CO101 Programming Fundamentals

- Flowchart and Algorithm -

Flowcharts

- **Diagrammatic/pictorial representation** of an algorithm.
- Can be used for writing programs and explaining the program to others.
- Makes use of symbols which when connected together, indicates the flow of information and processing.

Symbols used in Flowchart

1. **Flow Lines**: 

- Arrows represent the direction of flow of control and relationship among different symbols of flowchart.
- Indicate exact sequence in which instructions are executed.

2. **Terminal:** 

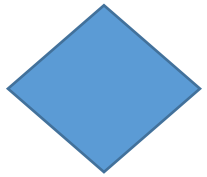
- Oval symbol indicates Start, Stop and Halt in a program's logic flow.
- A pause/halt is generally used in a program logic under some error conditions.
- Terminal is the first and last symbols in the flowchart.

3. **Input/Output:** 

- Parallelogram depicts input/output.
- Program instructions that take input from input devices and display output on output devices are indicated with parallelogram in a flowchart.

4. **Processing:** 

- Box/Rectangle is used to represent arithmetic instructions.
- All arithmetic operations (adding, subtracting, multiplication, etc.) are indicated by action or process symbol.

5. **Decision:** 

Diamond symbol represents a decision point.

Decision based operations (yes/no or true/false) are indicated by diamond in flowchart.

6. Connectors: 

- Represented by a circle.
- If flowchart becomes complex or it spans over multiple pages, it is useful to use connectors.

Advantages of Flowchart

- Flowcharts are better way of communicating the logic of system.
- Flowcharts act as a guide for blueprint during program designed.
- Flowcharts helps in debugging process.
- With the help of flowcharts programs can be easily analyzed.
- It provides better documentation.

Disadvantages of Flowchart

- It is difficult to draw flowchart for large and complex programs.
- In this there is no standard to determine the amount of detail.
- Difficult to reproduce the flowcharts.
- It is very difficult to modify the Flowchart.

Example Flowchart

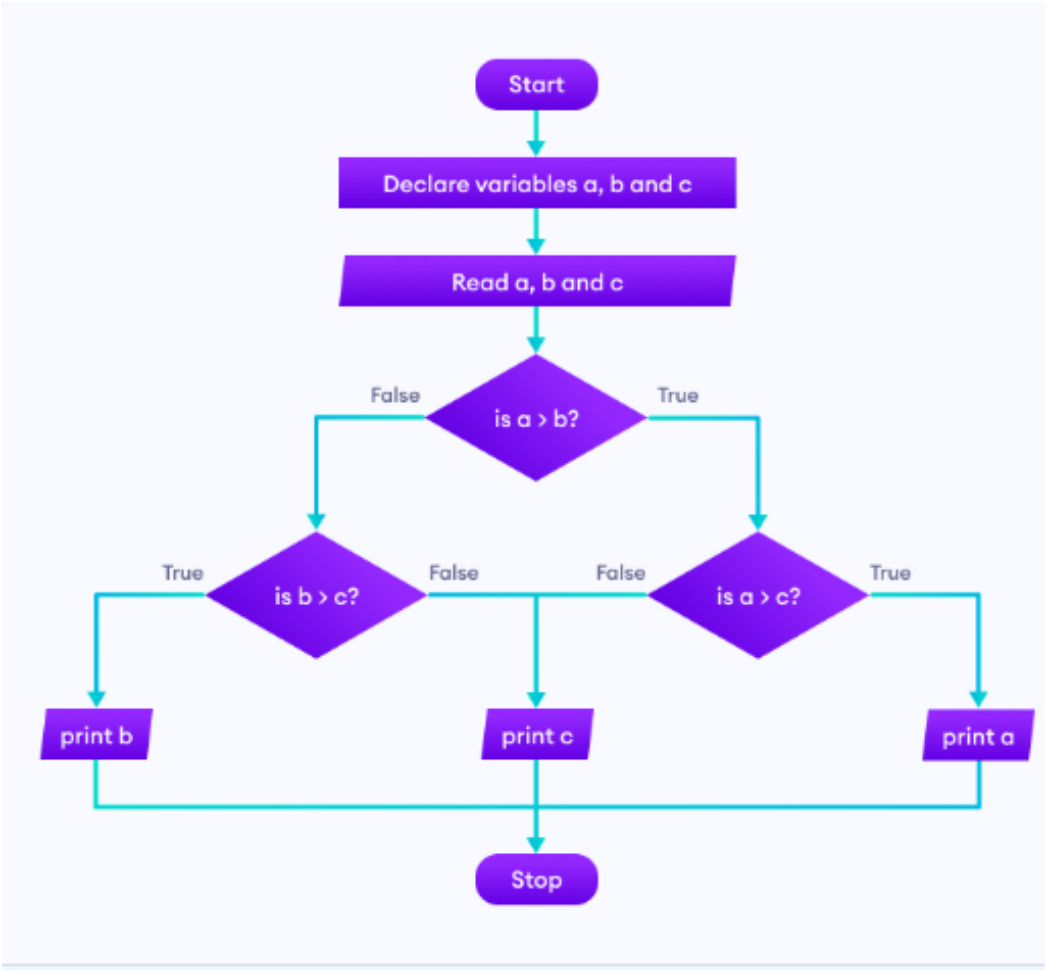
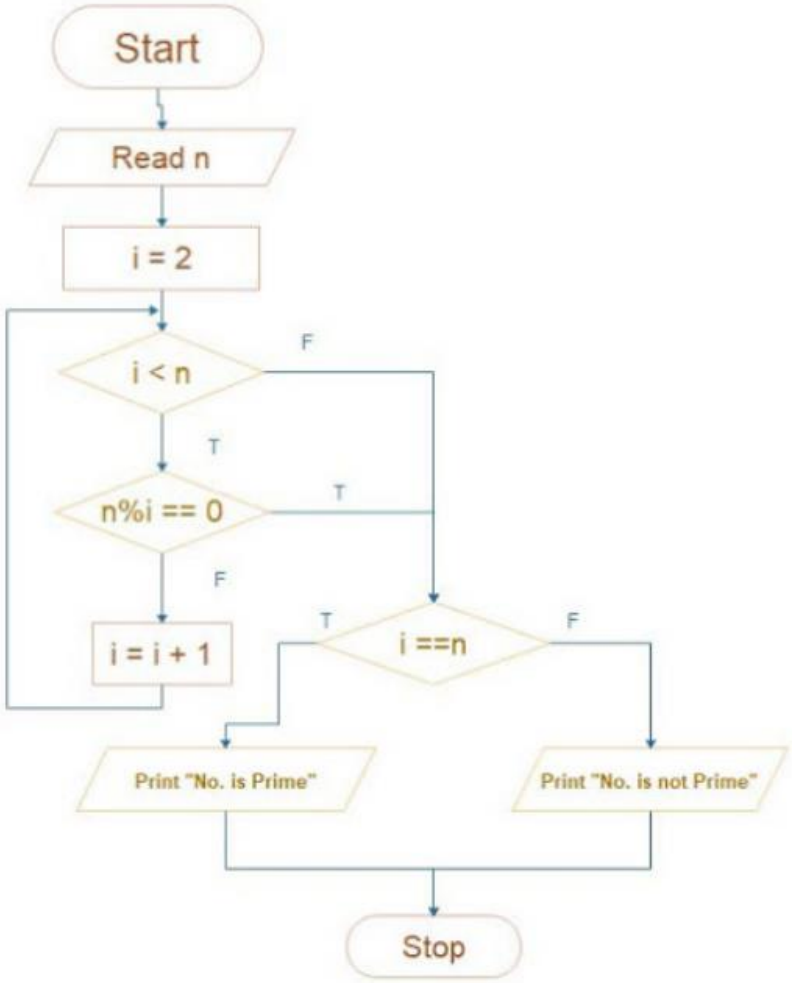
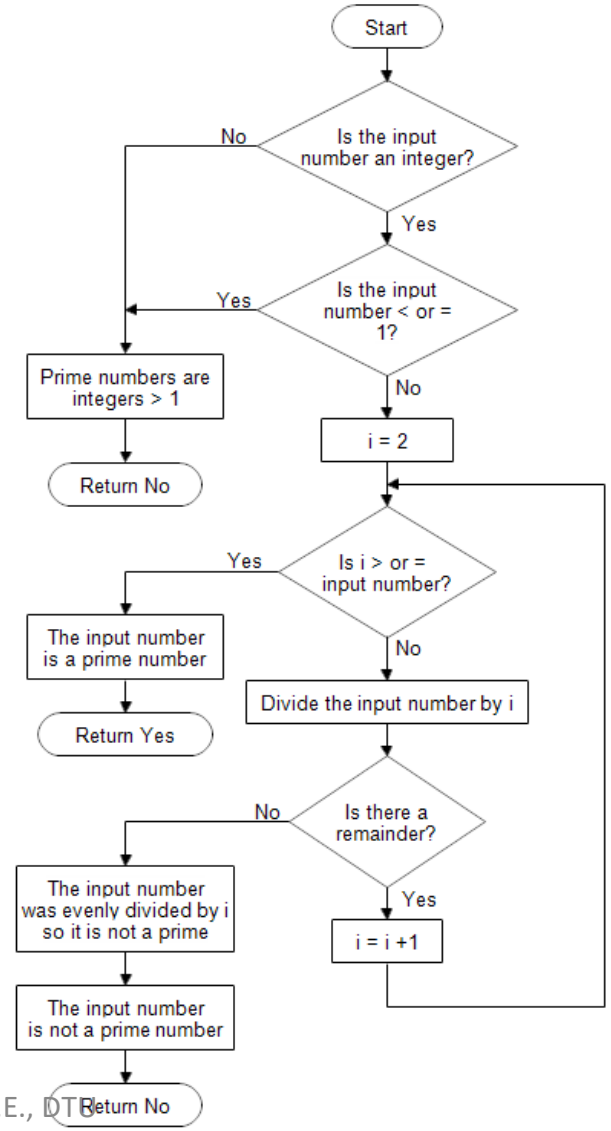


Fig1. Find the largest of three numbers

More examples:

- Find if a number is prime or not.





Algorithm

Definition:

An algorithm is a detailed step-by-step instruction set or formula for solving a problem or completing a task. In computing, programmers write algorithms that instruct the computer how to perform a task.

- Used to provide a solution to a particular problem in form of well-defined steps.
- Can be expressed using **natural language, flowcharts**, etc.

Characteristics of an algorithm

- **Unambiguous** − Algorithm should be clear and unambiguous.
- **Input** − An algorithm should have 0 or more well-defined inputs.
- **Output** − An algorithm should have 1 or more well-defined outputs, and should match the desired output.
- **Finiteness** − Algorithms must terminate after a finite number of steps.
- **Feasibility** − Should be feasible with the available resources.
- **Independent** − An algorithm should have step-by-step directions, which should be independent of any programming code.

Syntax Proposal

Generic imperative language that accepts recursive call

Control structures: indentation delimits the scope:

for all element ∈ Set do

for iterator = lowerbound to upperbound step increment do

while conditional do

do ... while conditional

if conditional then ... else

case element in value :

return value

break

continue

intructions: standard C++ syntax without pointers/reference

function call: standard C++ syntax without pointers/reference

exception: when your algorithm cannot safely terminate and/or respect the output specification

Example Algorithm

Problem: **Find Greatest Common Divisor (GCD) of two numbers**

input: integer a, b

output: greatest common divisor of a and b

```
if a = 0 then return b
while b >= 0 do
    if a > b then
        a = a−b else
        b = b−a
return a
```

More example:

- Algorithm to check if number is prime

```
step 1: start
step 2: [Accept a number] read n
step 3: set i = 2
step 4: Repeat steps 5and 6 until i < n
step 5: [ check whether n is divisible or not]
        if n % i == 0 then
           Goto step7
        Else
           Goto step6
step 6:  set i = i + 1
step 7:  if i == n then
          Print "number is prime"
       Else
          Print "number is not prime"
step 8: stop
```

```
Step 1: Start
Step 2: Read number n
Step 3: Set f=0
Step 4: For i=2 to n-1
Step 5: If n mod 1=0 then
Step 6: Set f=1 and break
Step 7: Loop
Step 8: If f=0 then
          print 'The given number is prime'
        else
          print 'The given number is not prime'
Step 9: Stop
```

Tips for writing algorithm

- Determine the input and output
- Find a correct data structure to represent the problem
- Convert the input to a suitable form, and to preprocess it.
- Try to reduce your problem to a variation of a well-known one Decide whether you look for a recursive algorithm or an imperative one, or a mix
- Write the algorithm
- Run all your examples on it, manually, before trying to prove it