INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

# CSN-101 (Introduction to Computer Science and Engineering)

## *Lecture 18: Problem Solving using Computers*

**Dr. Sudip Roy**

*Assistant Professor*

*Department of Computer Science and Engineering*

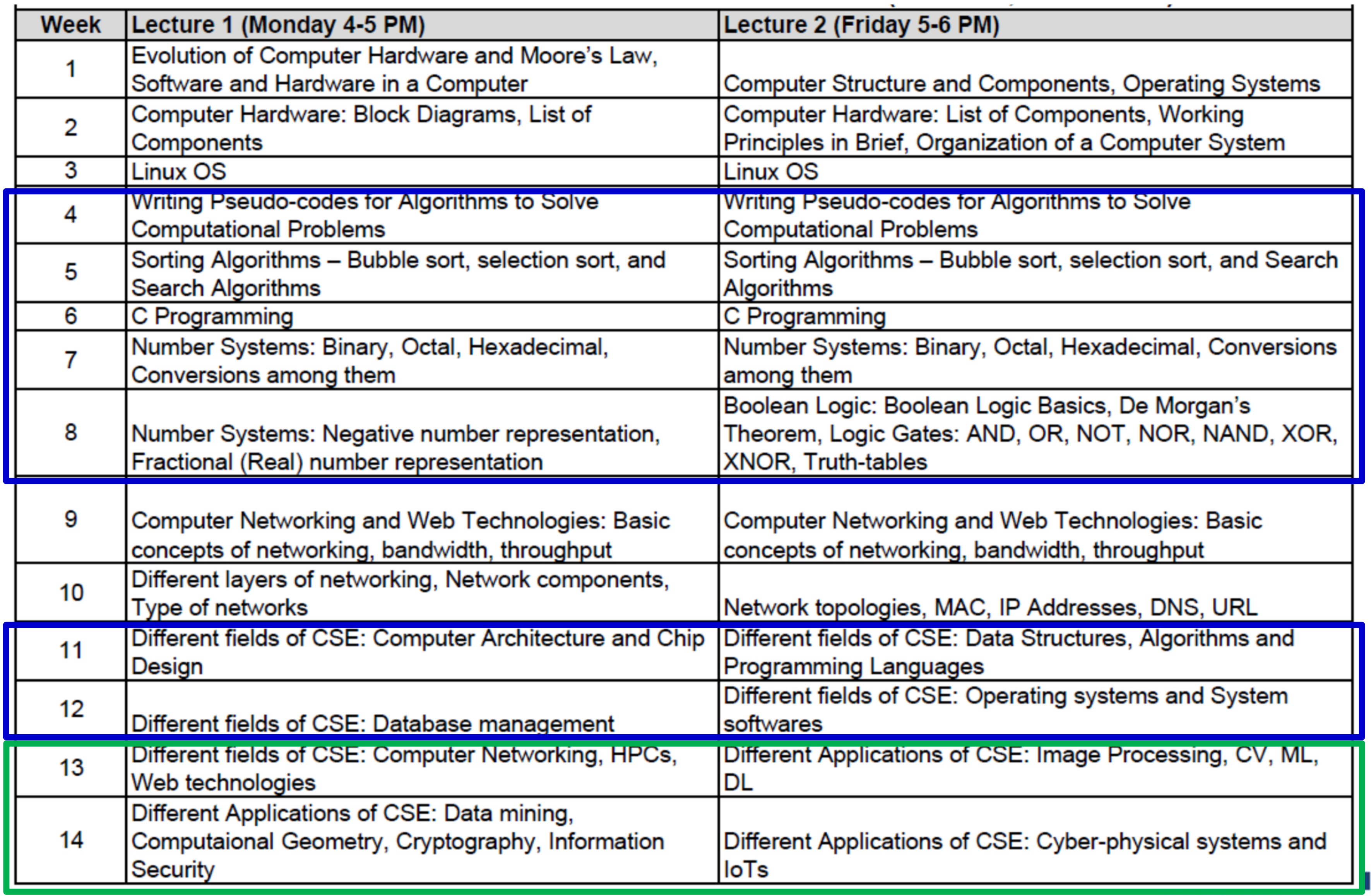
Piazza Class Room: https://piazza.com/iitr.ac.in/fall2019/csn101

[Access Code: csn101@2019]

Moodle Submission Site: https://moodle.iitr.ac.in/course/view.php?id=45

[Enrollment Key: csn101@2019]

# Plan for Lecture Classes in CSN-101 (Autumn, 2019-2020)



| Week | Lecture 1 (Monday 4-5 PM) | Lecture 2 (Friday 5-6 PM) |
|---|---|---|
| 1 | Evolution of Computer Hardware and Moore's Law, Software and Hardware in a Computer | Computer Structure and Components, Operating Systems |
| 2 | Computer Hardware: Block Diagrams, List of Components | Computer Hardware: List of Components, Working Principles in Brief, Organization of a Computer System |
| 3 | Linux OS | Linux OS |
| 4 | Writing Pseudo-codes for Algorithms to Solve Computational Problems | Writing Pseudo-codes for Algorithms to Solve Computational Problems |
| 5 | Sorting Algorithms – Bubble sort, selection sort, and Search Algorithms | Sorting Algorithms – Bubble sort, selection sort, and Search Algorithms |
| 6 | C Programming | C Programming |
| 7 | Number Systems: Binary, Octal, Hexadecimal, Conversions among them | Number Systems: Binary, Octal, Hexadecimal, Conversions among them |
| 8 | Number Systems: Negative number representation, Fractional (Real) number representation | Boolean Logic: Boolean Logic Basics, De Morgan's Theorem, Logic Gates: AND, OR, NOT, NOR, NAND, XOR, XNOR, Truth-tables |
| 9 | Computer Networking and Web Technologies: Basic concepts of networking, bandwidth, throughput | Computer Networking and Web Technologies: Basic concepts of networking, bandwidth, throughput |
| 10 | Different layers of networking, Network components, Type of networks | Network topologies, MAC, IP Addresses, DNS, URL |
| 11 | Different fields of CSE: Computer Architecture and Chip Design | Different fields of CSE: Data Structures, Algorithms and Programming Languages |
| 12 | Different fields of CSE: Database management | Different fields of CSE: Operating systems and System softwares |
| 13 | Different fields of CSE: Computer Networking, HPCs, Web technologies | Different Applications of CSE: Image Processing, CV, ML, DL |
| 14 | Different Applications of CSE: Data mining, Computaional Geometry, Cryptography, Information Security | Different Applications of CSE: Cyber-physical systems and IoTs |

ETE

ETE

Term Project

## How to solve a problem?

- Programming is a problem-solving activity.
- If you are a good problem solver, you could become a good programmer.
- Problem-solving methods are covered in many subject areas:

  - Business students learn to solve problems with a **systems approach**
  - Engineering and Science students use the **engineering and science methods**
  - Programmers use the **Software Development Method**

# **Software Development Method**

To solve a problem using a computer, you need to:

- Think carefully about the problem & its solution
  - Analyze the problem
  - Specify the problem requirements

- Plan it out beforehand (write an **algorithm**)
- Check it over to see that it addresses the problem
- Modify the solution if necessary
- Use the outlines to write a **program** that you can run on a computer
- Finally, the program is tested to verify that it behaves as intended.

# Introduction to Algorithms

- The **algorithm** is the abstract idea of solving a problem.
- An **algorithm** is a step-to-step problem solving process in which a solution is arrived at a finite amount of time.
- The **algorithm** is written in an algorithmic language (or a pseudo code).

# Algorithms vs. programs

- When an algorithm is coded using any programming language (e.g., C++), then it is called a **program**.

- The **program** is a set of instructions that can run by the computer.

# Characteristics of Algorithms

- It should be **accurate**:

  - It shouldn’t be ambiguous (each step in the algorithm should exactly determine the action to be done).

- It should be **effective**:

  - It shouldn’t include a step which is difficult to follow by a person (or the computer) in order to perform it.

- The algorithm should be **finite**:

  - the algorithm has to end up in a point at which the task is complete

# Algorithm (Contd…):

- To find largest of three numbers

```
1)  Start
2)  Read 3 numbers: num1, num2, num3
3)  if num1 > num2 then go to step 5
4)  if num2 > num3 then
                print num2 is largest
    else
                print num3 is largest
    goto step 6
5)  if num1 > num3 then
                print num1 is largest
    else
                print num3 is largest
6)  end.
```

# Algorithm (Contd…):

**Example:** One of the simplest algorithms is to find the largest number in an (unsorted) list of numbers.

**High-level description:**

1) Assume the first item is largest.
2) Look at each of the remaining items in the list and if it is larger than the largest item so far, make a note of it.
3) The last noted item is the largest in the list when the process is complete.

# Algorithm (Contd…):

**Formal description:** Written in prose but much closer to the high-level language of a computer program, the following is the more formal coding of the **algorithm in pseudo code (find the largest number in an (unsorted) list of numbers)**

**Algorithm** LargestNumber

Input: A non–empty list of numbers *L*.

Output: The *largest* number in the list *L*.

```
largest ← $L_0$
for each item in the list L, do
    if the item > largest, then
        largest ← the item
return largest
```

# Algorithm (Contd…):

**Formal description:** Written in prose but much closer to the high-level language of a computer program, the following is the more formal coding of the **algorithm in pseudo code (find the largest number in an (unsorted) list of numbers)**

**Algorithm** LargestNumber

**Input:** A non-empty list of numbers *L*.

**Output:** The *largest* number in the list *L*.

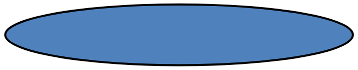
```
largest ← $L_0$
for each item in the list L, do
    if the item > largest, then
        largest ← the item
return largest
```

# Flowchart

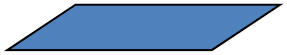
- It is another way to display the algorithm.
- It is composed of special geometric symbols connected by lines and contain the instructions.
- You can follow the lines from one symbol to another, executing the instructions inside it.

# Flowchart Symbols

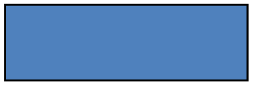
- Start / End symbol



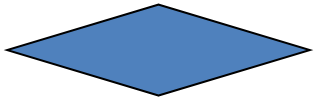
- Input/Output symbol



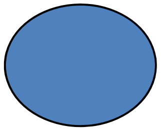
- Processing symbol



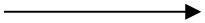
- Condition & decision symbol



- Continuation (connection symbol)



- Links



# Flowchart:

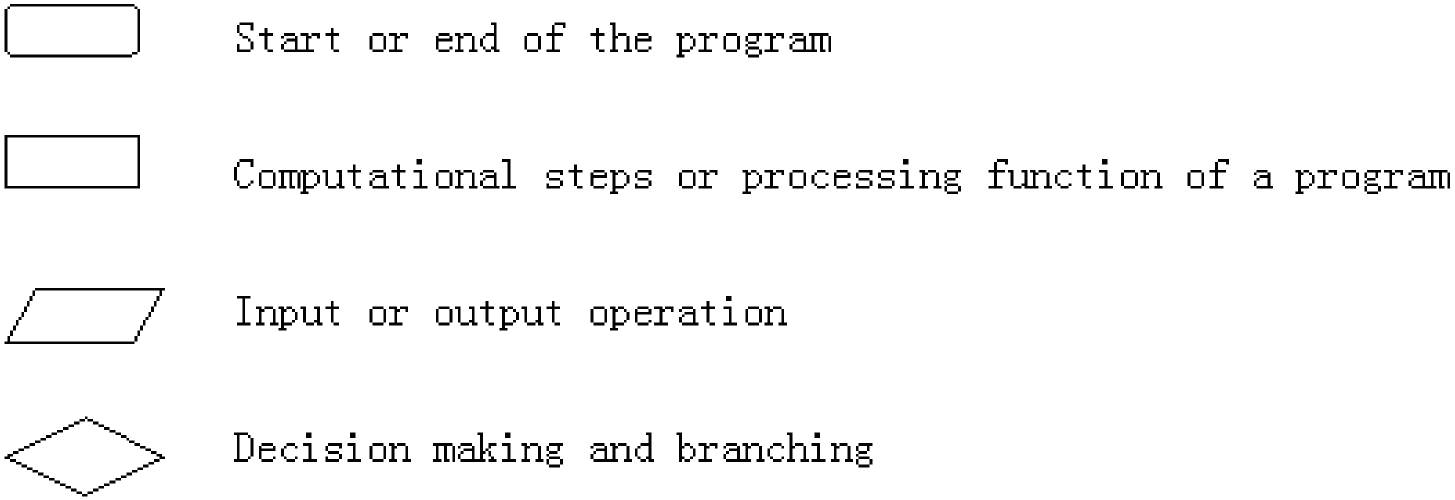
## What is a Flowchart?

- The **flowchart** is a means of visually presenting the flow of control through an information processing systems, the operations performed within the system and the sequence in which they are performed.
- It is a graphic representation of how a process works, showing, at a minimum, the sequence of steps.
- Flowcharts are generally drawn in the early stages of formulating computer solutions.

# Flowchart (Contd…):

## Guideline for drawing a flowchart:

Flowcharts are usually drawn using some standard symbols; Some standard symbols, which are frequently required for flowcharting many computer programs are shown below,-



Start or end of the program

Computational steps or processing function of a program

Input or output operation

Decision making and branching

# Flowchart (Contd…):

## A set of useful standard Flowchart symbols:

- **Rounded box**

  use it to represent an event which occurs automatically.
- **Rectangle or box**

  use it to represent an event which is controlled within the process. Typically this will be a step or action which is taken.
- **Diamond**

  use it to represent a decision point in the process.
- **Circle**

  use it to represent a point at which the flowchart connects with another process.

# ADVANTAGES OF USING FLOWCHARTS:

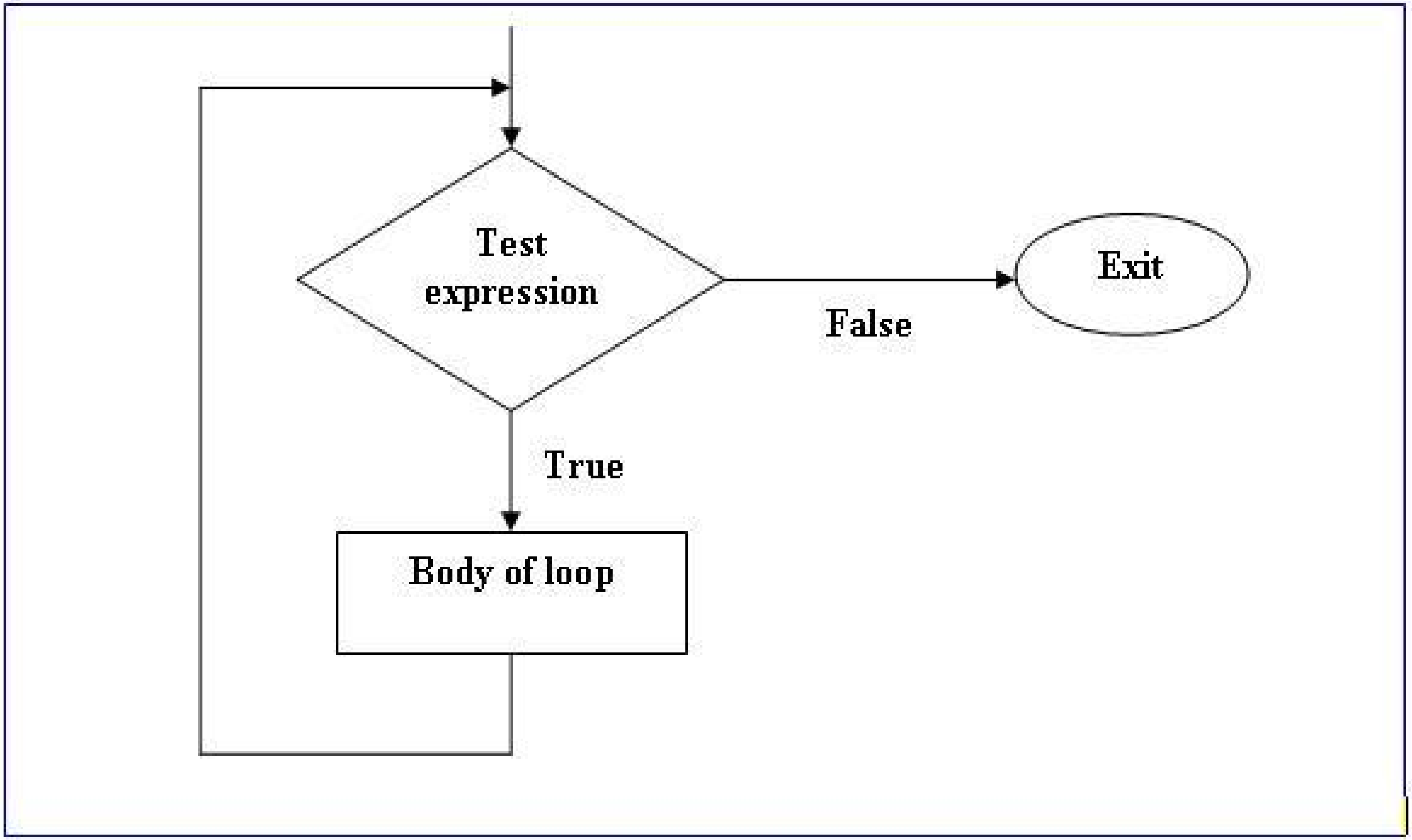
- **Communication:** Flowcharts are better way of communicating the logic of a system
- **Effective analysis:** Problem can be analyzed in more effective way.
- **Proper documentation:** Flowcharts serve as a good program documentation
- **Efficient Coding:** Flowcharts act as a guide or blueprint during the systems analysis and program development phase.

# ADVANTAGES OF USING FLOWCHARTS (Contd…):

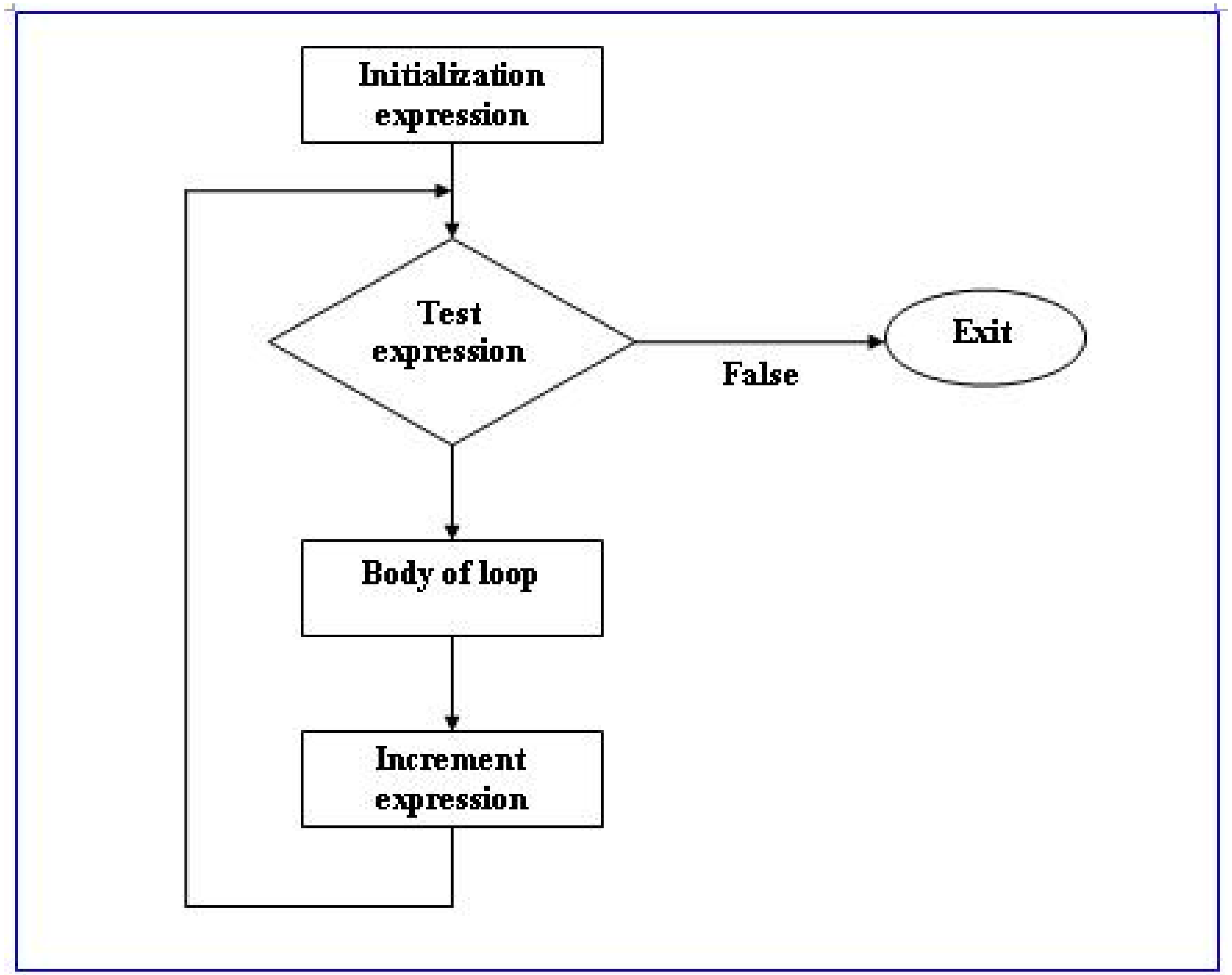
- **Proper Debugging:** Flowchart helps in debugging process.

- **Efficient Program Maintenance:** The maintenance of operating program becomes easy with the help of flowchart.

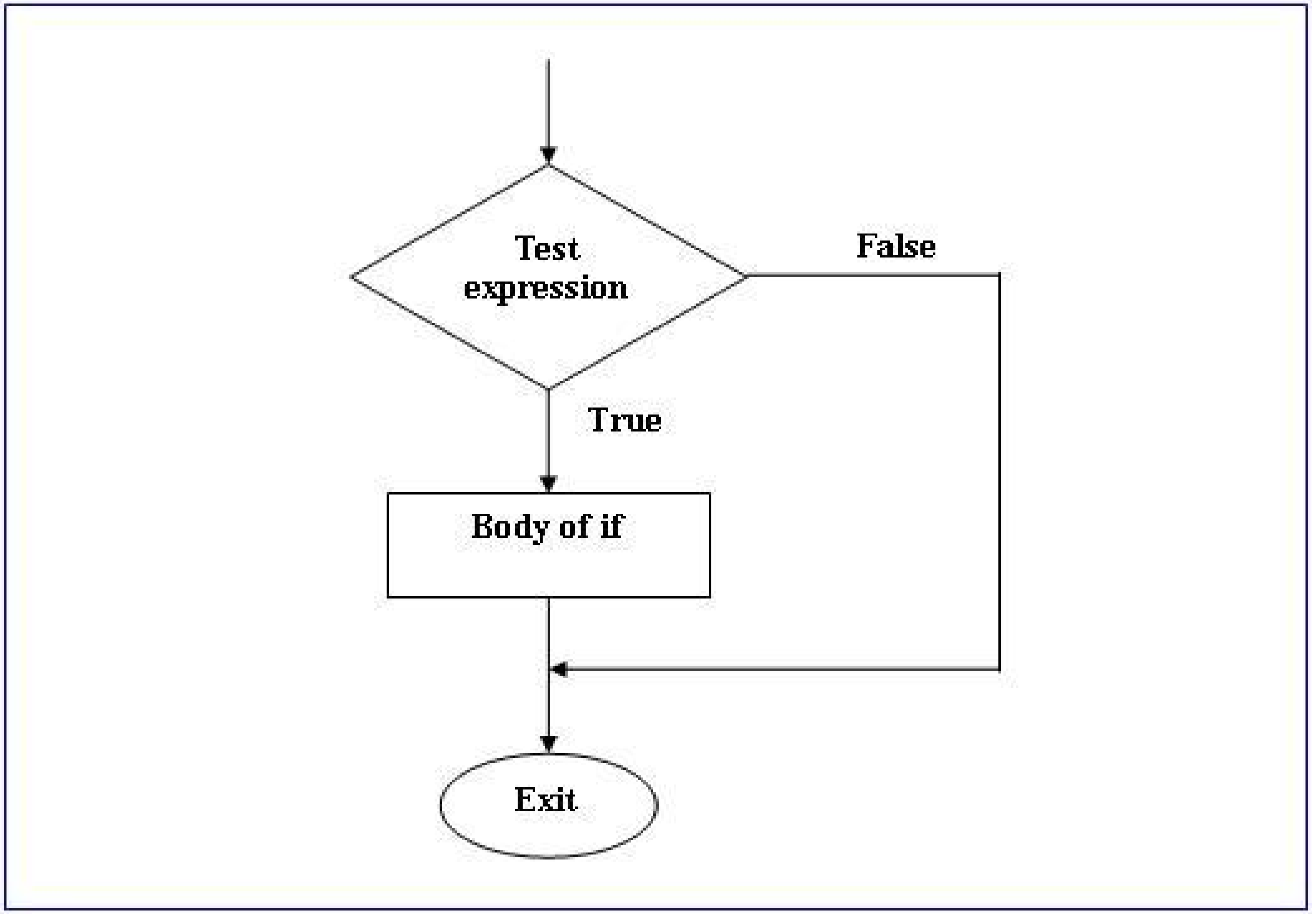
# Flow chart of the **while** loop :



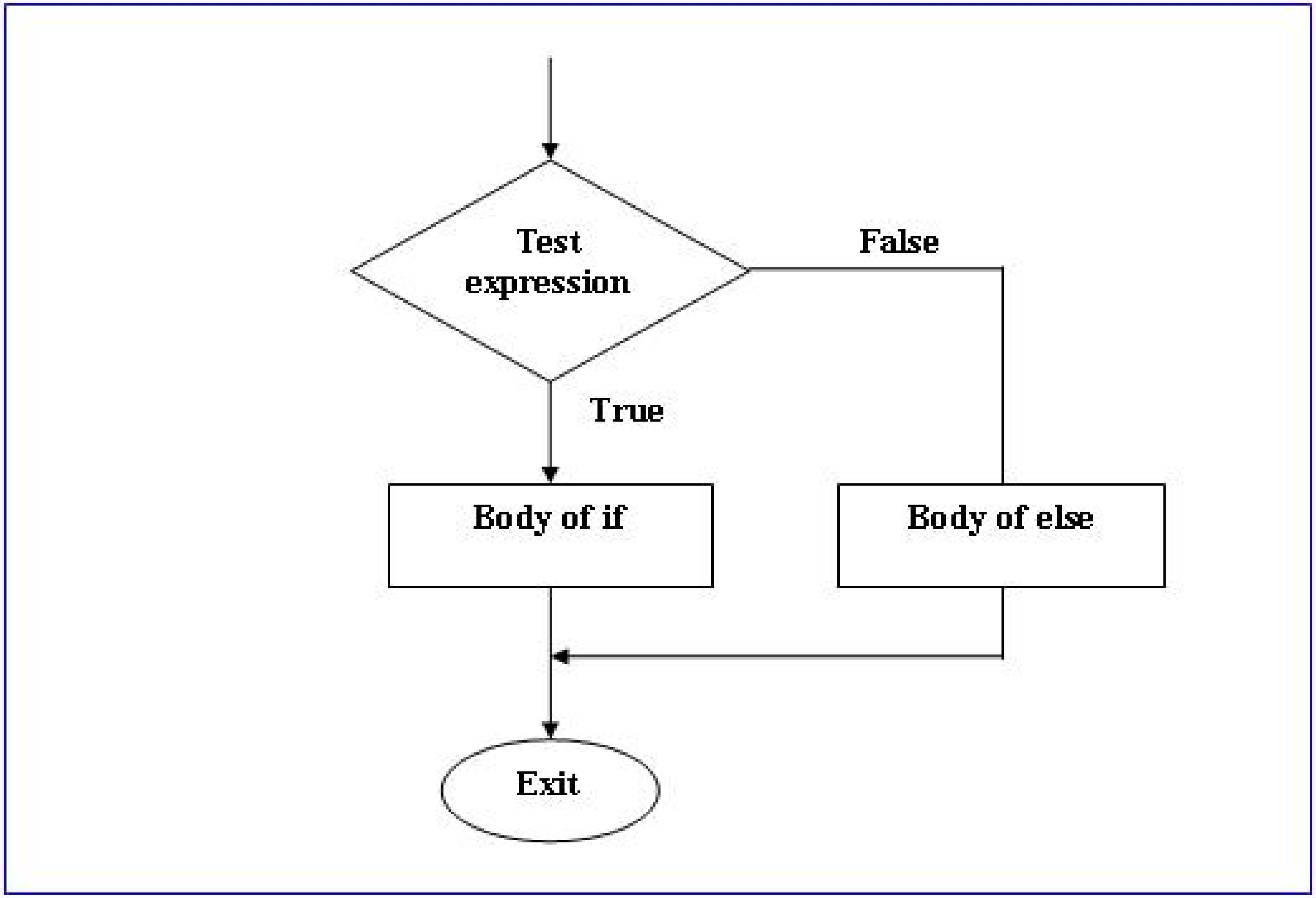
# Flow chart of the **for** loop:



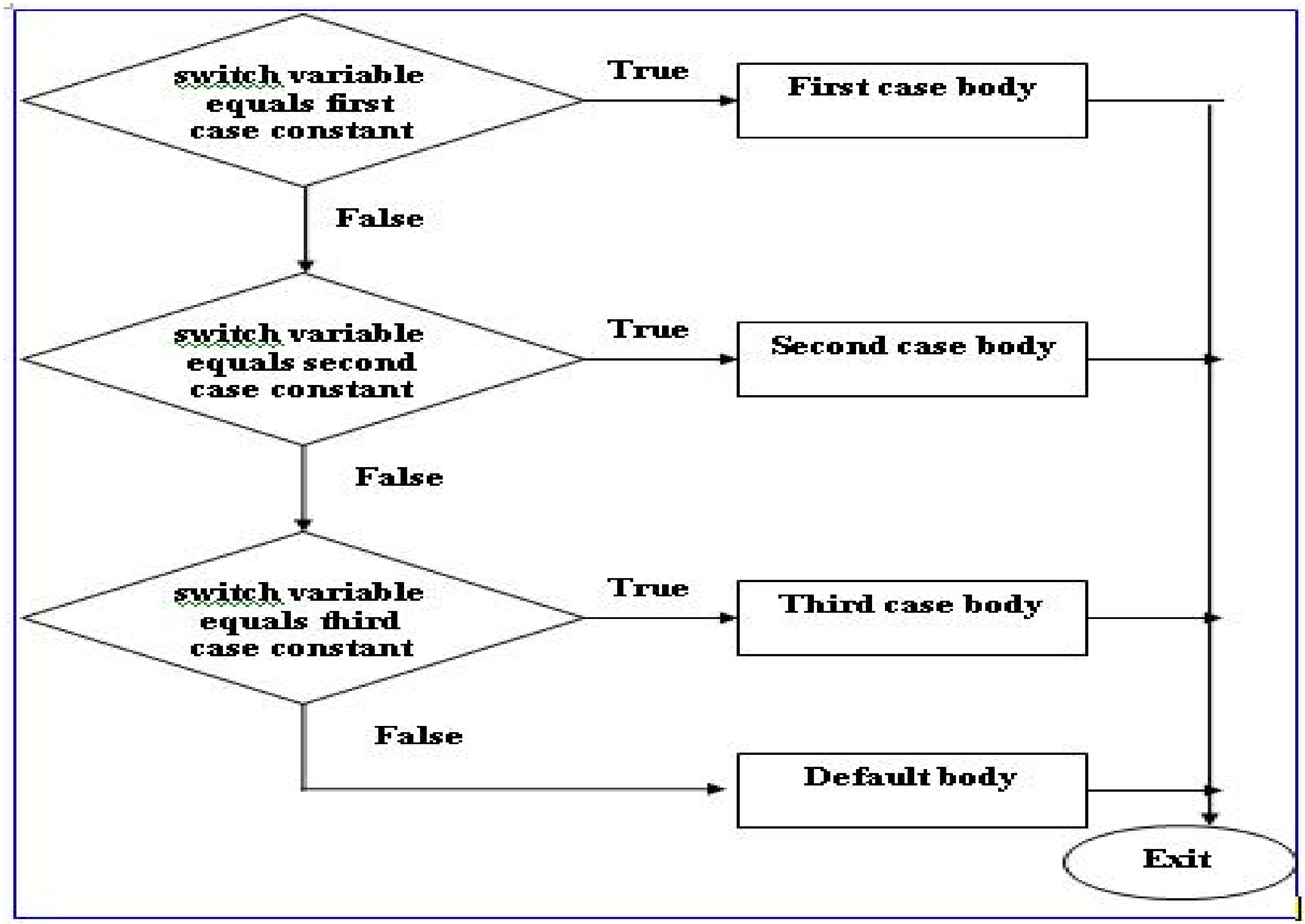
# The flow chart of the **if** statement:



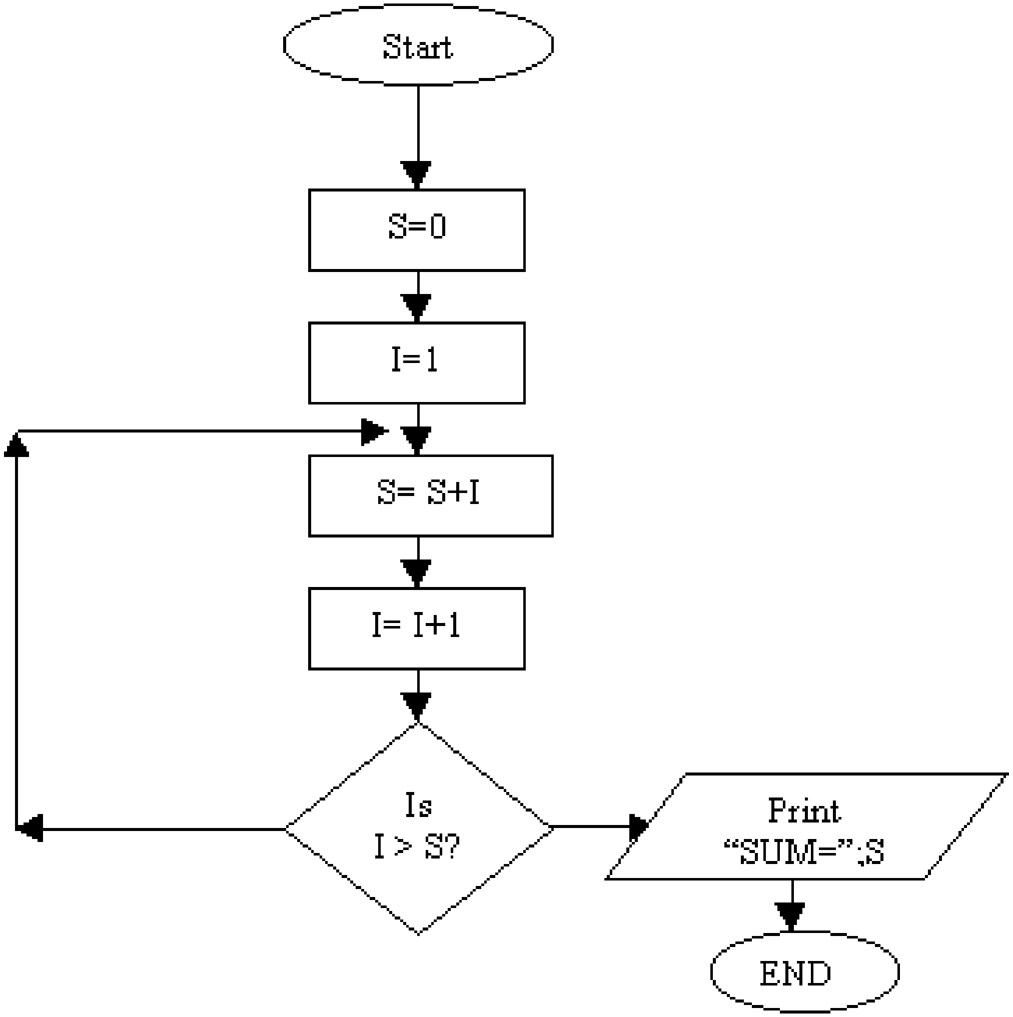
# The flow chart of the **if…else** statement:



# The flow chart of the switch statement:

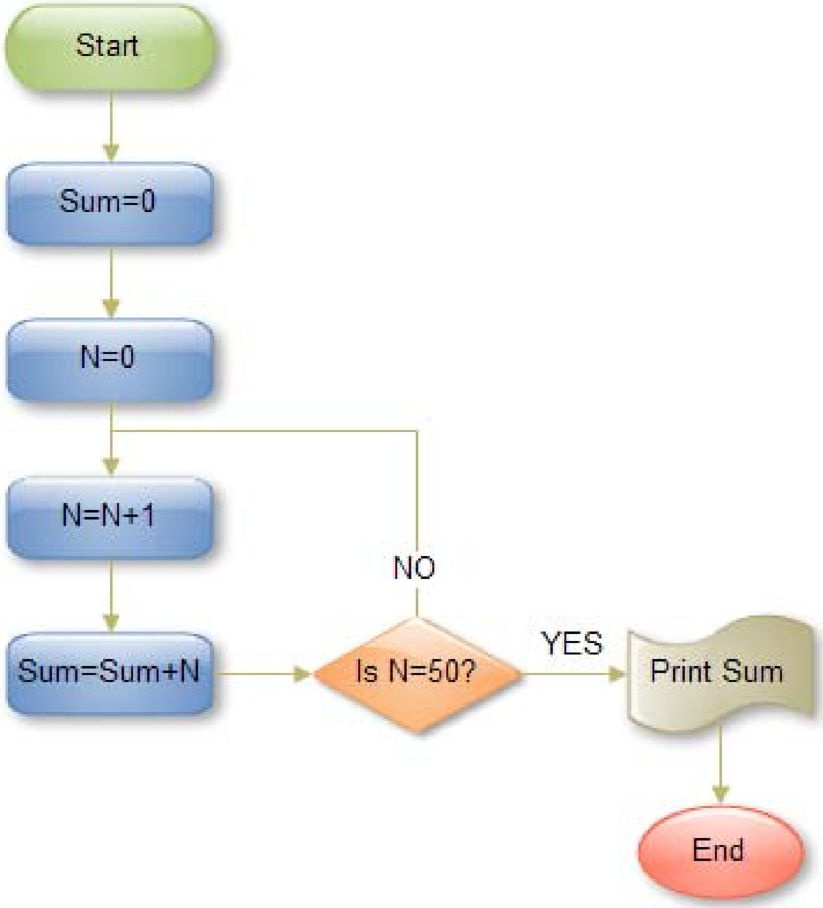

switch variable equals first case constant
True
First case body
False
switch variable equals second case constant
True
Second case body
False
switch variable equals third case constant
True
Third case body
False
Default body
Exit

# Flowchart for finding the sum of first five natural numbers (i.e., 1,2,3,4,5):

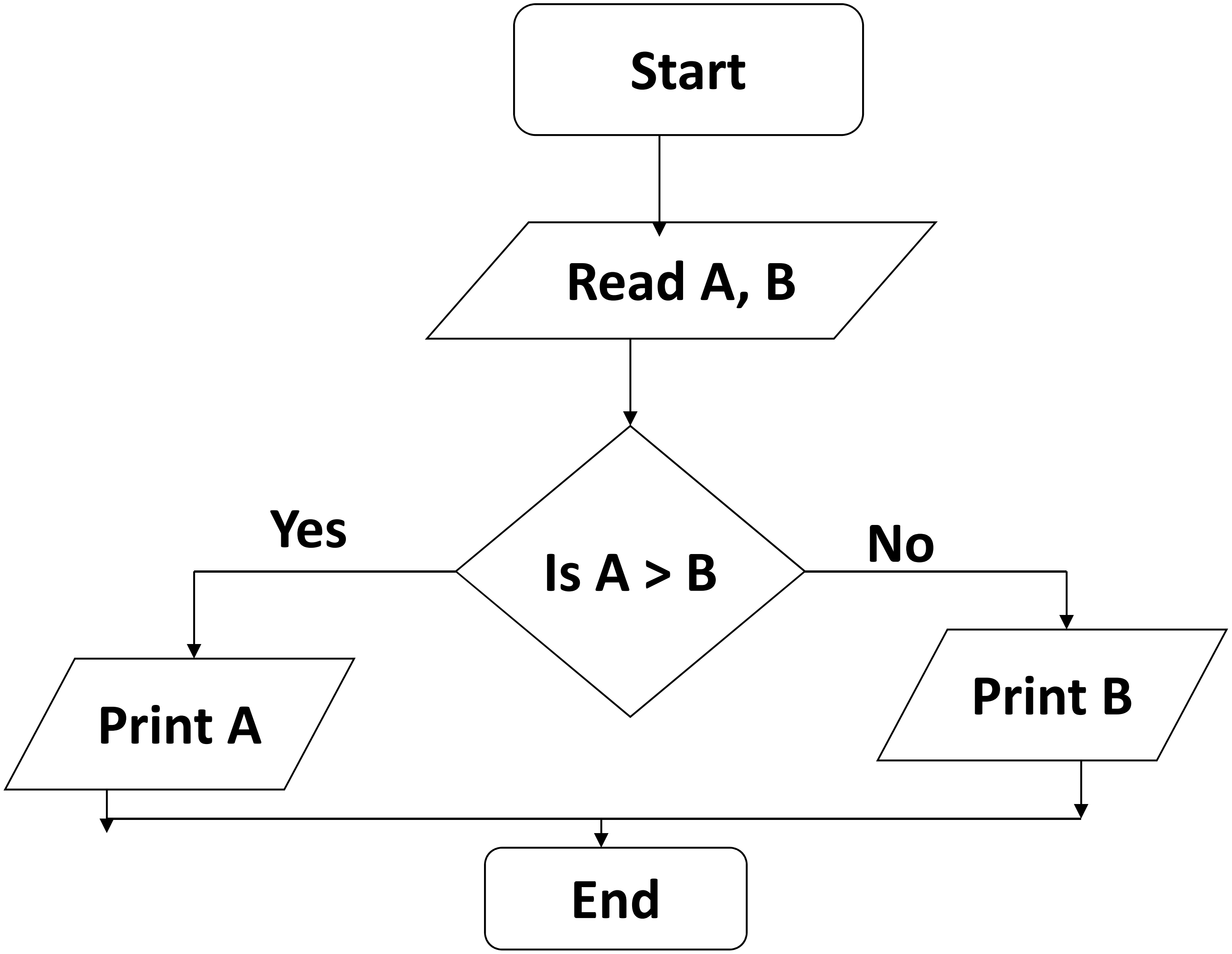


# Flowchart (Example):

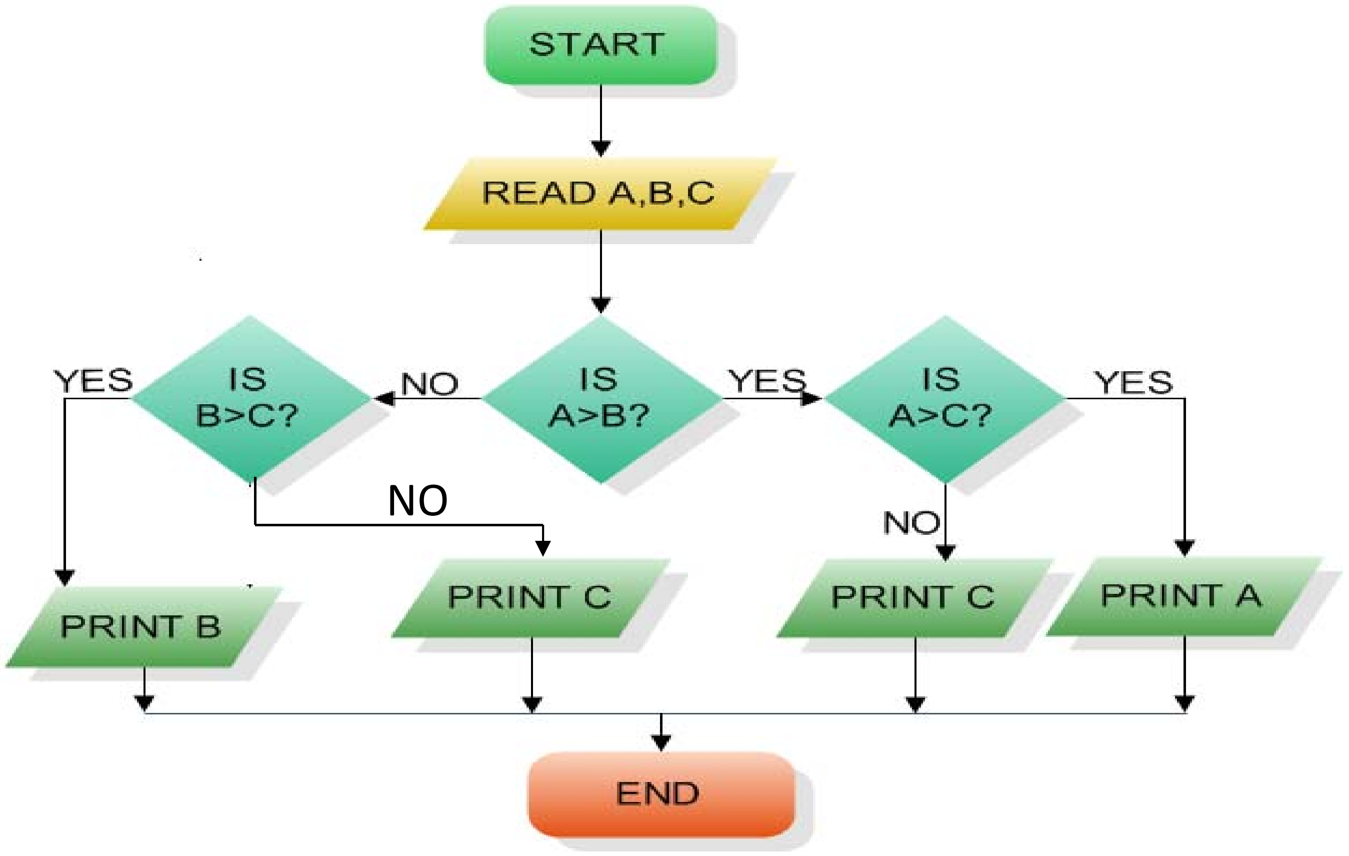
Flowchart to find the sum of first 50 natural numbers.



# Flow Chart to find largest of two numbers:



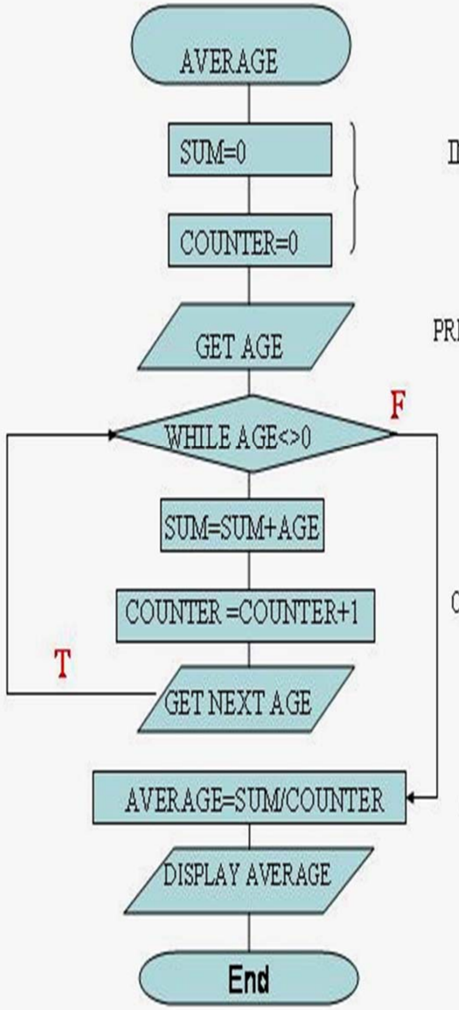
# Flowchart to find the largest of three numbers A,B, and C:



# Algorithm and Flowchart Using Automatic Counter Loop:

ALGORITHM

- Set sum to zero
- Set counter to zero
- Get age (priming Read)
- WHILE age <> 0
  - Sum = sum + age
  - Counter = counter + 1
  - Get next age
- WHILE END
- Average =sum/counter
- Display average
- End



INITIALISES SUM AND COUNTER TO 0

PRIMER READ (ENTER FIRST AGE)

VALUE AGE = 0 STOPS LOOP

ACCUMULATES AGE

COUNT THE NUMBER OF ENTRIES

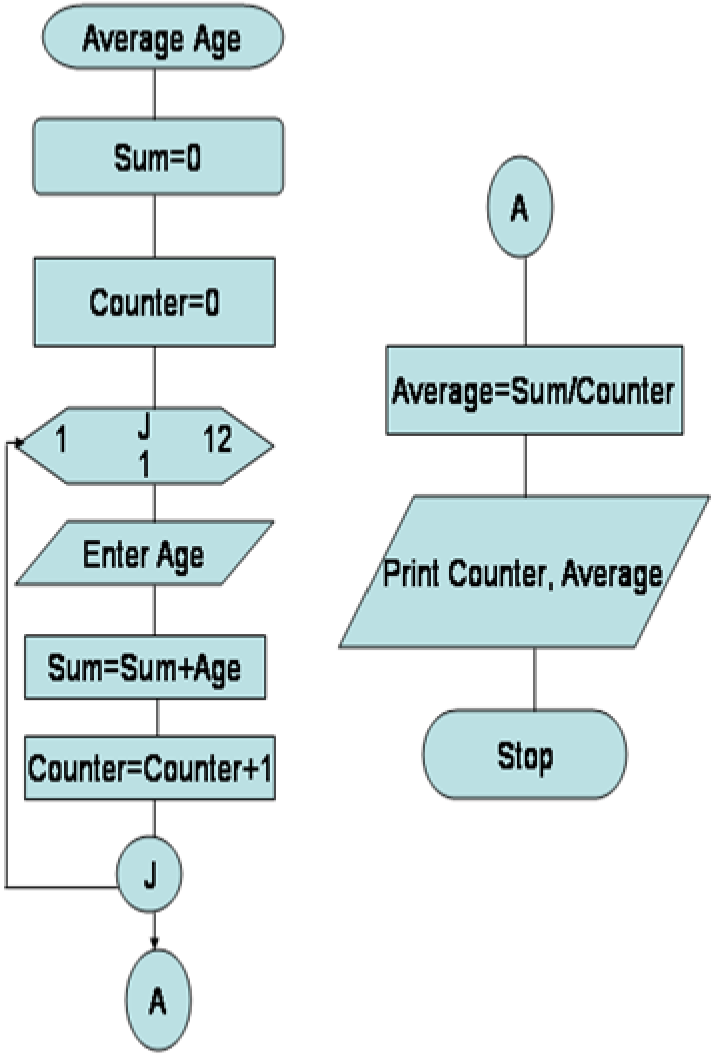
GET NEXT AGE

CALCULATES AVERAGE

DISPLAY AVERAGE

# Algorithm and Flowchart Using Automatic Counter Loop:

```
1.AverageAge
2.Sum=0
Counter=0
3.Loop:J=1 to 12
    Enter Age
    Sum= Sum +Age
    Counter=Countet+1
 Loop-End: J
4.Average=Sum/Counter
5.Print Counter, Average
6.End
```



# LIMITATIONS OF USING FLOWCHARTS:

- **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.

# Flowchart (Exercise):

1. Draw a flowchart to depict all steps that you do reach LHC-005 from your Hostel.

2. Draw Flowchart for Linear search of a number in a list of numbers.

# Problem Solving and Programming Strategy

- Programming is a process of problem solving.
- The problem is solved according to the problem domain (e.g., students, money).
- To be a good problem solver and hence a good programmer, you must follow good problem solving technique.

One problem solving technique to solve the problem includes

- **analyzing** the problem & outlining the problem’s requirements,
- **designing** steps (writing an algorithm)
- **implementing** the algorithm in a programming language (e.g. C++) and verify that the algorithm works,
- **maintaining** the program by using and modifying it if the problem domain changes.

## (a) Problem Analysis

1- Thoroughly understand the problem

2- Understand the problem **specifications.**

3- If the problem is complex, you need to divide it into **sub-problems** and repeat steps (1& 2). That is, you need to analyze each sub-problem and understand its requirements.

**Specifications** can include the following:

- Does the problem require interaction with the user?
- Does the problem manipulate data?
  What is the input data & how it is represented?
- Does the problem produce output? How the results should be generated and formatted.
- What are the required formula for solution
- Is there any constraints on problem solution?

- An Example for Problem Specifications:

***Problem Statement***:

Determine the total cost of apples given the number of kilos of apples purchased and the cost per kilo of apples.

We can summarize the information contained in the problem statement as follows:

- ***Problem Input:***

    - Quantity of apples purchased (in kilos)

    - Cost per kilo of apples (in dinars per kilo)

- ***Problem Output:***

    - Total cost of apples (in dinars)

- ***Formula:***

Total cost = Number of kilos of apples × Cost per kilo

# (b) Algorithm Design and Testing

- Design an algorithm for the problem.
- If the problem is divided into smaller sub-problems, then design an algorithm for each sub-problem.

***How to write an Algorithm?***

An algorithm is a sequence of statements to perform some operations. You can write an algorithm with the following layout:

```
ALGORITHM   algorithm_name
        statements of algorithm
END  algorithm_name
```

- The algorithm would consist of at least the following tasks:

1- **Input** (Read the data)

2- **Processing** (Perform the computation)

3- **Output** (Display the results)

***Structured Design*:**

Dividing the problem into smaller sub-problems is called “**structured design**”, “**top-down design**”,

***Structured Programming*:**

In the structured design

- The problem is divided into smaller sub-problems
- Each sub-problem is analyzed
- A solution is obtained to solve the sub-problem
- The solutions of all sub-problems are combined to solve the overall problem.

This process of implementing a structured design is called “**structured programming**”

**Testing the Algorithm:**

- You need to check the algorithm for correctness
- Use sample data for algorithm testing

## (c) Coding

- **Coding:**
  - After verifying that the algorithm is correct, you can code it in any high-level programming language
  - The algorithm is now converted into a **program**

## (d) Executing the Program

- Compile the program (to check for syntax error)
- Run the program. If the execution doesn’t go well, then reexamine the code, the algorithm, or even the problem analysis.

# Advantages of Structured Programming

- Easy to discover errors in a program that is well analyzed and well designed.

- Easy to modify a program that is thoroughly analyzed and carefully deigned.