

# BACSE101

## Problem Solving Using Python

PROF. SARAH PRITHVIKA  
P.C.  
ASSISTANT PROFESSOR  
SCOPE  
VIT CHENNAI

# Contact Details

- Contact No: 9840623676
- Mail Id: sarahprithvika.pc@vit.ac.in
- WhatsApp group

25BACSE101

WhatsApp group



# Class Structure

- 

S.No	Hours	Day	Time	Venue
1.	L33 + L34	Monday	3.50 PM to 5.30 PM	AB1-205A
2.	L43 + L44	Wednesday	2.00 PM to 3.40 PM	AB1-205A

**Note:** It is mandatory to have minimum 75% attendance for the examinations

# Course Objectives

- To provide the basics and fundamentals of the python programming language and introduce **algorithmic** skills through programming constructs
- To impart python **collections** for efficient **data handling** and processing and inculcate modular and recursive programming approaches for developing software applications
- To teach python **libraries** for **data analysis and manipulations**

# Course Outcomes

- At the end of the course, students should be able to
  - Identify appropriate algorithmic approach, data representation, control constructs in developing software solutions for solving real-world problems.
  - Inculcate data handling techniques using python collections and modular programming strategies for developing multi-disciplinary software applications
  - Idealize the importance of python libraries for handling, manipulating and analyzing multi-faceted real world problem domains.

# Syllabus

<b>Module:1</b>	<b>Introduction to Problem Solving and Python Fundamentals</b>	<b>6 hours</b>
Fundamentals of Problem Solving: Problem Analysis Chart - Robust and Boundary Test Cases – Algorithm- Flowchart - Pseudocode Data Types: Numeric, Boolean and Strings - Regular Expression - Operators - Expressions - Built in Functions		
<b>Module:2</b>	<b>Problem Solving Approaches and Constructs for Controlling Program Flow</b>	<b>14 hours</b>
Problem Solving Approaches: Top-down - Bottom-up - Divide & Conquer - Backtracking Conditional Statements -Branching-Looping - Break, Continue and Pass Statements		
<b>Module:3</b>	<b>Data Organization and Manipulation using Collections</b>	<b>14 hours</b>
Data Handling Strategies Using Lists, Tuples, Dictionary and Set- Data Comprehension - Iterators - Data Selection - Ordered, Unordered and Unique Data Organization - Data Modification - Grouping and Categorization – Searching Techniques: Linear Search, Binary Search - Sorting Techniques: Bubble Sort, Selection Sort- Insertion Sort- Quick Sort- Merge Sort.		
<b>Module:4</b>	<b>Modular Programming</b>	<b>14 hours</b>
Fundamentals of Function: Definition, Call, Return Value – Parameters and Arguments: Positional arguments, Keyword arguments, Parameters with Default Values, Arbitrary Arguments – Local and Global Scope - Lambda Functions – Decorators - Recursive Functions- Menu driven implementations: Stacks- Queues.		
<b>Module:5</b>	<b>Data Processing using Numpy and Pandas</b>	<b>10 hours</b>
Numpy: Array Operations – Mathematical Functions - Pandas: Handling Files - Creating Dataframes – Data Cleaning – Filtering – Selection – Grouping – Sorting – Aggregation – Merging.		
<b>Module:6</b>	<b>Contemporary Issues</b>	<b>2 hours</b>

# Problem

- Generally a problem refers to a situation or challenge that requires a solution.
- Types of Problems
  - All Problems do not have a straightforward solution. Eg. How to be happy and prosperous always.
  - Some problems, such as making a dark room bright or baking a cake, can be solved with a series of actions.
  - These solutions are called algorithmic solutions.
  - There may be more than one solution for a problem
  - Identify all possible ways to solve a problem and choose one among them.

# Computational Problem

- A computational problem is a specific challenge or task that requires a solution through computational methods. This could be anything from sorting a list of numbers to analyzing data or automating a task.
- Computers are built to solve problems with algorithmic solutions.
- Solving a complicated calculus problem is an easy task for the computer.

.



# Problem solving

- Problem solving is the process of identifying a problem, developing a strategy to solve it, and implementing a solution. In programming, this involves breaking down the problem into smaller, manageable tasks and then writing code to address each task. Sometimes there may be alternate solutions to solve a problem.

## Steps in Problem Solving:

- **Understand the Problem:** Clearly define what needs to be solved.
- **Plan a Solution:** Decide how to solve the problem.
- **Implement the Solution:** Write code to execute the plan.
- **Test and Debug:** Run the code with various inputs to ensure it works correctly and fix any issues.
- **Optimize and Refine:** Improve the solution for efficiency and clarity.

# Types of Computational

where the answer for every instance is either yes or no.

## Decision Problem

Deciding whether a given number is prime

Searching an element from a given set of elements. Or arranging them in an order

## Searching & Sorting Problem

Finding product name for given product ID and arranging products in alphabetical order of names

Counting no. of occurrences of a type of elements in a set of elements

## Counting Problem

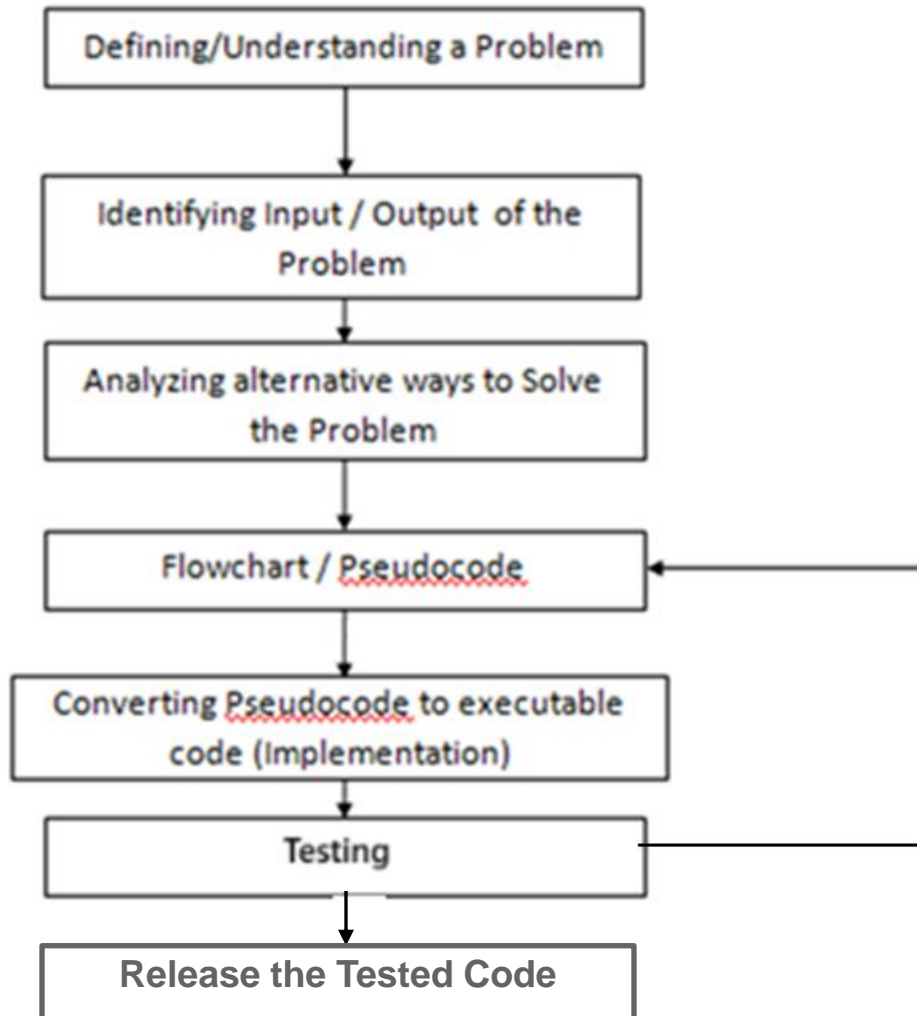
Counting how many different type of items are available in the store

Finding the best solution out of several feasible solutions

## Optimization Problem

Finding best combination of products for promotional campaign

# Problem Solving Life Cycle



# Two phases in problem solving

## – Phase 1:

- Organizing the problem or **pre-programming phase**.

## – Phase 2:

- Programming phase.

# **PRE-PROGRAMMING PHASE**

# PRE-PROGRAMMING PHASE

- **Analyzing The Problem**
  - Understand and analyze the problem to determine whether it can be solved by a computer.
  - Analyze the requirements of the problem.
  - Identify the following:
    - Data requirement.
    - Processing requirement or procedures that will be needed to solve the problem.
    - The output.

# Problem Solving Methods in pre-programming phase

- **Problem Analysis Chart** : Provides a high-level view of the problem and its components. Used in the early stages of project development to analyze and understand the problem before designing solutions.
- **Algorithm**: A finite sequence of steps required to get the desired output. Used in the later stages to design and implement solutions based on the problem analysis.
- **Flowchart**: This is a graphical representation of computation
- **Pseudo code**: Pseudocode does not use any programming language in its representation instead it uses the simple English language text as it is intended for human understanding.

# Problem Analysis Chart (PAC)

Problem Analysis Chart typically includes the concepts of **Input**, **Processing**, and **Output**.

<b>Data</b>	<b>Processing</b>	<b>Output</b>	<b>Solution Alternative s</b>
given in the problem or provided by the user	List of processing required or procedures.	Output requirement.	List of ideas for the solution of the problem.



# Payroll Problem

- Write a PAC to calculate the salary of an employee who works by hourly basis. The formula to be used is

$$\text{Salary} = \text{Hours worked} * \text{Pay rate}$$

Data	Processing	Output	Solution Alternatives
Hours work, Pay rate	Salary = Hours worked * payrate	Salary	

# Determine whether a given number is prime or not?

Data	Processing	Output	Solution Alternatives
Number, N	Check if there is a factor for N	Print Prime or Not Prime	<ol style="list-style-type: none"><li>1. Divide N by numbers from 2 to N and if for all the division operations, the remainder is non zero, the number is prime otherwise it is not prime</li><li>2. Same as 1 but divide the N from 2 to <math>N/2</math></li><li>3. Same as Logic 1 but divide N from 2 to square root of N</li></ol>

# Problem 1

Write a **Problem Analysis Chart (PAC)** to convert the distance in miles to kilometers, where  $1.609 \text{ kilometers} = 1 \text{ mile}$ .

# Problem 1 - answer

Data	Processing	Output	Solution Alternatives
Distance in miles	$\text{Kilometers} = 1.609 \times \text{miles}$	Distance in kilometers	<ol style="list-style-type: none"><li>1. Define the miles as constants.</li><li>2. Define the miles as input values.</li></ol>

# Problem 2

Write a **Problem Analysis Chart (PAC)** to find an area of a circle where  $\text{area} = \pi * \text{radius} * \text{radius}$

# Problem 2 - answer

Data	Processing	Output
radius	$\text{area} = 3.14 \times \text{radius} \times \text{radius}$	area

# Problem 3

Write a **Problem Analysis Chart (PAC)** to compute and display the temperature inside the earth in Celsius and Fahrenheit. The relevant formulae are

$$\text{Celsius} = 10 \times (\text{depth}) + 20$$

$$\text{Fahrenheit} = 1.8 \times (\text{Celsius}) + 32$$

# Problem 3 - Answer

Write a **Problem Analysis Chart (PAC)** to compute and display the temperature inside the earth in Celsius and Fahrenheit. The relevant formulae are

$$\text{Celsius} = 10 \times (\text{depth}) + 20$$

$$\text{Fahrenheit} = 1.8 \times (\text{Celsius}) + 32$$

Data	Processing	Output
depth	$\text{celsius} = 10 \times (\text{depth}) + 20$ $\text{fahrenheit} = 1.8 \times (\text{celsius}) + 32$	Display celsius, Display fahrenheit



# Problem 4

Given the distance of a trip in miles, miles travelled using one litre of petrol by the car and the current price of one litre of petrol.

Write a PAC to determine the quantity of petrol required for the trip and the cost spent on the petrol.

# Problem 4 - Answer

Input	Processing	Output
Distance in miles, miles per litre, cost per litre	$\text{Petrol needed} = \text{distance} / \text{miles per litre.}$  $\text{estimated cost} = \text{cost per litre} \times \text{petrol needed}$	Display petrol needed  Display estimated cost

# Extended Payroll Problem

- You are required to write a PAC to calculate both the gross pay and the net pay of an employee of your company. Use the following formulae for calculation:
  - Gross pay = number of hours worked \* pay rate
  - Net pay = gross pay – deductions

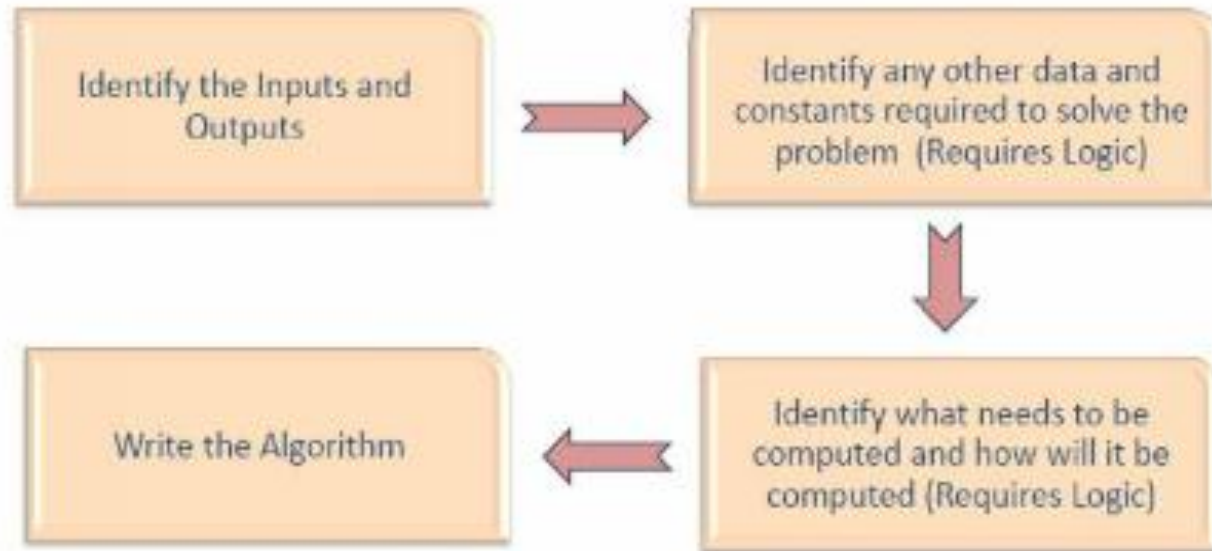
# PAC for Extended Payroll Problem

Input	Processing	Output
Number of hours worked, pay rate, deductions	Gross pay = number of hours * pay rate Net pay = Gross pay – deductions	Net pay

# Algorithm

- Finite set of instructions carried out in a specific order to perform a particular task
- Algorithms are not specific to any programming language
- An algorithm can be implemented in any programming language
- Facilitates easy development of programs

# Steps to Develop an Algorithm

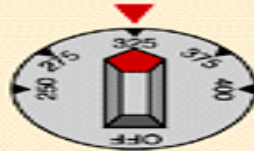


# Properties of an ideal Algorithm

- **Unambiguous** : each step of the algorithm should be clear and unambiguous, and must lead into only one meaning.
- **Input** -An algorithm must have 0 or well defined inputs
- **Output** -An algorithm must have 1 or more well defined outputs
- **Finiteness** -Algorithms must terminate after a finite number of steps
- **Definiteness** -It must give the desired solution.

# ALGORITHM FOR BAKING A CAKE

Heat oven to 325°F



Gather the ingredients



Mix ingredients thoroughly in a bowl



Pour the mixture into a baking pan



Bake in the oven 50 minutes



Repeat

Bake 5 minutes more

Until cake top springs back when touched in the center

Cool on a rack before cutting





# Algorithm for heating a can of soup

**PROBLEM:** Heat up a can of soup

**ALGORITHM:**

Step 1 open can using can opener

- Step 2 pour contents of can into saucepan
- Step 3 place saucepan on burner on stove
- Step 4 turn on correct burner
- Step 5 stir soup until warm

May seem a bit of a silly example but it does show us that the order of the events is important since we cannot pour the contents of the can into the saucepan before we open the can.

# Different patterns in Algorithm

## **Sequential**

Executes the statements in the order in which they appear in the algorithm

## **Selectional (Conditional)**

controls the flow of statements execution based on some condition

## **Iterational (Loop)**

used when a part of the algorithm is to be executed several times

# Sequential Algorithms

- Algorithm for adding two numbers
- Step 1: START

Step 2: Read two numbers A and B

Step 3: Let  $C = A + B$

Step 4: Display C

Step 5: END

# Sequential Algorithms

- Area of a Circle

**Step 1 :** Read the RADIUS of a circle

**Step 2 :** Find the square of RADIUS and store it in SQUARE

**Step 3 :** Multiply SQUARE with 3.14 and store the result in  
AREA

**Step 4:** Print AREA

# Sequential Algorithms

- Find the average marks scored by a student in 3 subjects:

**Step 1 :** Read Marks1, Marks2, Marks3

**Step 2 :** Add Marks1, Marks2 and Marks3 to get sum

**Step 3 :** Divide sum by 3 to get average

**Step 4 :** Display Average

# Selection Algorithms for Conditional Problems

**PROBLEM:** To decide if a fire alarm should be sounded

**ALGORITHM:**

- |                         |                  |
|-------------------------|------------------|
| 1 IF fire is detected   | <b>condition</b> |
| 2 THEN sound fire alarm | <b>action</b>    |

**Another example is:-**

**PROBLEM:** To decide whether or not to go to school

**ALGORITHM:**

- 1 IF it is a weekday AND it is not a holiday
- 2 THEN go to school
- 3 otherwise stay at home

# Pass/ Fail and Average

- Write an algorithm to find the average marks of a student. Also check whether the student has passed or failed. For a student to be declared pass, average marks should not be less than 65.

- 

Step 1 : Read Marks1, Marks2, Marks3

Step 2 : Add Marks1, Marks2 and Marks3 to get sum

Step 3 : Divide sum by 3 to get average

Step 4 : Display Average

•**Step 5:** Set Output as “Student Passed”

**Step 6:** if Average is less than 65 then Set Output as “Student Failed”

**Step 7:** Display Average and Output

# Algorithm for Iterative Problems

This type of loop keeps on carrying out a command or commands UNTIL a given condition is satisfied, the condition is given with the UNTIL command, for example:-

**PROBLEM:** To wash a car

**ALGORITHM:**

1 REPEAT

2 wash with warm soapy water

3 UNTIL the whole car is clean



# Iterational Algorithms – Repetitive Structures

- Find the average marks scored by 'N' number of students

**Step 1** : Read Number Of Students

**Step 2** : set Counter as 1

**Step 3** : Read Marks1, Marks2, Marks3

**Step 4** : **find** Total as sum of Marks1, Marks2 and Marks3

**Step 5** : **find** Average as Total divided by 3

**Step 6** : Set Output as "Student Passed"

**Step 7** : If Average is less than 65 then Set Output as "Student Failed"

**Step 8** : Display Average and Output

**Step 9** : Set Counter as Counter + 1

**Step 10** : If Counter lesser than or equal to numberOfStudents then goto step 3

Step 11: end

# Bigger Problems

- If you are asked to find a solution to a major problem, it can sometimes be very difficult to deal with the complete problem all at the same time.
- For example building a car is a major problem and no-one knows how to make every single part of a car.
- A number of different people are involved in building a car, each responsible for their own bit of the car's manufacture.
- The problem of making the car is thus broken down into smaller manageable tasks.
- Each task can then be further broken down until we are left with a number of step-by-step sets of instructions in a limited number of steps.
- The instructions for each step are exact and precise.

# Top Down Design

- Top Down Design uses the same method to break a programming problem down into manageable steps.
- First of all we break the problem down into smaller steps and then produce a Top Down Design for each step.
- In this way sub-problems are produced which can be refined into manageable steps.

# Top Down Design for Real Life Problem

**PROBLEM:** To repair a puncture on a bike wheel. **ALGORITHM:**

1. remove the tyre
2. repair the puncture
3. replace the tyre

# Step 1: Refinement:

## 1. Remove the tyre

1.1 turn bike upside down

1.2 lever off one side of the tyre

1.3 remove the tube from inside the tyre

## Step 2: Refinement:

### 2. Repair the puncture Refinement:

2.1 find the position of the hole in the tube

2.2 clean the area around the hole

2.3 apply glue and patch

# Step 3: Refinement:

## 3. Replace the tyre Refinement:

3.1 push tube back inside tyre

3.2 replace tyre back onto wheel

3.3 blow up tyre

3.4 turn bike correct way up



## Still more Refinement:

Sometimes refinements may be required to some of the sub-problems, for example if we cannot find the hole in the tube, the following refinement can be made to 2.1:-

# Still more Refinement:

## Step 2.1: Refinement

2.1 Find the position of the hole in the tube

2.1.1 WHILE hole cannot be found

2.1.2 Dip tube in water

2.1.3 END WHILE

# Problem Solving Methods

## Representation of Algorithm

```
graph TD; A[Representation of Algorithm] --> B[Flowchart  
(Diagrammatical or Visual Representation)]; A --> C[Pseudocode]
```

Flowchart

(Diagrammatical or  
Visual Representation)

Pseudocode

Generally, flowcharts work well for small problems but pseudocode is used for larger problems.

# Drawing Flowcharts

- Flowchart is the graphic representations of the individual steps or actions to implement a particular module
- Flowchart can be likened to the blueprint of a building
- An architect draws a blueprint before beginning construction on a building, so the programmer draws a flowchart before writing a program
- Flowchart is independent of any programming language.

# Flow Charts

A flow chart is an organized combination of shapes, lines and text that graphically illustrate a process or structure.

## Symbols used



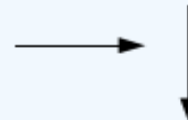
**Start/Stop**



**Process**



**Input/Output (Data)**



**Flow Lines**

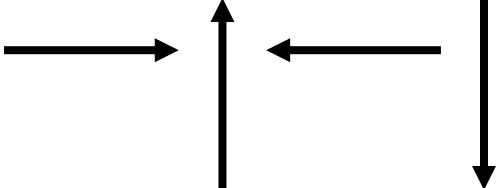
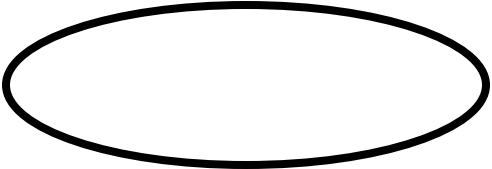



**Decision symbol**

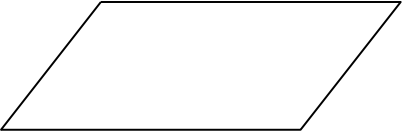
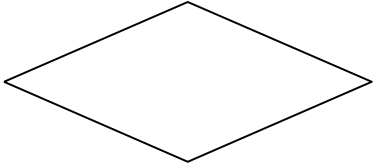
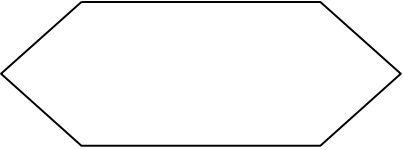
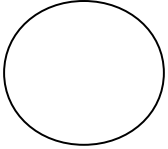
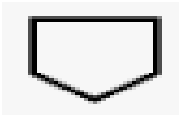


**Connector**

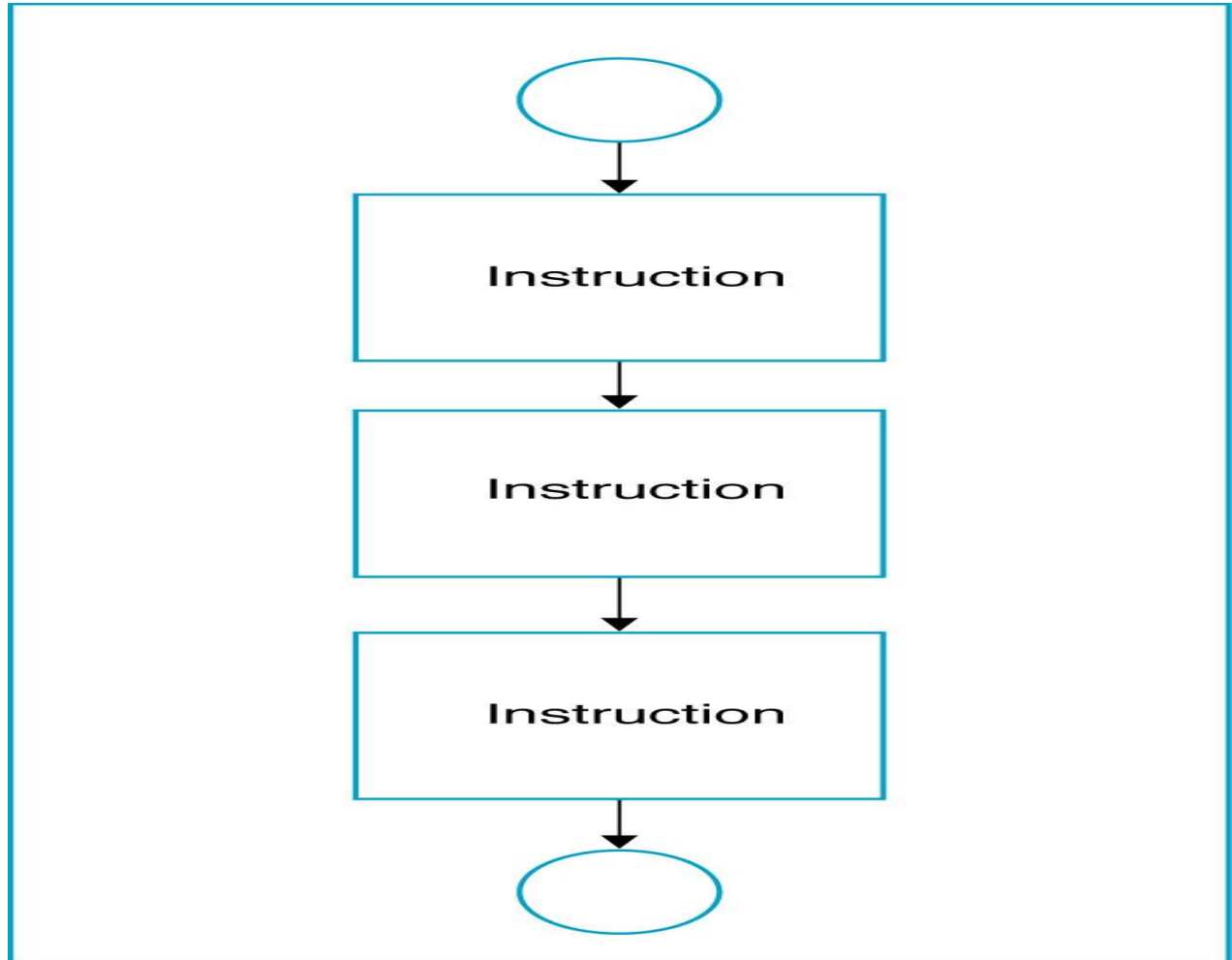
# Symbols in flowchart

Symbol	Function
	Show the direction of data flow or logical solution.
	Indicate the beginning and ending of a set of actions or instructions (logical flow) of a module or program.
	Indicate a process, such as calculations, opening and closing files.

# Symbols in flowchart

	Indicate input to the program and output from the program.
	Use for making decision. Either True or False based on certain condition.
	Use for doing a repetition or looping of certain steps.
	Connection of flowchart on the same page.
	Connection of flowchart from page to page.

# Sequential Logic Structure





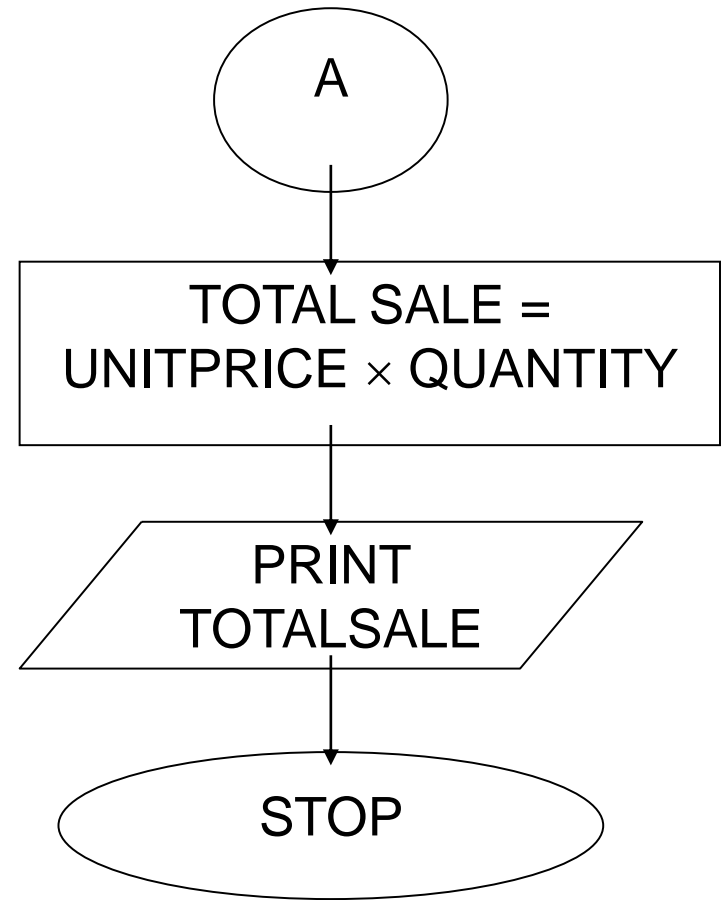
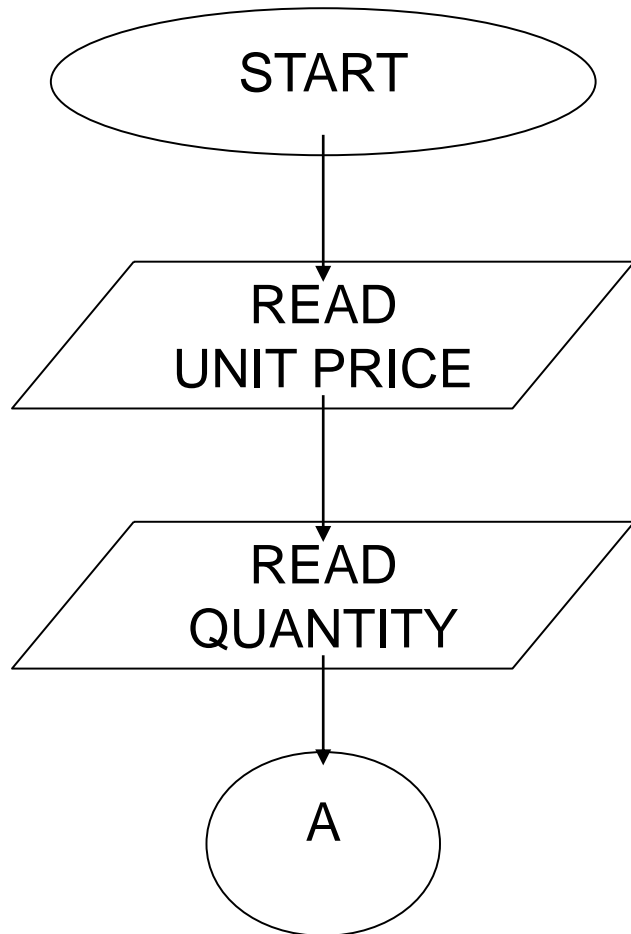
# Sale Problem

Given the unit price of a product and the quantity of the product sold, draw a flowchart to calculate and print the total sale.

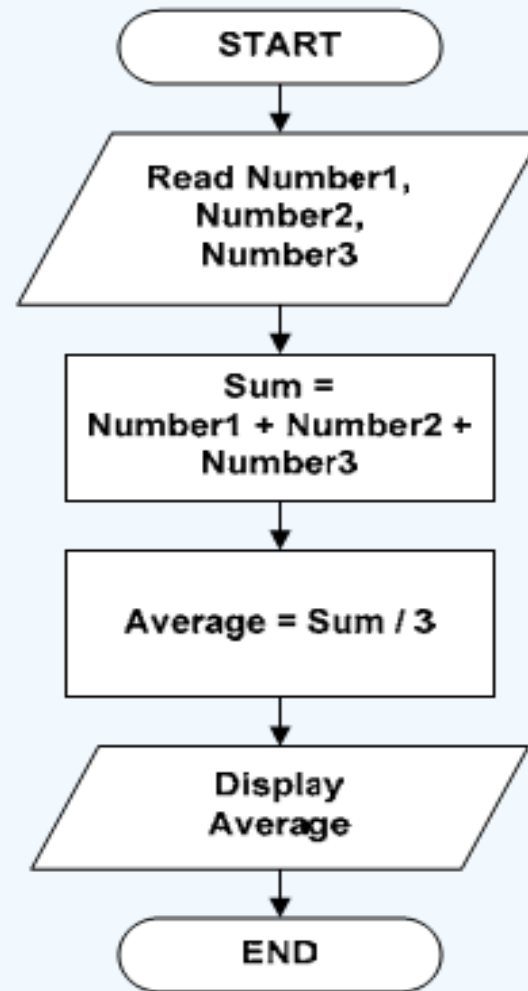
## **Solution:** Stepwise Analysis of the Sale Problem

- Read the unit price and the quantity
- Calculate total sale = unit price \* quantity
- Print total sale

# Flowchart for sale problem



# Find the average of three numbers

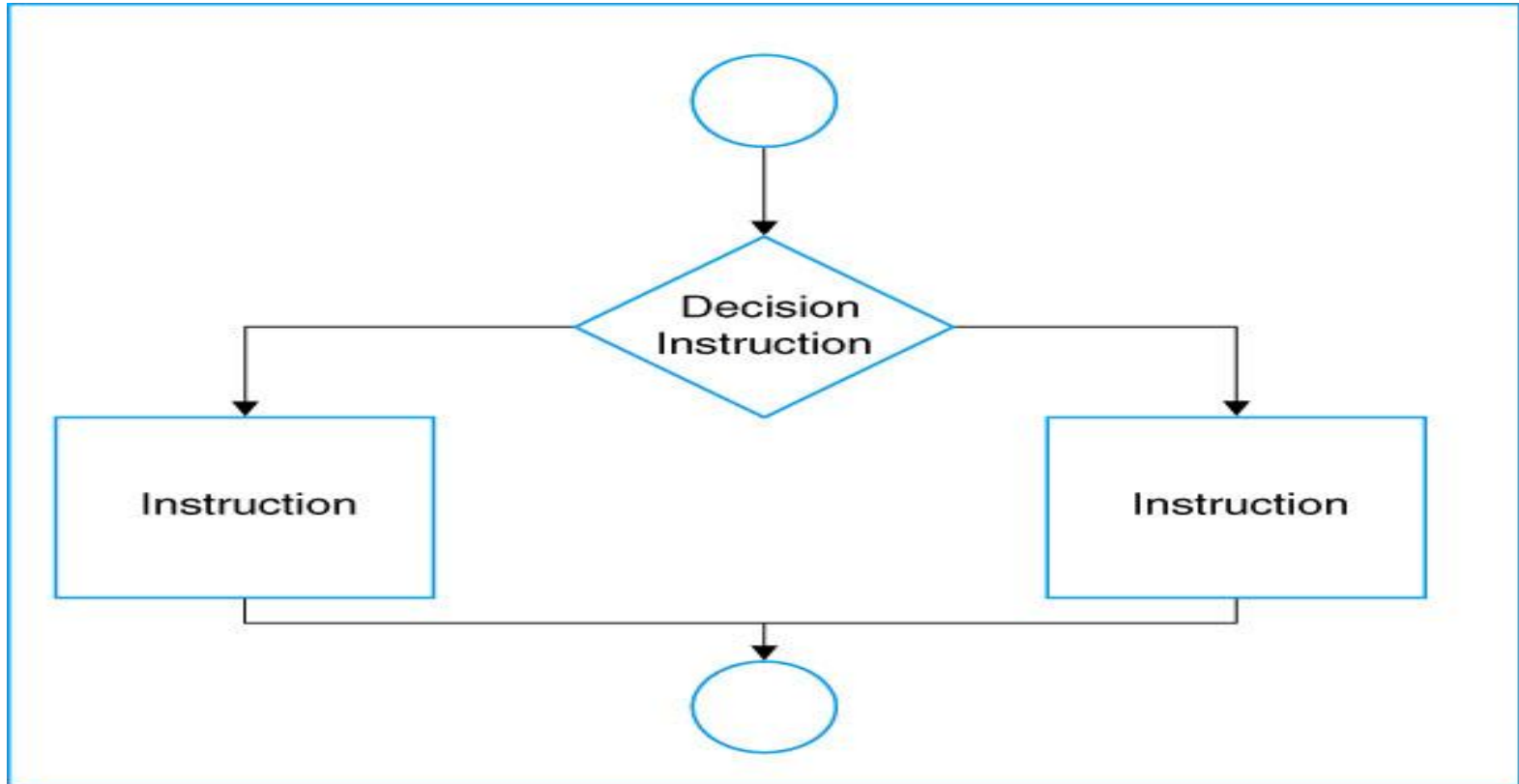


# The Decision Logic Structure

- Implements using the IF/THEN/ELSE instruction.
- Tells the computer that IF a condition is true, THEN execute a set of instructions, or ELSE execute another set of instructions
- ELSE part is optional, as there is not always a set of instructions if the conditions are false.
- Algorithm:

```
IF <condition(s)> THEN  
    <TRUE instruction(s)>  
ELSE  
    <FALSE instruction(s)>
```

# Decision Logic Structure



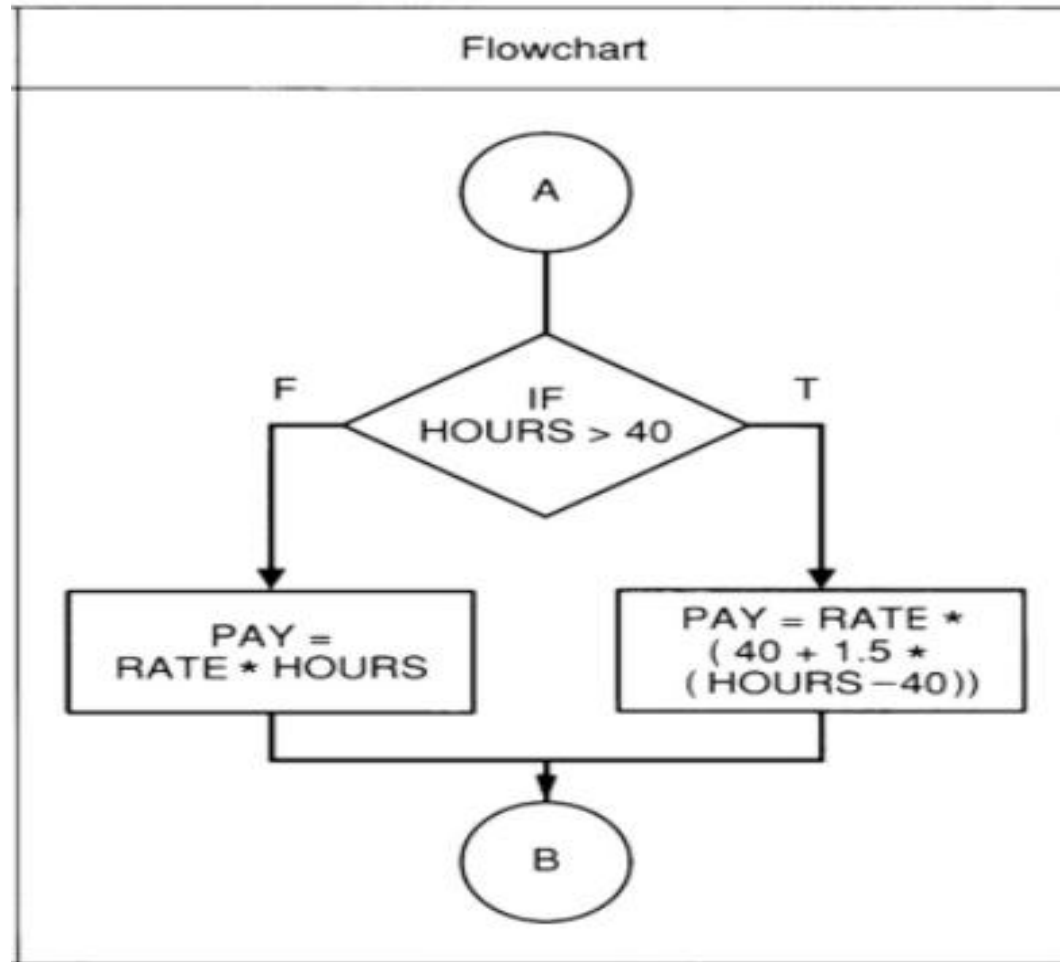
# Examples of conditional expressions

- $A < B$  (A and B are the same data type  
– either numeric, character, or string)
- $X + 5 \geq Z$  (X and Z are numeric data)
- $E < 5$  or  $F > 10$  (E and F are numeric data)

# Conditional Pay Calculation

- Assume you are calculating pay at an hourly rate, and overtime pay(over 40 hours) at 1.5 times the hourly rate.
  - IF the hours are greater than 40, THEN the pay is calculated for overtime, or ELSE the pay is calculated in the usual way.

# Example Decision Structure

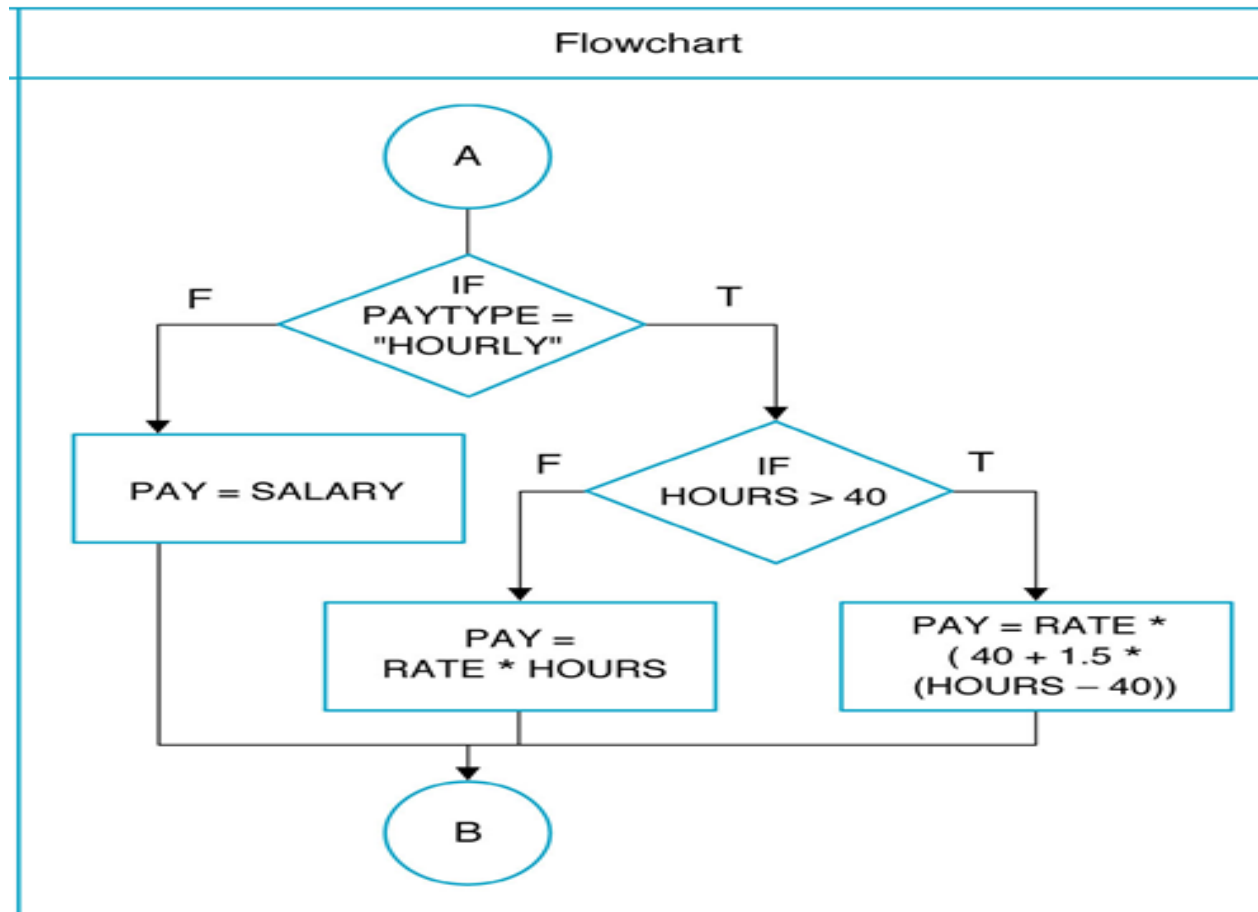




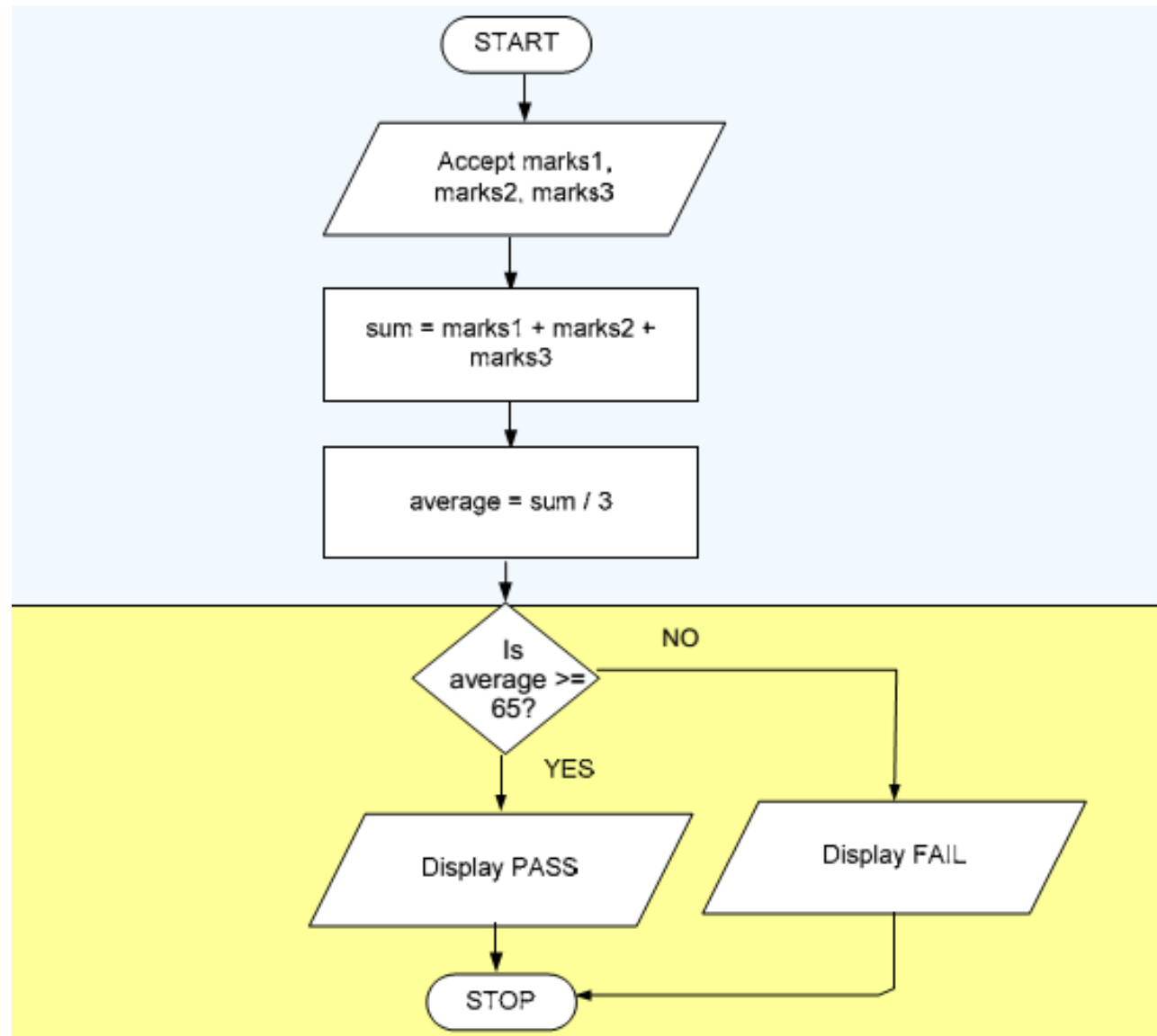
# NESTED IF/THEN/ELSE INSTRUCTIONS

- Multiple decisions.
- Instructions are sets of instruction in which each level of a decision is embedded in a level before it.

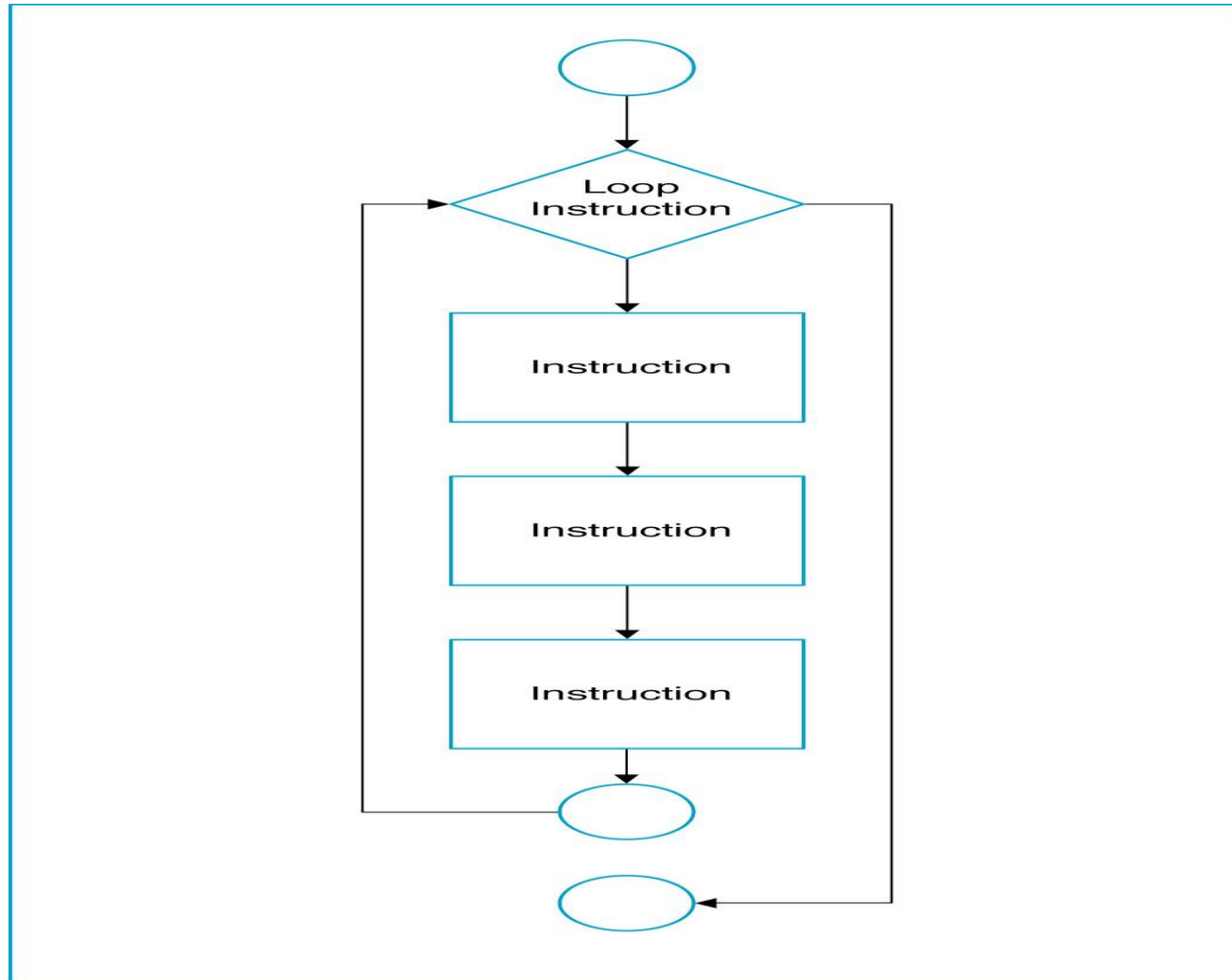
# NESTED IF/THEN/ELSE INSTRUCTIONS



# Flow Chart - Selectional



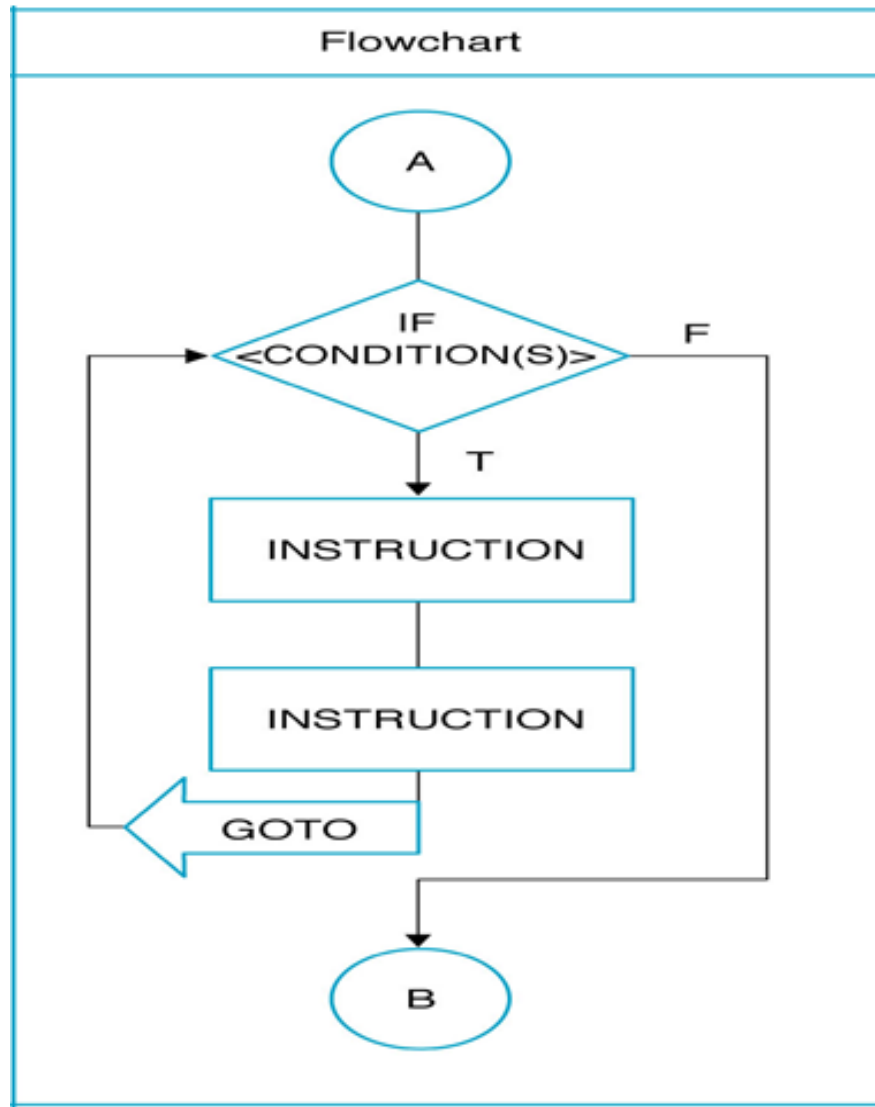
# Loop Logic Structure



# Iterational Structure

- Repeat structure
- To solve the problem that does the same task over and over for different sets of data
- Types of loop:
  - WHILE loop
  - Do..WHILE loop
  - Automatic-Counter Loop

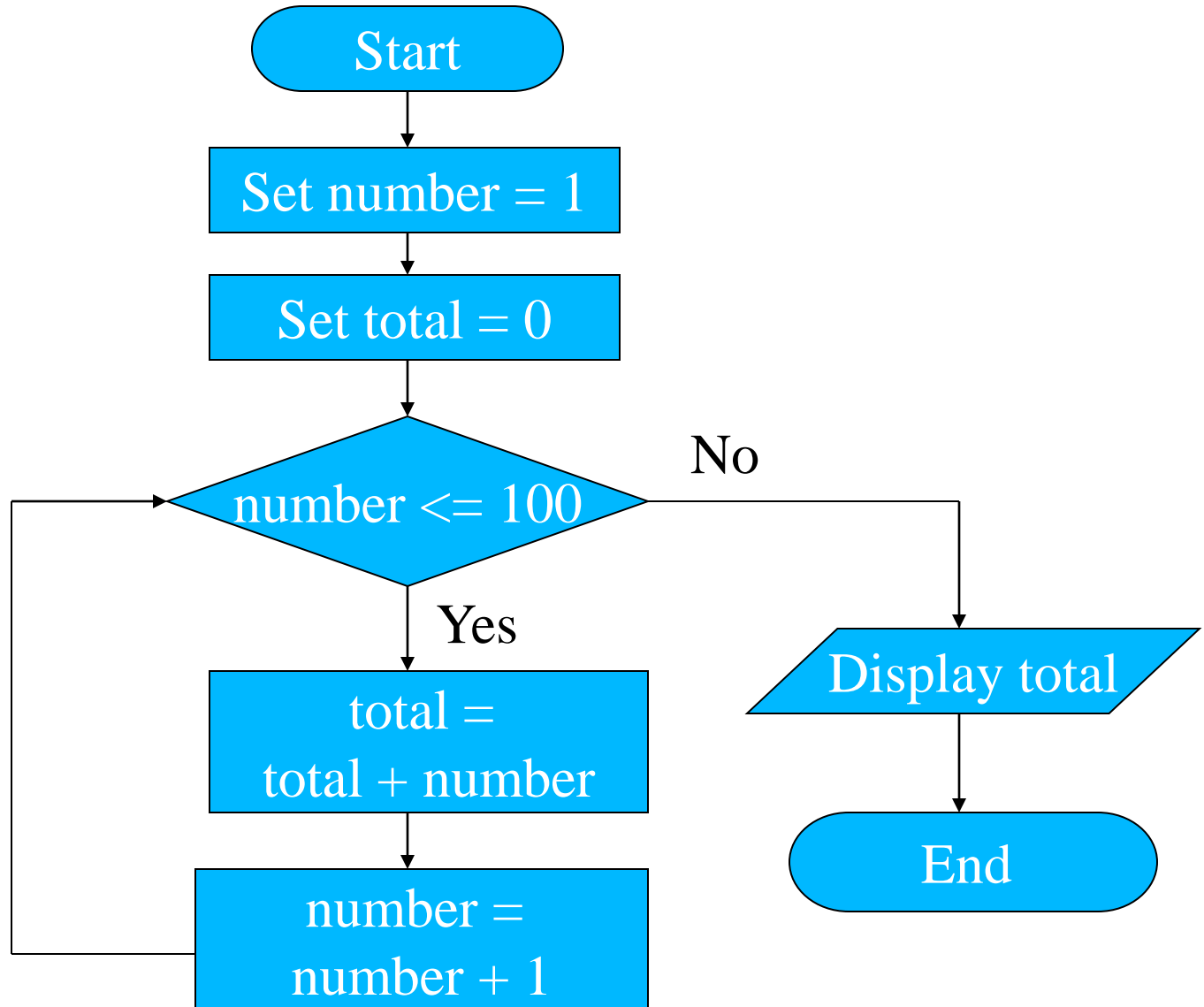
# WHILE loop



# WHILE loop

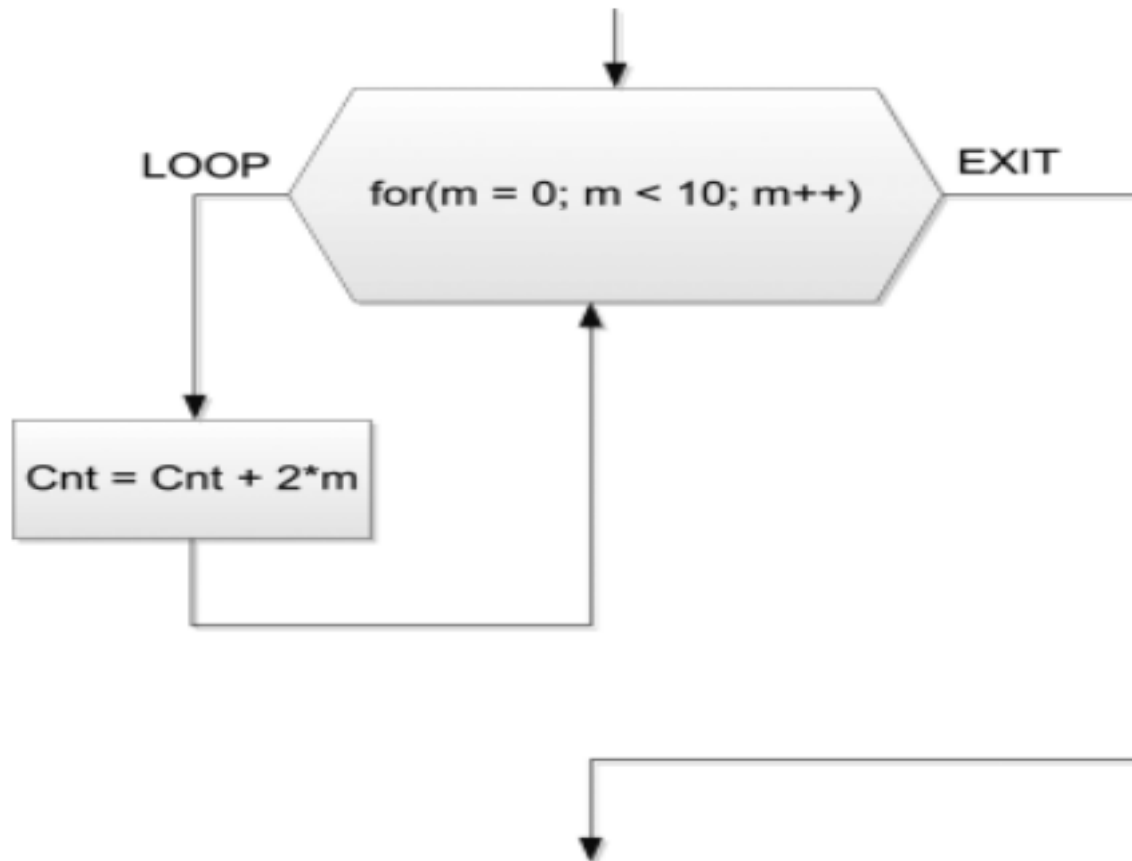
- Do the loop body if the condition is true.
- Example: Get the sum of 1, 2, 3, ..., 100.

# WHILE loop





# For loop



# Automatic Counter Loop

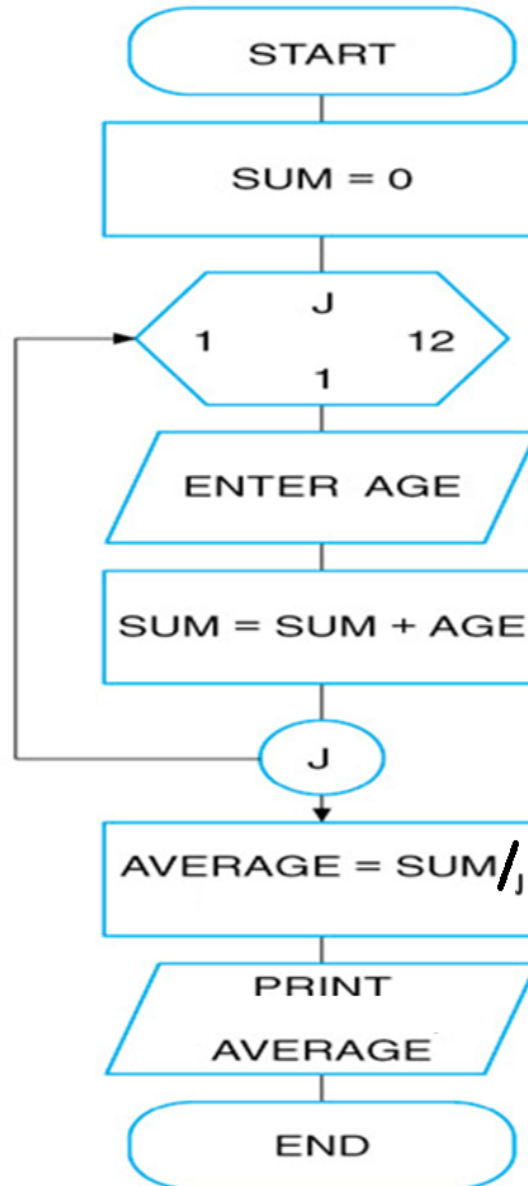
- Use variable as a counter that starts counting at a specified number and increments the variable each time the loop is processed.
- The beginning value, the ending value and the increment value may be constant.
- They should not be changed during the processing of the instruction in the loop.

# Automatic-Counter Loop



# Automatic-Counter Loop

Find average age of 12 people



# NESTED LOOP

Output the following

1,1

1,2

1,3

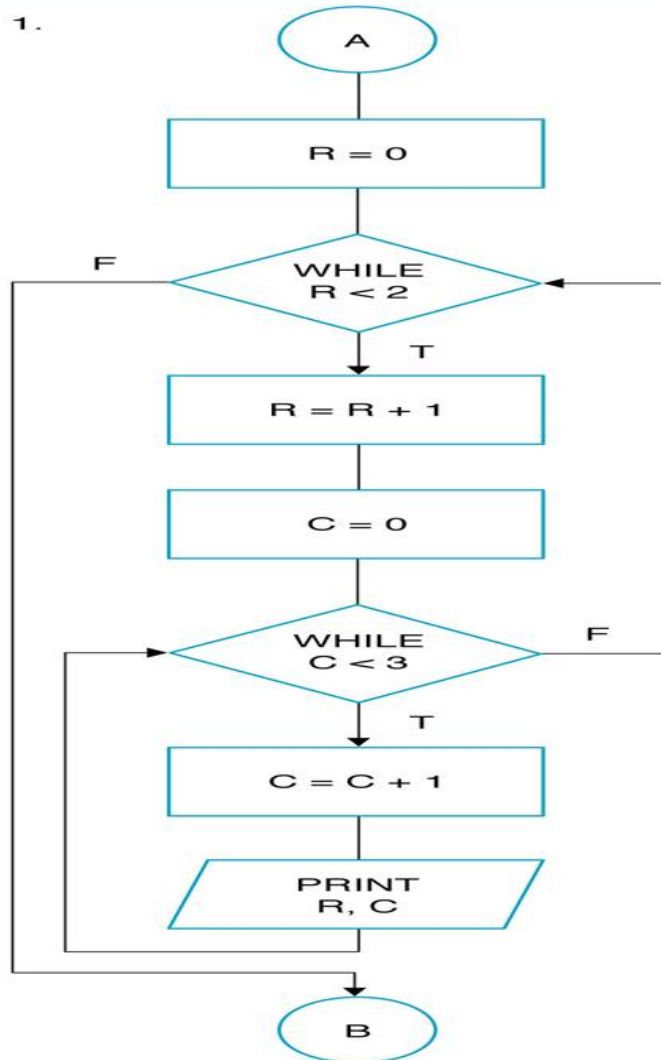
2,1

2,2

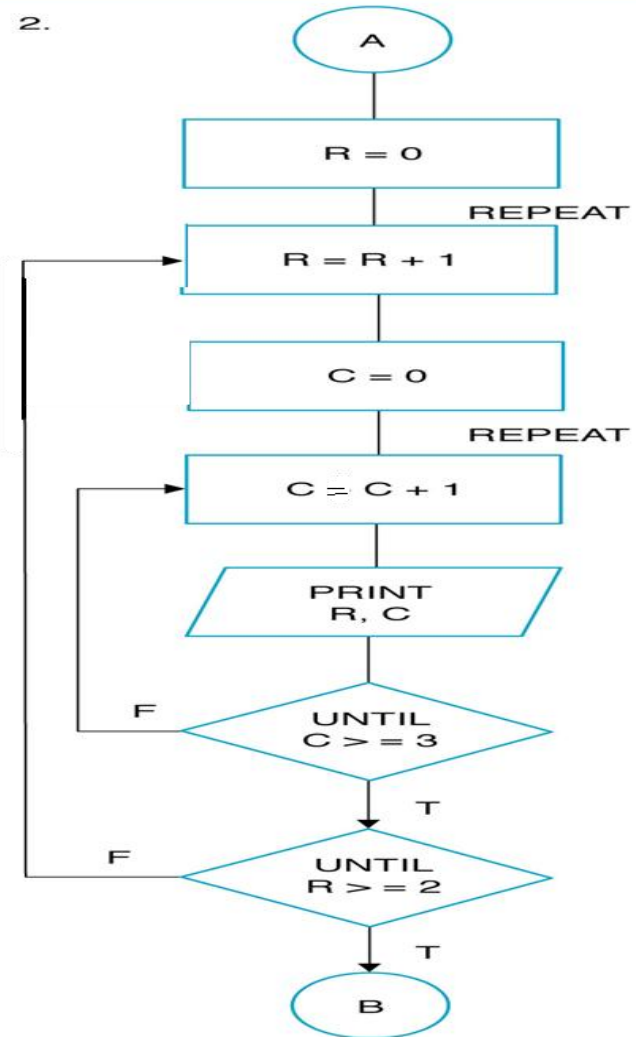
2,3

# NESTED LOOP

1.

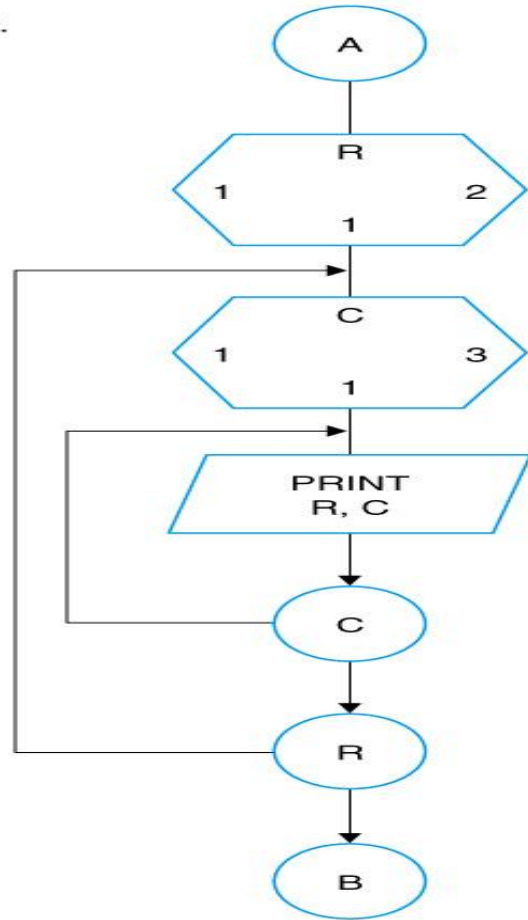


2.

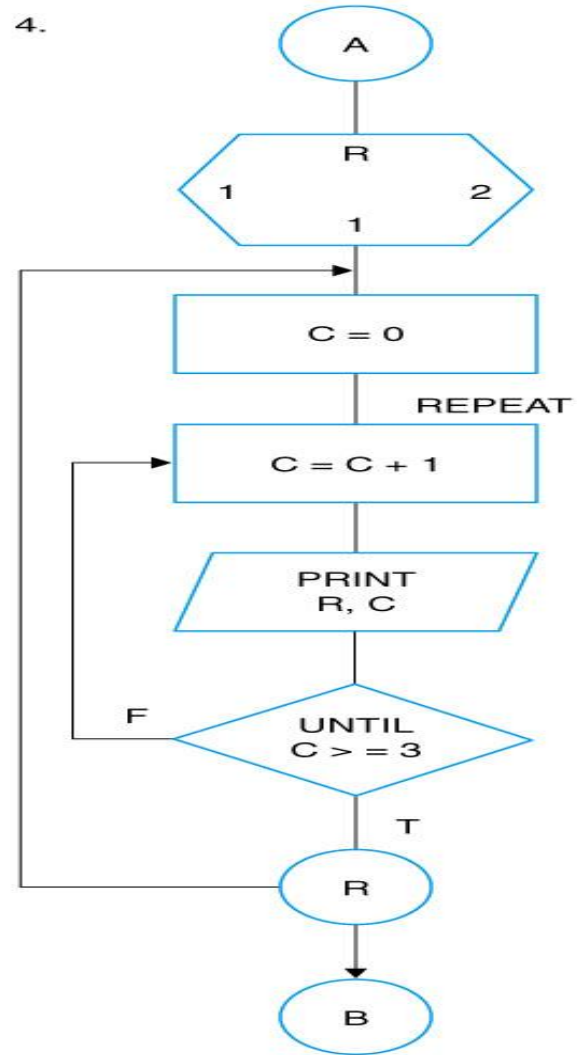


# NESTED LOOP

3.



4.

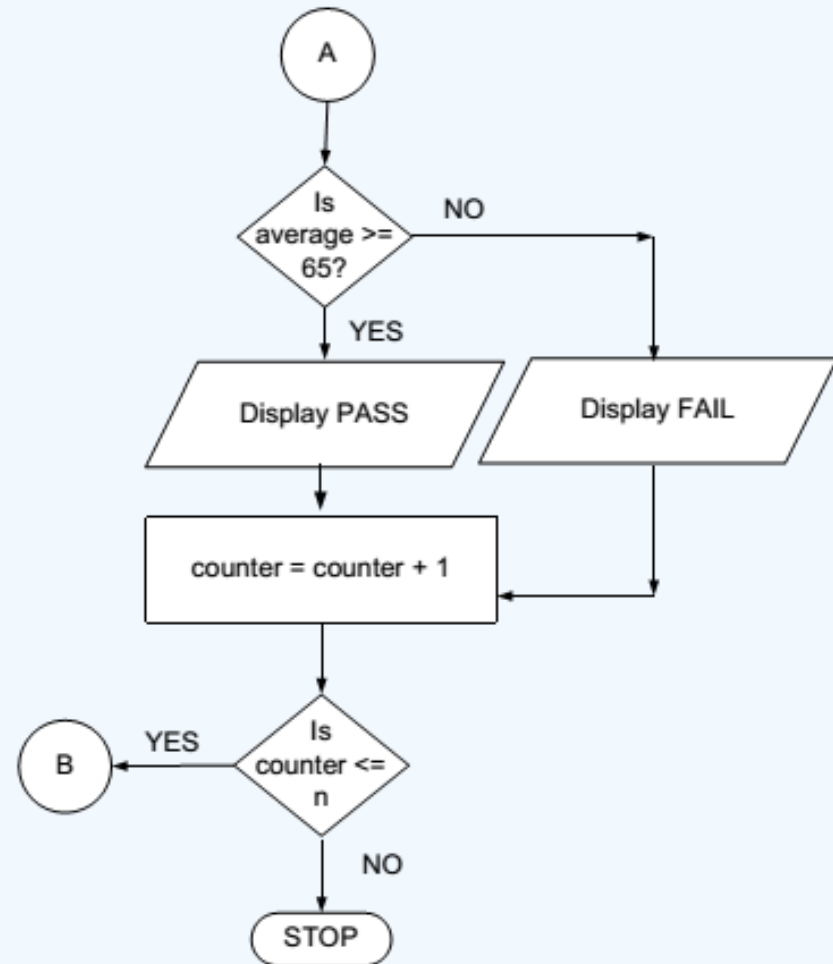
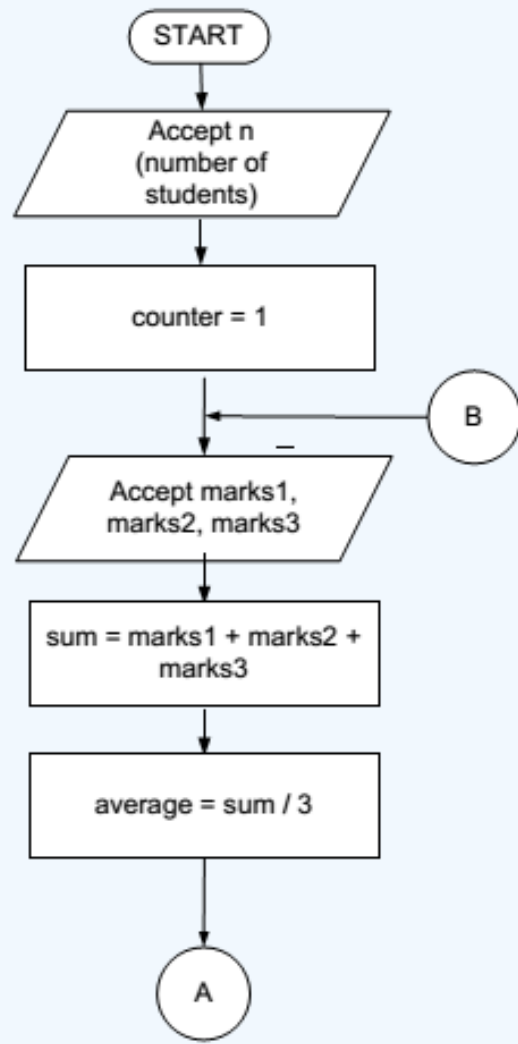


# Example (Iterational)

- Write a program to find the average of marks scored by him in three subjects for 'N' students. And then test whether he passed or failed. For a student to pass, average should not be less than 65.



# Flow Chart Iterational



# Refer for other flowchart symbols

- <https://www.gliffy.com/blog/guide-to-flowchart-symbols>

# Pseudocode

- Pseudocode is a way of representing algorithms or code in a simplified, human-readable format. It is more structured than plain text but does not adhere to the syntax rules of programming languages.
- No defined language, no real language (pseudo)
- For non-programmers, actual programs are difficult to read and understand, but pseudocode helps them to review the steps to confirm that the proposed implementation is going to achieve the desired output.
- Ensure no important step is missed out.

# Keywords in Pseudocode

- Several keywords are often used to **indicate common** input, output, and **processing operations**.
- **Input:** READ, OBTAIN, GET
- **Output:** PRINT, DISPLAY, SHOW
- **Compute:** COMPUTE, CALCULATE, DETERMINE
- **Initialize:** SET, INIT
- **Add one:** INCREMENT, BUMP
- **Minus One :** DECREMENT

# Mathematical Keywords often in Pseudocode

**Assignment:**  $\leftarrow$  or  $:=$

*Example:*  $c \leftarrow 2\pi r$ ,  $c := 2\pi r$

**Comparison:**  $=$ ,  $\neq$ ,  $<$ ,  $>$ ,  $\leq$ ,  $\geq$

**Arithmetic:**  $+$ ,  $-$ ,  $\times$ ,  $/$ ,  $\text{mod}$

**Floor/ceiling:**  $\lfloor$ ,  $\rfloor$ ,  $\lceil$ ,  $\rceil$

$a \leftarrow \lfloor b \rfloor + \lceil c \rceil$

**Logical:**  $\text{and}$ ,  $\text{or}$

**Sums, products:**  $\Sigma$   $\Pi$

# Five important rules for writing pseudocode

- Write **one** statement per line.
- Initial keywords should be represented in **capital case**(READ, WRITE, IF, WHILE, UNTIL).
- Indentation of pseudocode should be similar to the actual program to show hierarchy.
- Ending the multiline structure is necessary.
- Keep statements in simple language(English).

# Pseudocode for Sequential Problem

**Find area of rectangle**

# Pseudocode for Sequential Problem

## **Find area of rectangle**

- READ height of rectangle
- READ width of rectangle
- COMPUTE area as height times width



# IF THEN ELSE

- Binary choice on a given **Boolean condition** is indicated by the use of **four keywords**:
- **IF, THEN, ELSE, and ENDIF.**

The general form is:

IF condition THEN

    sequence 1

ELSE

    sequence 2

ENDIF

# Example

```
IF HoursWorked > NormalMax THEN  
    Display overtime message  
ELSE  
    Display regular time message  
ENDIF
```

# Problem

**Write Pseudocode for Sum of 1 to 100**

```
READ n;  
SET sum := 0;  
WHILE i := 1 to n  
    SET sum := sum + i;  
    i:=i+1;  
ENDWHILE  
PRINT sum;
```

# WHILE

- used to specify a loop with a test at the top.
- beginning and ending of the loop are indicated by **two keywords: WHILE and END WHILE.**

- **General form is:**

```
WHILE condition
    sequence
ENDWHILE
```

# WHILE

- Loop is entered only if the condition is true.
- "sequence" is performed for each iteration.
- At the conclusion of each iteration, the condition is evaluated and the loop continues as long as the condition is true.

# Example

**WHILE** Population < Limit

    Compute Population as Population + Births - Deaths

**ENDWHILE**

**WHILE** employeeType NOT EQUAL manager

    Compute employeeBonus = employeeSalary\*20/100

**ENDWHILE**

# FOR Loop

- This loop is a specialized construct for iterating a specific number of times, often called a "counting" loop.
- **Two keywords:** **FOR** and **ENDFOR** are used.
- **The general form is:**

FOR iteration bounds

sequence

ENDFOR



# Example FOR Loop

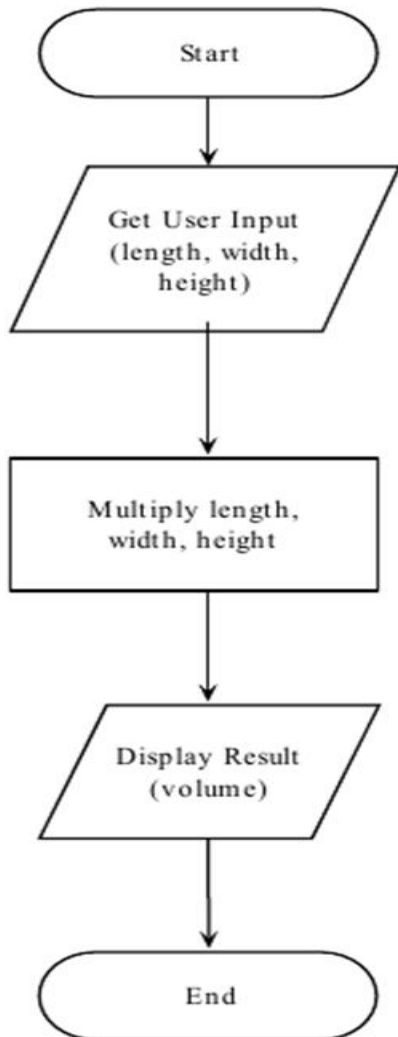
- FOR each month of the year
- FOR each employee in the list

# Repeat until

Pseudocode that will input 1000 numbers

```
Count  $\leftarrow$  0  
Repeat  
    Input Num  
    Count  $\leftarrow$  Count + 1  
Until (Count  $\leq$  1000)
```

# Pseudocode



**Get** length, width, height  
**Compute** volume  
     $\text{volume} := \text{length} * \text{width} * \text{height}$   
    **Store** volume  
**Display** volume

# INVOKING FUNCTIONS

- Use the **CALL** keyword.

For example:

1. CALL SquareRoot with orbitHeight

RETURNING nominalOrbit

SquareRoot is the function name

orbitHeight is the input to the function

nominalOrbit is the output of the function

2. CALL AvgAge with StudentAges
3. CALL Swap with CurrentItem and TargetItem
4. CALL Account.debit with CheckAmount
5. CALL getBalance RETURNING aBalance

# Pseudocode for Finding Grade – Try it

- Enter your Test Score and Your Grade will be Displayed

- >90 A grade
- >80 B grade
- >70 C grade
- >60 D grade
- <60 Fail

# Answer - Pseudocode for Finding Grade

```
READ Test Score
IF TestScore>90
    DISPLAY "Your Grade is an A"
Else IF TestScore>80
    DISPLAY "Your Grade is an B"
Else IF TestScore>70
    DISPLAY "Your Grade is an C"
Else IF TestScore>60
    DISPLAY "Your Grade is an D"
Else
    DISPLAY "Your Grade is an F"
ENDIF
```

**PROGRAMMING PHASE**

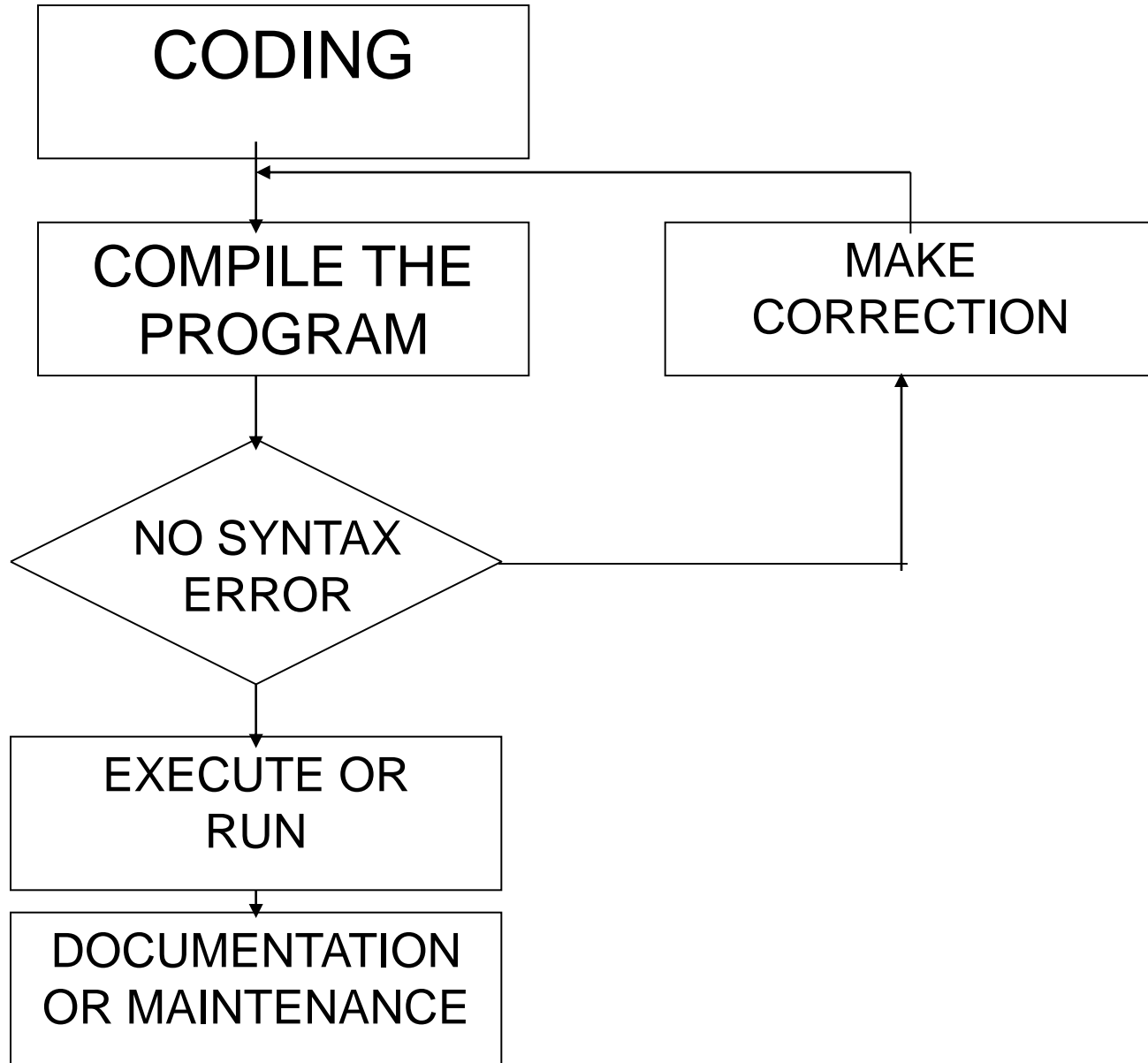
# Programming Or Implementation Phase

- Transcribing the logical flow of solution steps in flowchart or algorithm to program code and run the program code on a computer using a programming language.
- Programming phase takes 5 stages:
  - Coding.
  - Compiling.
  - Debugging.
  - Run or Testing.
  - Documentation and maintenance.



# Programming Or Implementation Phase

- Once the program is coded using one of the programming language, it will be compiled to ensure there is no syntax error.
- Syntax free program will then be executed to produce output and subsequently maintained and documented for later reference.



# Coding

- **Translation or conversion** of each operation in the **flowchart or algorithm (pseudocode) into a computer-understandable language.**
- Coding should follow the format of the chosen programming language.

# Compiling and Debugging

- **Compiling** - Translates a program written in a particular high-level programming language into a form that the computer can understand
- Compiler checks the program code so that any part of source code that does not follow the format or any other language requirements will be flagged as syntax error.
- This **syntax error** is also called **bug**, when error is found the programmer will debug or correct the error and then recompile the source code again
- **Debugging** process is continued until there is no more error in program

# Testing

- The program code that contains no more error is called executable program. It is ready to be tested.
- When it is tested, the data is given and the result is verified so that it should produced output as intended.
- Though the program is error free, sometimes it does not produced the right result. In this case the program faces logic error.
- Incorrect sequence of instruction is an example that causes logic error.

# Documentation and Maintenance

- When the program is thoroughly tested for a substantial period of time and it is consistently producing the right output, it can be documented.
- Documentation is important for future reference. Other programmer may take over the operation of the program and the best way to understand a program is by studying the documentation.
- Trying to understand the logic of the program by looking at the source code is not a good approach.
- Studying the documentation is necessary when the program is subjected to enhancement or modification.
- Documentation is also necessary for management use as well as audit purposes.

# Try it

- Little Bob loves chocolate, and he goes to a store with Rs.  $N$  in his pocket. The price of each chocolate is Rs.  $C$ . The store offers a discount: for every  $M$  wrappers he gives to the store, he gets one chocolate for free. This offer is available only once. How many chocolates does Bob get to eat?
- Prepare PAC, algorithm and pseudocode for the above problem.