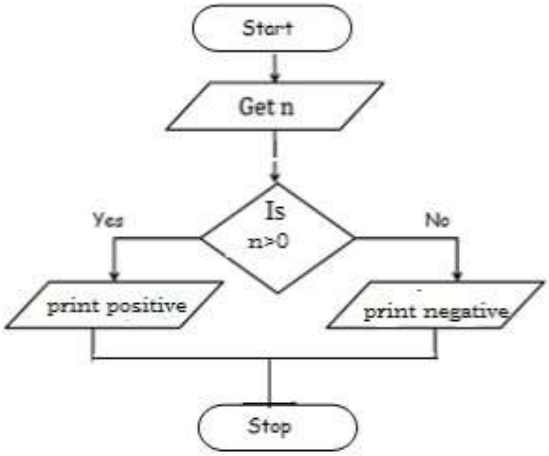
Step 1: Start

Step 2: get num

Step 3: check if(num>0) print a is positive

Step 4: else num is negative

Step 5: Stop



BEGIN

READ num

IF (num>0) THEN

DISPLAY num is positive

ELSE

DISPLAY num is negative

END IF

END

To check odd or even number

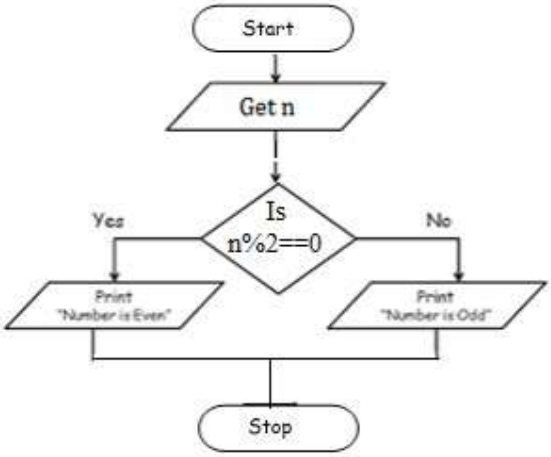
Step 1: Start

Step 2: get num

Step 3: check if(num%2==0) print num is even

Step 4: else num is odd

Step 5: Stop



BEGIN

READ num

IF (num%2==0) THEN

DISPLAY num is even

ELSE

DISPLAY num is odd

END IF

END

To check greatest of three numbers

Step1: Start

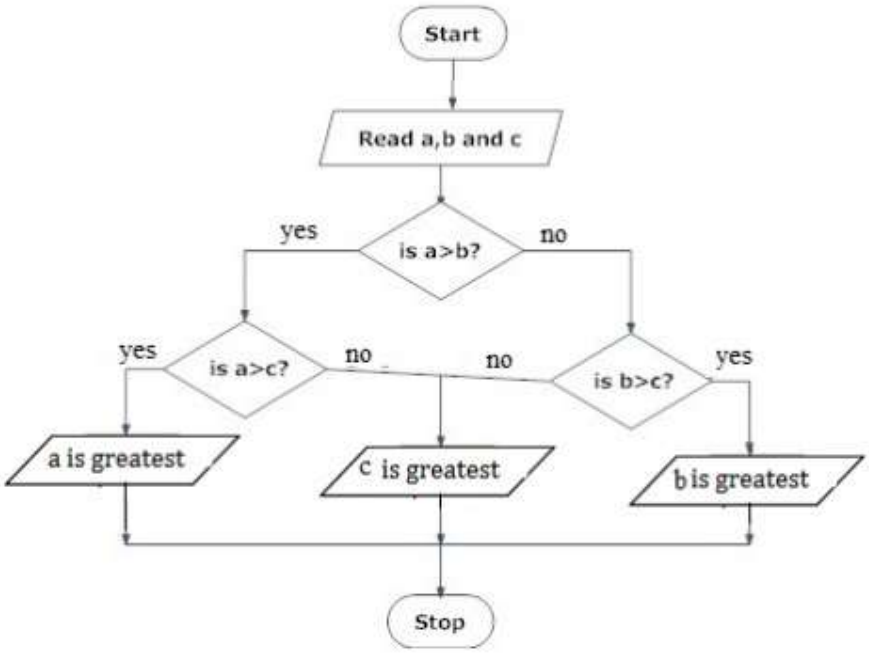
Step2: Get A, B, C

Step3: if(A>B) goto Step4 else goto step5

Step4: If(A>C) print A else print C

Step5: If(B>C) print B else print C

Step6: Stop



BEGIN

READ a, b, c

IF (a>b) THEN

IF(a>c) THEN

DISPLAY a is greater

ELSE

DISPLAY c is greater

END IF

ELSE

IF(b>c) THEN

DISPLAY b is greater

ELSE

DISPLAY c is greater

END IF

END IF

END

Write an algorithm to check whether given number is +ve, -ve or zero.

Step 1: Start

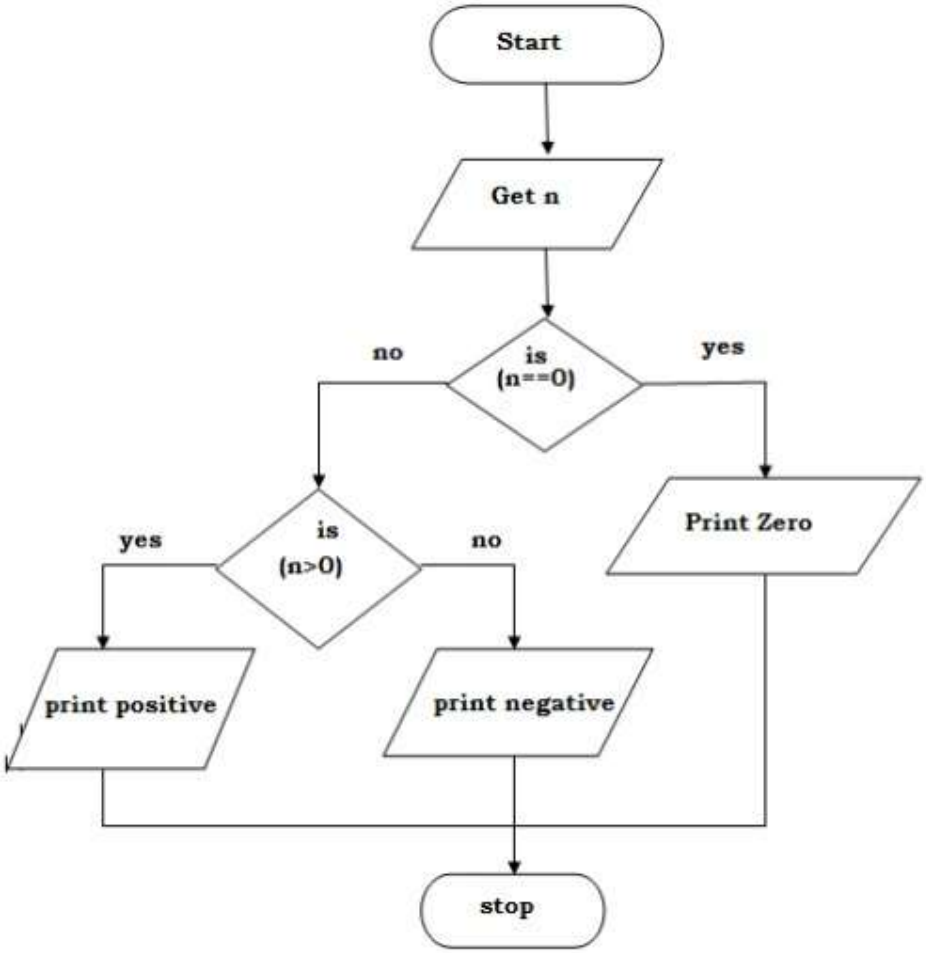
Step 2: Get n value.

Step 3: if (n ==0) print “Given number is Zero” Else goto step4

Step 4: if (n > 0) then Print “Given number is +ve”

Step 5: else Print “Given number is -ve”

Step 6: Stop



BEGIN

GET n

IF(n==0) THEN

DISPLAY " n is zero"

ELSE

IF(n>0) THEN

DISPLAY "n is positive"

ELSE

DISPLAY "n is positive"

END IF

END IF

END

Write an algorithm to print all natural numbers up to n

Step 1: Start

Step 2: get n value.

Step 3: initialize i=1

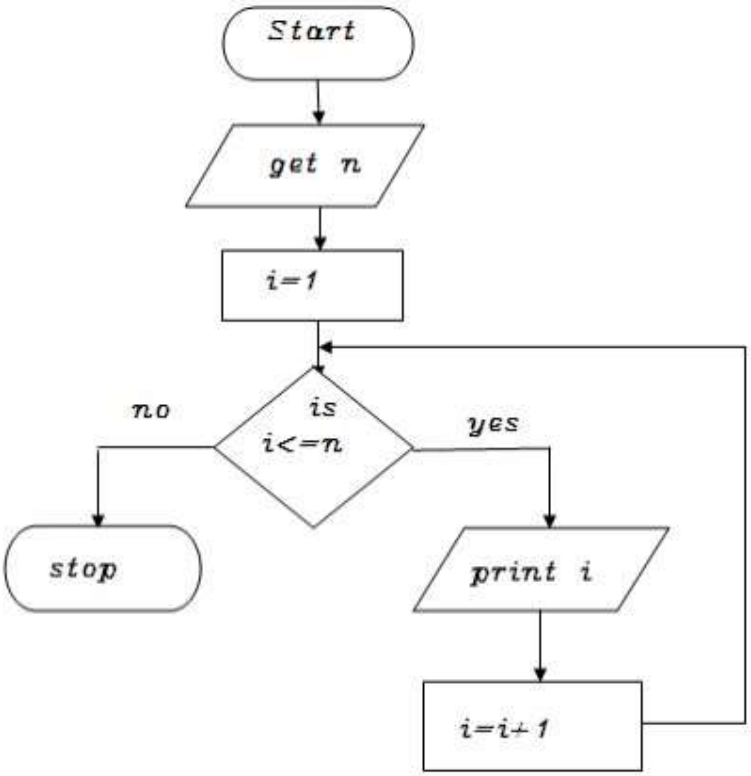
Step 4: if (i<=n) go to step 5 else go to step 8

Step 5: Print i value

step 6 : increment i value by 1

Step 7: go to step 4

Step 8: Stop



BEGIN

GET n

INITIALIZE i=1

WHILE(i<=n) DO

PRINT i

i=i+1

ENDWHILE

END

Write an algorithm to print n odd numbers

Step 1: start

step 2: get n value

step 3: set initial value i=1

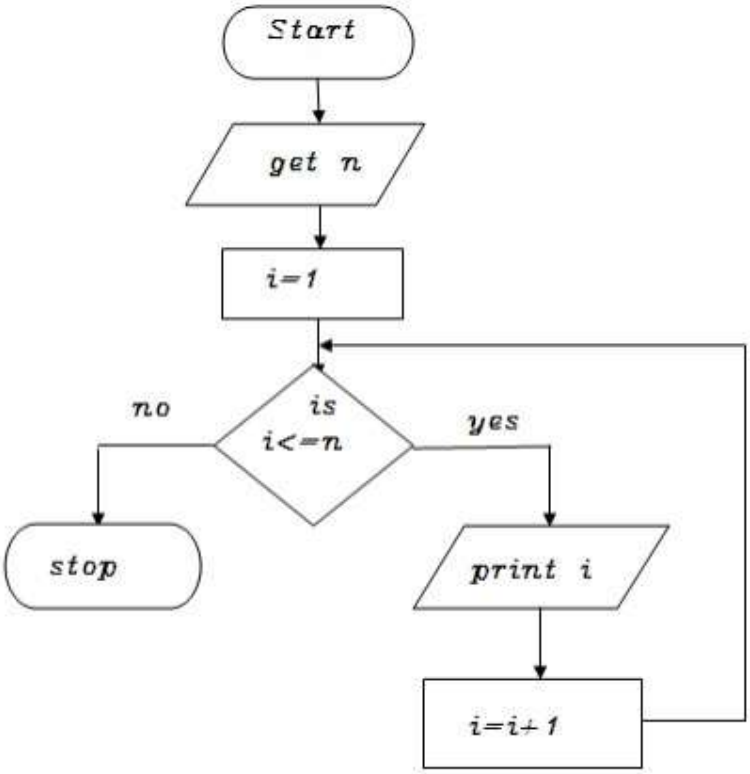
step 4: check if(i<=n) goto step 5 else goto step 8

step 5: print i value

step 6: increment i value by 2

step 7: goto step 4

step 8: stop



BEGIN

GET n

INITIALIZE i=1

WHILE(i<=n) DO

    PRINT i

    i=i+2

ENDWHILE

END

Write an algorithm to print n even numbers

Step 1: start

step 2: get n value

step 3: set initial value i=2

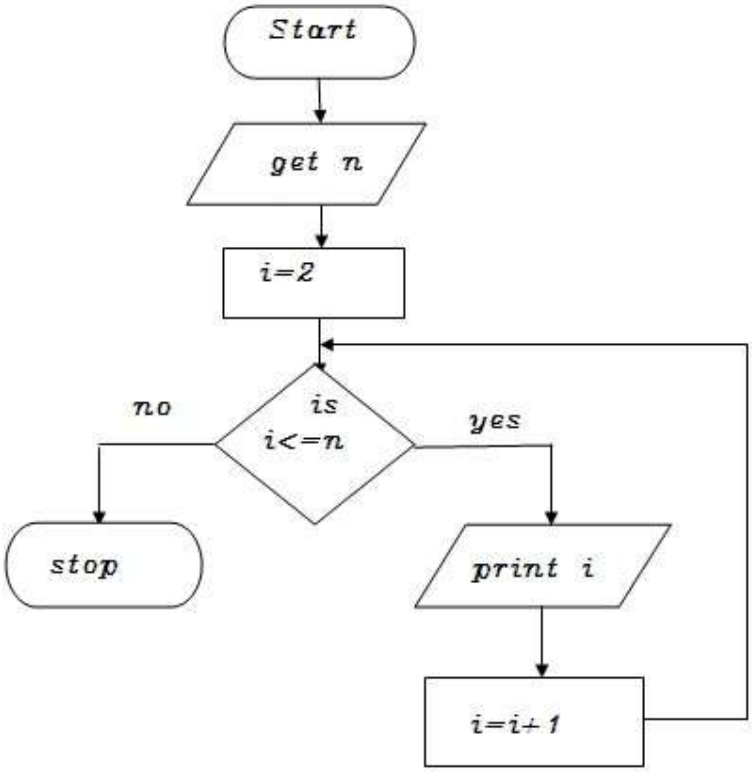
step 4: check if(i<=n) goto step 5 else goto step8

step 5: print i value

step 6: increment i value by 2

step 7: goto step 4

step 8: stop



BEGIN

GET n

INITIALIZE i=2

WHILE(i<=n) DO

    PRINT i

    i=i+2

ENDWHILE

END

Write an algorithm to print squares of a number

Step 1: start

step 2: get n value

step 3: set initial value i=1

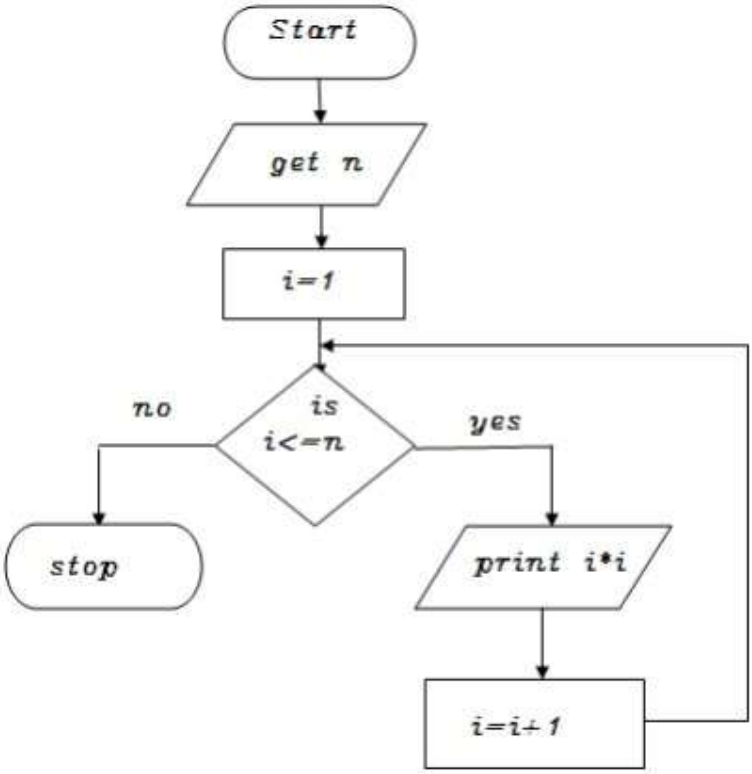
step 4: check i value if(i<=n) goto step 5 else goto step8

step 5: print i*i value

step 6: increment i value by 1

step 7: goto step 4

step 8: stop



BEGIN

GET n

INITIALIZE i=1

WHILE(i<=n) DO

    PRINT i*i

    i=i+2

ENDWHILE

END

Write an algorithm to print to print cubes of a number

Step 1: start

step 2: get n value

step 3: set initial value i=1

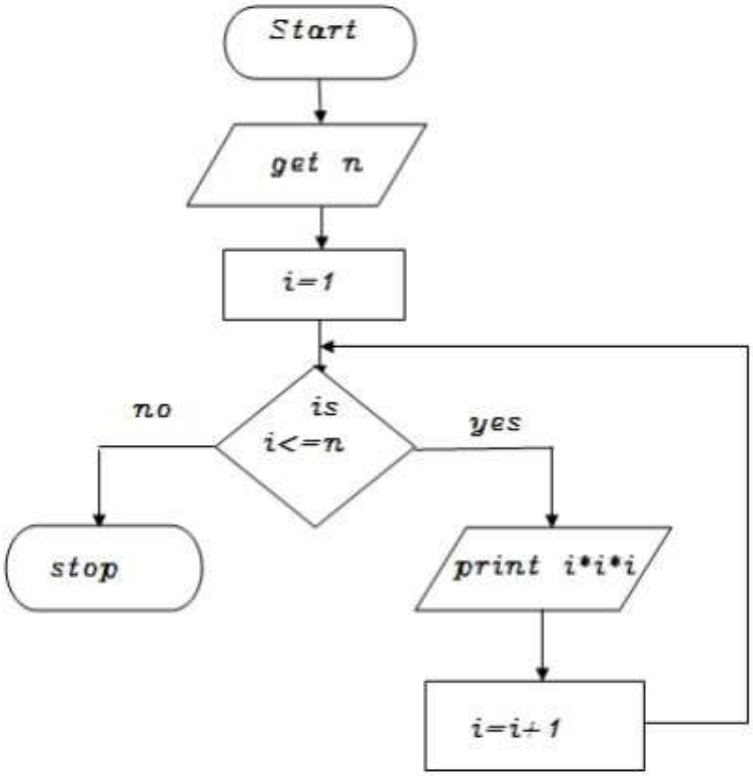
step 4: check i value if(i<=n) goto step 5 else goto step8

step 5: print i*i *i value

step 6: increment i value by 1

step 7: goto step 4

step 8: stop



BEGIN

GET n

INITIALIZE i=1

WHILE(i<=n) DO

    PRINT i*i*i

    i=i+2

ENDWHILE

END

Write an algorithm to find sum of a given number

Step 1: start

step 2: get n value

step 3: set initial value i=1, sum=0

Step 4: check i value if(i<=n) goto step 5 else goto step8

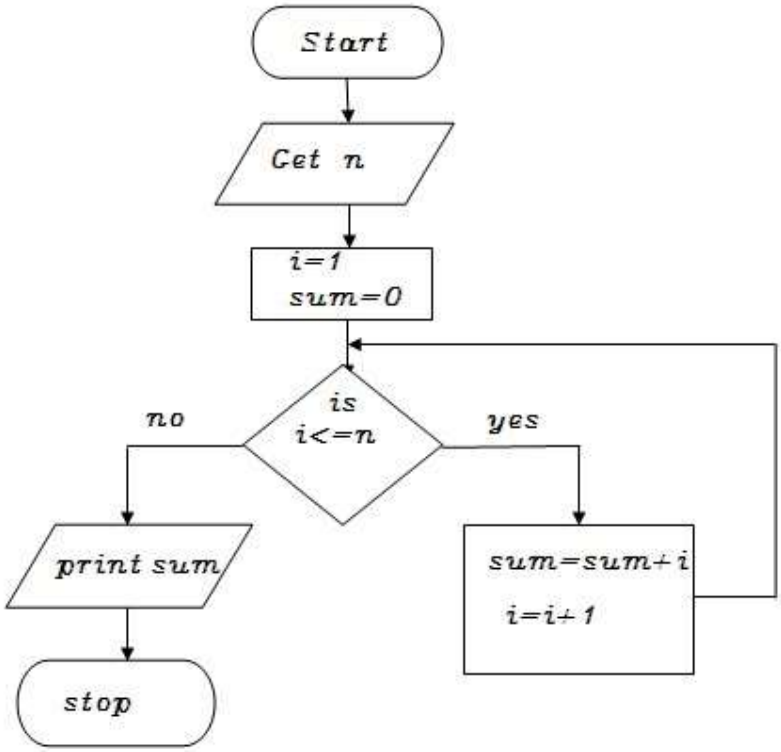
step 5: calculate sum=sum+i

step 6: increment i value by 1

step 7: goto step 4

step 8: print sum value

step 9: stop



BEGIN

GET n

INITIALIZE i=1,sum=0

WHILE(i<=n) DO

    sum=sum+i

    i=i+1

ENDWHILE

PRINT sum

END

Write an algorithm to find factorial of a given number

Step 1: start

step 2: get n value

step 3: set initial value i=1, fact=1

Step 4: check i value if(i<=n) goto step 5 else goto step8

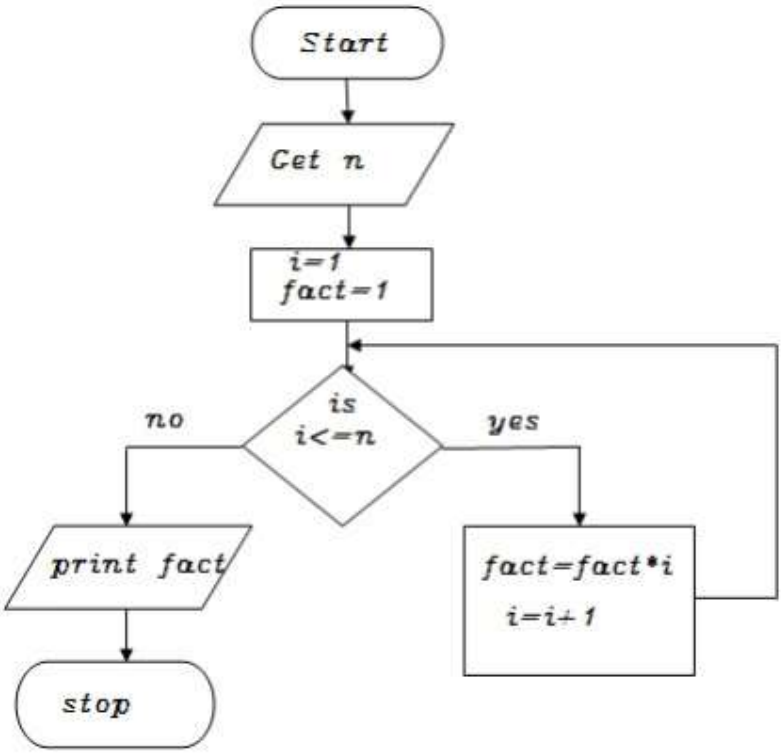
step 5: calculate fact=fact*i

step 6: increment i value by 1

step 7: goto step 4

step 8: print fact value

step 9: stop



BEGIN

GET n

INITIALIZE i=1,fact=1

WHILE(i<=n) DO

fact=fact*i

i=i+1

```
ENDWHILE
PRINT fact
END
```