CSE 1xx

Problem Solving and Programming

Course Objective

- **Course Objective**
- **Introduce** the essential skills for a logical thinking to problem solving
- Introduce the essential skills in programming for problem solving using computer
- **Outcomes**
- On completion of the course, students will have the
- -ability to identify an appropriate approach to solve the problem
- -ability to write a pseudo code for the identified strategy
- -ability to translate the pseudocode into an executable
- program
- -ability to validate the program for all the possible inputs.

Skills of Software Engineer

- **Technical Skills**
- **-Software Design**
- **- Coding**
- **-Testing**
- **Problem Solving Skills**
- **-logical and analytical thinking**
- Soft Skills
- -Communication
- -Team Work

Problem

- **Definition:**A problem is a puzzle that requires logical thought or mathematics to solve.
  A puzzle could be a set of questions on a scenario which consists of ***description of reality*** **and set of constraints about the scenario.**

  Scenario: VIT campus has a library. The librarian issues books only to VIT employees.
  Careful observation suggests...
  Description of reality : There is a library in VIT campus and
  there is a librarian in the library.
  Constraint : Librarian issues books only to VIT employees
  Questions about the scenario:
  1 How many books are there in the library?
  2 How many books can be issued to an employee?
  3. Does the librarian issue a book to himself?
  etc

Case study - Discussion

Consider a bigger scenario...

A Retail Shop

Demand Planning

Inventory Control

Store layout

Item Management

Billing

Pricing

Promotions

Have you ever observed this scenario?
Yes!!! What are the problems in the scenario?

Types of Problems

- Problems do not always have straightforward solutions. Some problems, such as balancing a checkbook or baking a cake, can be solved with a series of actions. These solutions are called **algorithmic solutions**.
- Once the alternatives have been eliminated, for example, and once one has chosen the best among several methods of balancing the checkbook, the solution can be reached by completing the actions in steps. These steps are called the **algorithm**.
- The solutions of other problems, such as how to buy the best stock or whether to expand the company, are not so straightforward. These solutions require reasoning built on knowledge and experience, and a process of trial and error. Solutions that cannot be reached through a direct set of steps are called **heuristic solution**

Analytical Skill

- *Thinking analytically is a skill like carpentry or driving a car. It can be taught, it can be learned, and it can improve with practice. But like many other skills, such as riding a bike, it is not learned by sitting in a classroom and being told how to do it. Analysts learn by doing.*

Problem Solving with Computers

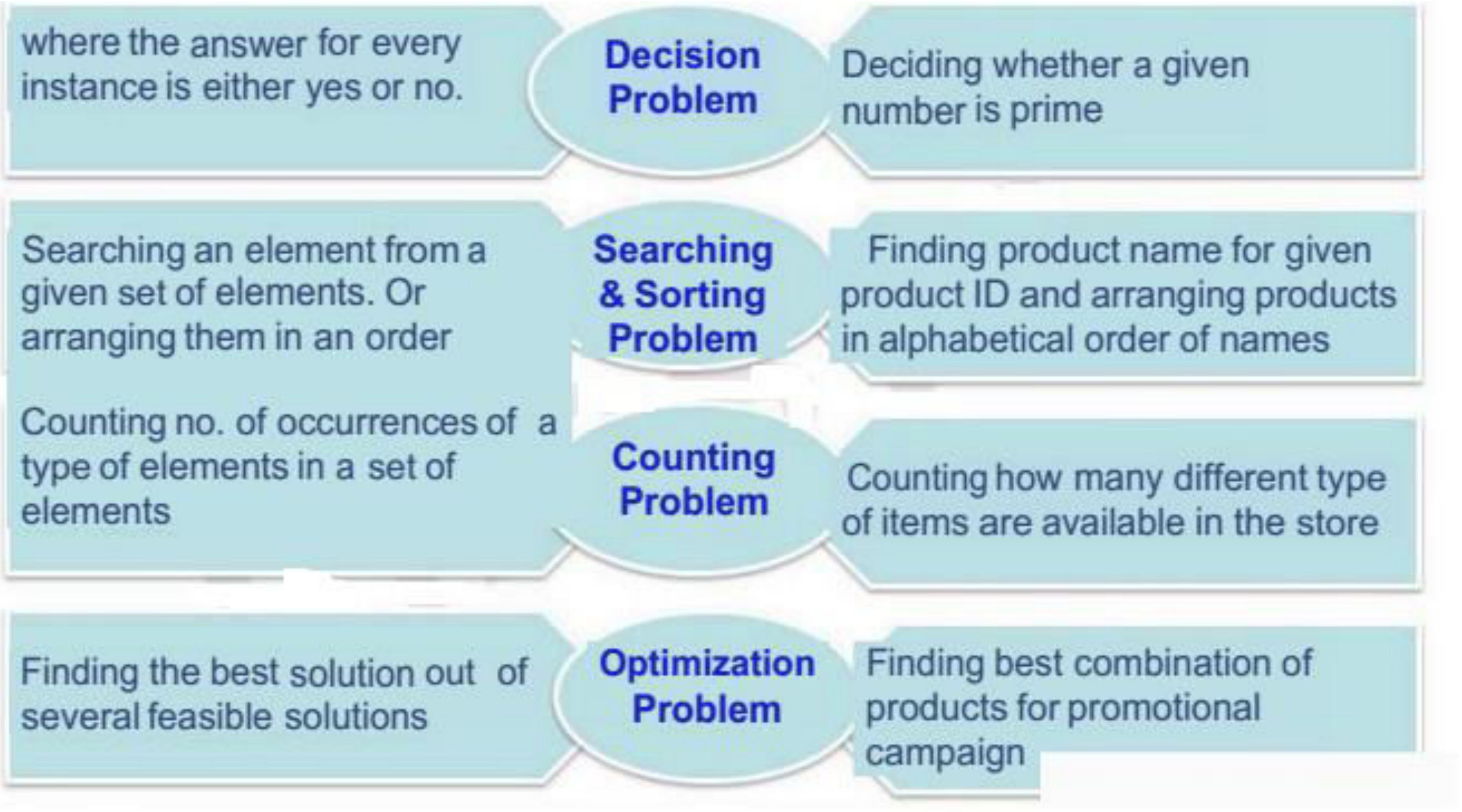
- Computers are built to deal with algorithmic solutions, which are often difficult or very time consuming for humans.
- People are better than computers at developing heuristic solutions.
- Solving a complicated calculus problem or alphabetizing 10,000 names is an easy task for the computer, but the problem of how to throw a ball or how to speak Tamil is not. The difficulty lies in the programming.
- How can problems such as how to throw a ball or speak English be solved in a set of steps that the computer can understand?

- The field of computers that deals with heuristic types of problems is called artificial intelligence. Artificial intelligence enables a computer to do things like build its own knowledge bank and speak in a human language.
- As a result, the computer’s problem-solving abilities are similar to those of a human being. Artificial intelligence is an expanding computer field, especially with the increased use of Robotics.
- Until computers can be built to think like humans, people will process most heuristic solutions and computers will process many algorithmic solutions

Computational Problems

- **Computation Definition:** Computation is the process of evolution from one **state to** another in accordance with some **rules.**

Examples of Computational Problem



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

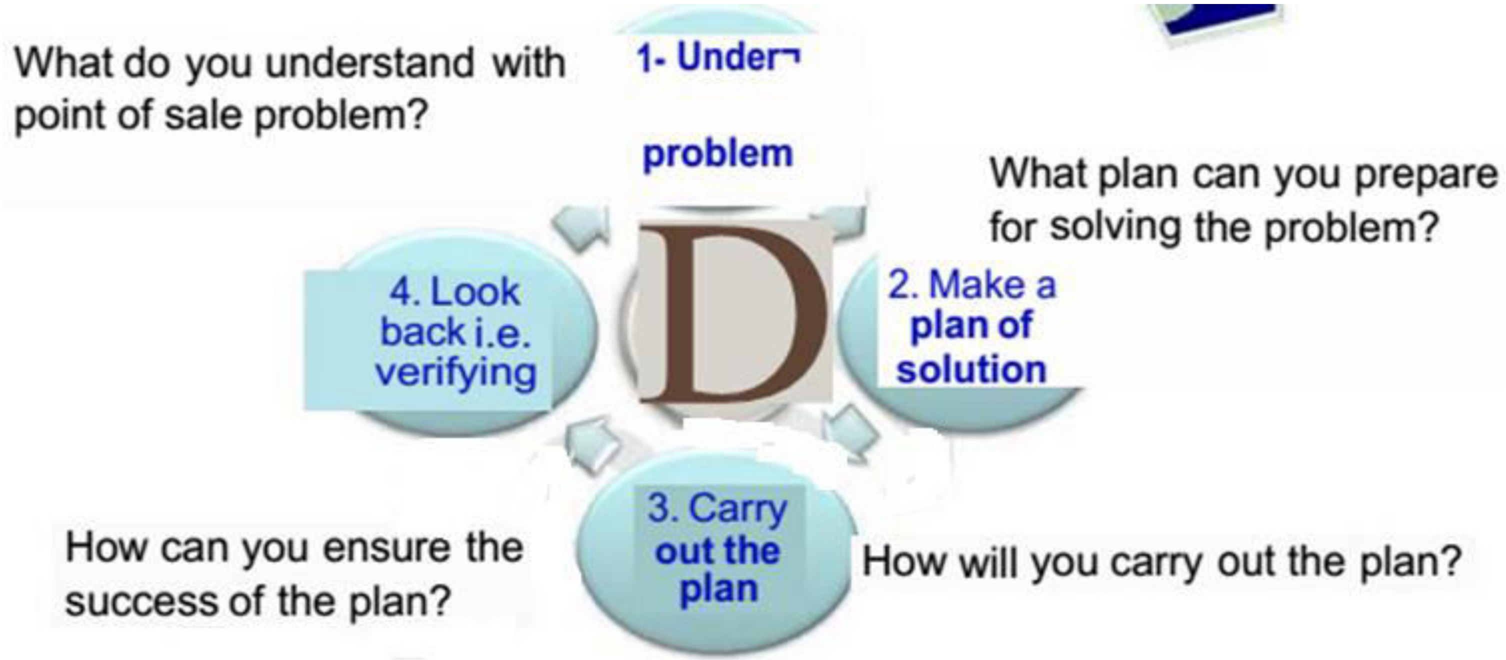
Optimization Problem

Finding best combination of products for promotional campaign

The Process of Computational Problem Solving

- Computational problem solving does not simply involve the act of computer programming. It is a *process*, with programming being only one of the steps.
- **Before a program is written, a design for the program must be developed**. And before a design can be developed, the problem to be solved must be well understood. Once written, the program must be thoroughly tested

Problem Solving Life Cycle



What do you understand with point of sale problem?

1- Under¬

problem

What plan can you prepare for solving the problem?

2. Make a plan of solution

How will you carry out the plan?

3. Carry out the plan

How can you ensure the success of the plan?

4. Look back i.e. verifying

What Problem Can Be Solved By Computer

- When the solution can be produced by a set of step-by-step procedures or actions.
- This step-by-step action is called an *algorithm*.
- The algorithm will process some inputs and produced output.
- Solving problem by computer undergo two phases:
  - Phase 1:
    - Organizing the problem or pre-programming phase.
  - Phase 2:
    - Programming phase.

PRE-PROGRAMMING PHASE

- This phase requires five steps:
  - Analyzing the problem.
  - Developing the Hierarchy Input Process Output (HIPO) chart or Interactivity Chart (IC).
  - Developing the Input-Process-Output (IPO) Chart.
  - Drawing the Program flowcharts.
  - Writing the algorithms.

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.

PRE-PROGRAMMING PHASE

- All These requirements can be presented in a Problem Analysis Chart (PAC)

Data	Processing	Output	Solution Alternatives
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.

PRE-PROGRAMING PHASE

- Example: **Payroll Problem**
  - Calculate the salary of an employee who works by hourly basis. The formula to be used is

Salary = Hour works * Pay rate

Data	Processing	Output	Solution Alternatives
Hours work, Pay rate	Salary = Hours work * payrate	Salary	1. Define the hours worked and pay rate as constants. ∗2. Define the hours worked and pay rate as input values.

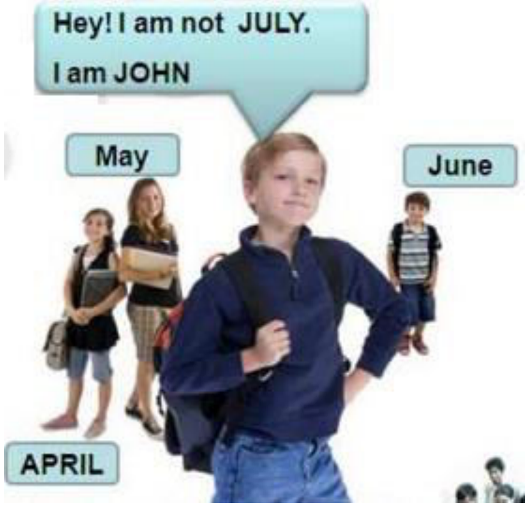
Problem 1

Write a Problem Analysis Chart (PAC) to convert the distance in miles to kilometers where 1.609 kilometers per mile.

Data	Processing	Output	Solution Alternatives
Distance in miles	Kilometers = 1.609 x miles	Distance in kilometers	1. Define the miles as constants. ∗2. Define the miles as input values.

Logic

- **Definition : A method of human thought that involves thinking in a linear, step by step manner about how a problem can be solved**
- Logic is a language for reasoning. It is a collection of rules we use when doing reasoning.
- Example
- John's mum has four children.
- • The first child is called April.
- • The second May.
- • The third June.
- What is the name of the fourth child?



Importance of Logic in problem solution

- Solution for any problem (summation of two numbers) requires **three** things
  Data : Input values (e.g. 2 and 3)
  Process : Process of Summation
  Output : Output after process (e.g. sum of numbers (5))
  The process part(e.g. Summation) of the solution requires logic (e.g. How to sum)
  Or in other words based on the logic the process is developed

Importance of Logic in problem solving

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	1. Divide N by numbers from 2 to N and if for all the division operations, the reminder is non zero, the number is prime otherwise it is not prime 2. Same as 1 but divide the N from 2 to N/2 3. Same as Logic 1 but divide N from 2 to square root of N

Importance of Logic in problem solving

In a fun game, MXM grid is given with full of coins. The player has to give a number 'N' of his choice. If N is lesser than M then he is out of game and doesn't gain any points. Otherwise he has to place all coins in the MXM grid in the NXN grid and he gains points equal to the number of free cells in the N X N grid.

Data	Processing	Output	Solution Alternatives
Numbers M and N	If N is less than M Points = 0 Otherwise Compute Points as N^2 - M^2	Number of points gained	1. Compute N^2 - M^2 as NXN – MXM 2. Compute (N + M) X (N – M) (Number of multiplication is reduced)

Problem 2

- Write a Problem Analysis Chart (PAC) to find an area of a circle where area = pi * radius * radius

Data	Processing	Output
radius	area = 3.14 x radius x 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 formulas are

  Celsius = 10 x (depth) + 20

  Fahrenheit = 1.8 x (Celsius) + 32

Data	Processing	Output

depth	celsius = 10 x (depth) + 20
	fahrenheit = 1.8 x (celsius) + 32
		Display celsius,
		Display fahrenheit

Problem 4

- Write a problem analysis chart (PAC) that asks a user to enter the distance of a trip in miles, the miles per gallon estimate for the user’s car, and the average cost of a gallon of gas. Calculate and display the number of gallons of gas needed and the estimated cost of the trip.

Data	Processing	Output
distance, miles per gallon, cost per gallon	gas needed = distance / miles per gallon. estimated cost = cost per gallon x gas needed	Display gas needed Display estimated cost

PRE-PROGRAMMING PHASE

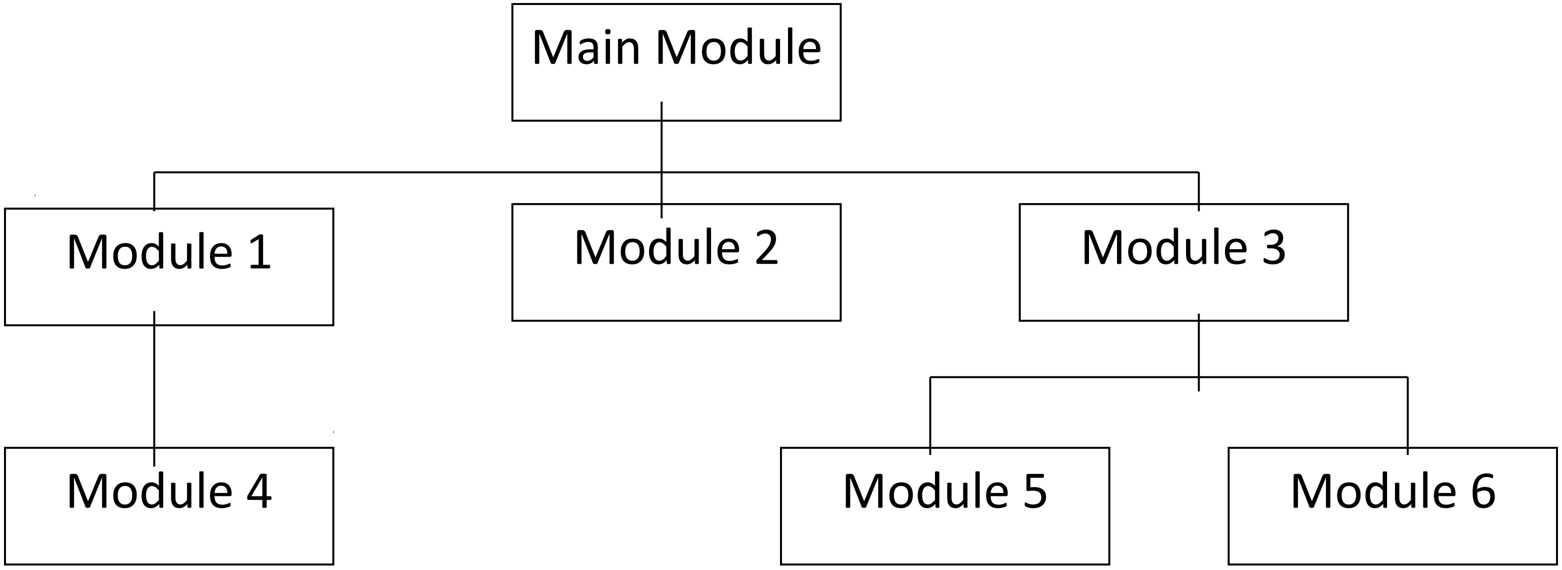
- **Developing the Hierarchy Input Process Output (HIPO) or Interactivity Chart**
  - The problem is normally big and complex.
  - Thus, requires big program.
  - Thus, the processing can be divided into subtasks called modules.
  - Each module accomplishes one function.
  - These modules are connected to each other to show the interaction of processing between the modules.

PRE-PROGRAMMING PHASE

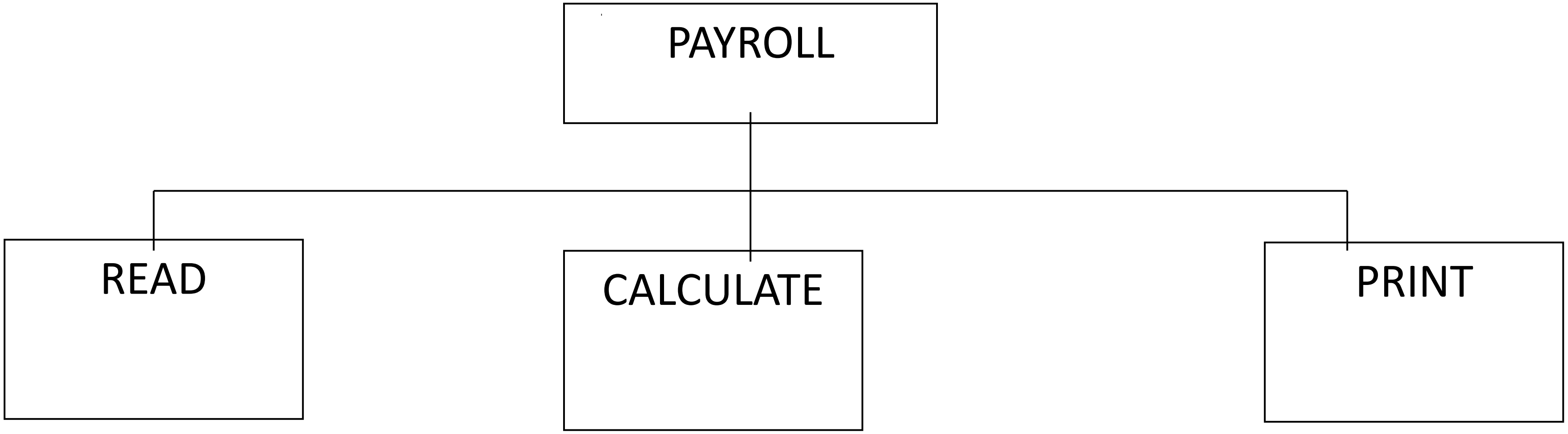
- Main/control module controls the flow all other modules.
- The IC is developed using top-down-method: top to down left to right order (also refer to order of processing).
- Modules are numbered, marked for duplication, repetition or decision.

PRE-PROGRAMMING PHASE

- The interaction will form a hierarchy, called Hierarchy Input Process Output Chart (HIPO) or Interactivity Chart (IC). Programming which use this approach (problem is divided into subtasks) is called *Structured Programming*.



PRE-PROGRAMING PHASE


PAYROLL

READ

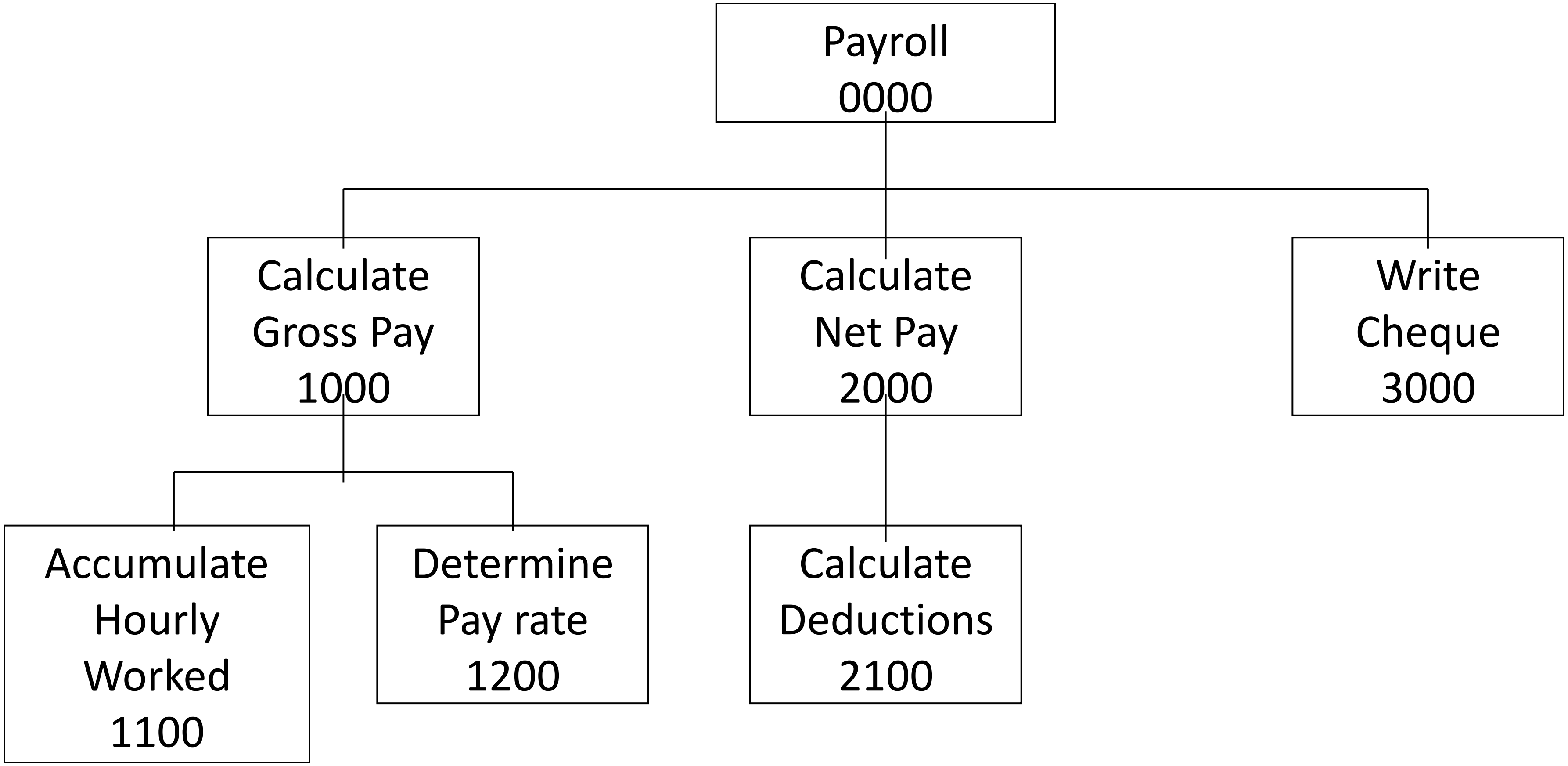
CALCULATE

PRINT

PRE-PROGRAMMING PHASE

- **Example 2.2: Extended Payroll Problem**
- You are required to write a program to calculate both the gross pay and the net pay of every employee of your company. To determine the gross pay, you have to multiply the accumulated total hours worked by the employee, by the appropriate pay rate. The program should print the cheque that tells the total net pay. The net pay is calculated by subtracting the gross pay with any deductions that may be incurred by the employee.

PRE-PROGRAMMING PHASE

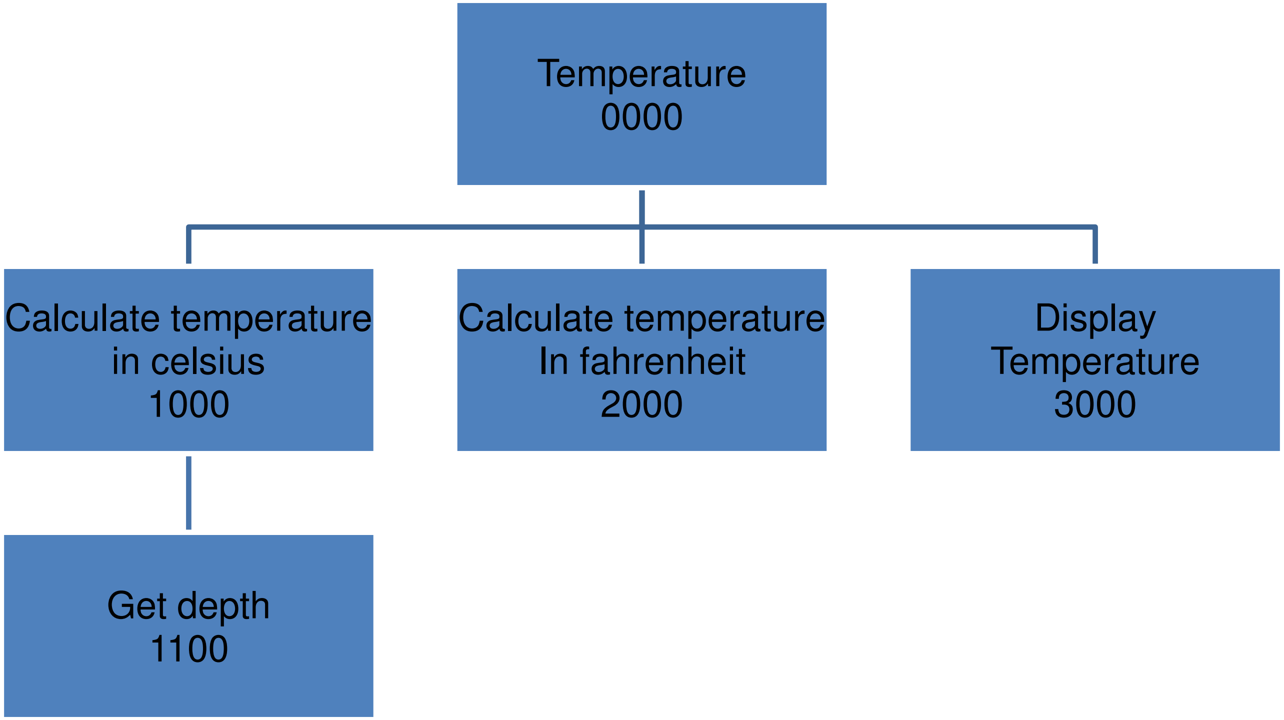


Problem 1

- Write a Hierarchy Input Process Output (HIPO) to compute and display the temperature inside the earth in Celsius and Fahrenheit. The relevant formulas are

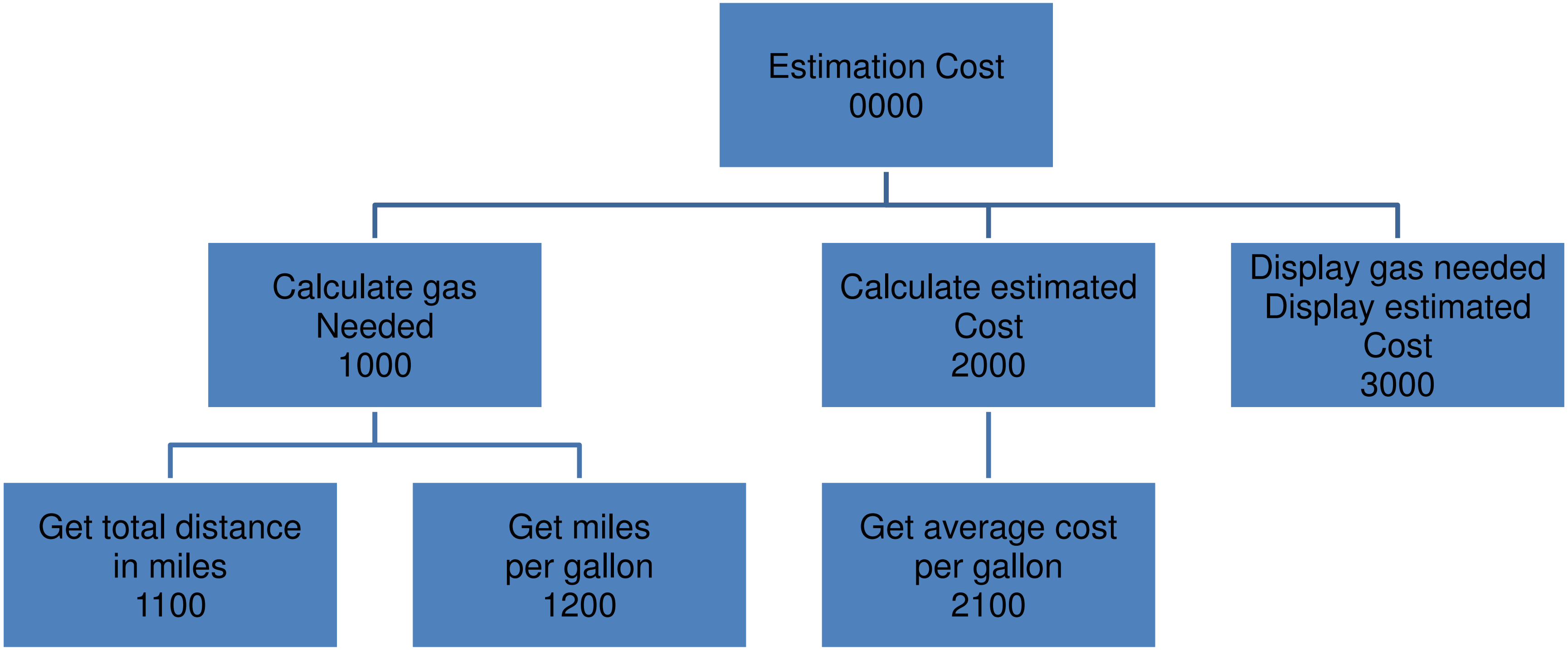
  Celsius = 10 x (depth) + 20

  Fahrenheit = 1.8 x (Celsius) + 32



Problem 2

- Write a Hierarchy Input Process Output (HIPO) that asks a user to enter the distance of a trip in miles, the miles per gallon estimate for the user’s car, and the average cost of a gallon of gas. Calculate and display the number of gallons of gas needed and the estimated cost of the trip.



Estimation Cost
0000

Calculate gas
Needed
1000

Calculate estimated
Cost
2000

Display gas needed
Display estimated
Cost
3000

Get total distance
in miles
1100

Get miles
per gallon
1200

Get average cost
per gallon
2100

PRE-PROGRAMMING PHASE

- **Developing the Input Process Output (IPO) Chart**
  - Extends and organizes the information in the Problem Analysis Chart.
  - It shows in more detail what data items are input, what are the processing or modules on that data, and what will be the result or output.
  - It combines information from PAC and HIPO Chart.

PRE-PROGRAMING PHASE

Input	Processing	Module	Output
-Hours Worked -Pay Rate -Deduction	-Enter Hourly Worked -Enter Pay Rate -Calculate Gross Pay -Enter Deductions -Calculate Net Pay -Print Cheque -End	1100 1200 1000 2100 2000 3000 0000	-Net pay

Problem 4

- Write an Input Process Output (IPO) that asks a user to enter the distance of a trip in miles, the miles per gallon estimate for the user’s car, and the average cost of a gallon of gas. Calculate and display the number of gallons of gas needed and the estimated cost of the trip.

Input	Processing	Module	Output
- Distance in miles - Miles per gallon - Cost gas per gallon	- Enter distance - Enter miles per gallon - Calculate total gas needed - Enter cost gas per gallon - Calculate estimated cost - Display total gas and estimated cost - End	1100 1200 1000 2100 2000 3000 0000	-Total gas needed - Estimated cost

PRE-PROGRAMMING PHASE

- **Drawing the Program Flowcharts**
  - Flowchart is the graphic representations of the individual steps or actions to implement a particular module.
  - The 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.

PRE-PROGRAMMING PHASE

– Flowchart is the logical design of a program.

– It is the basis from which the actual program code is developed.

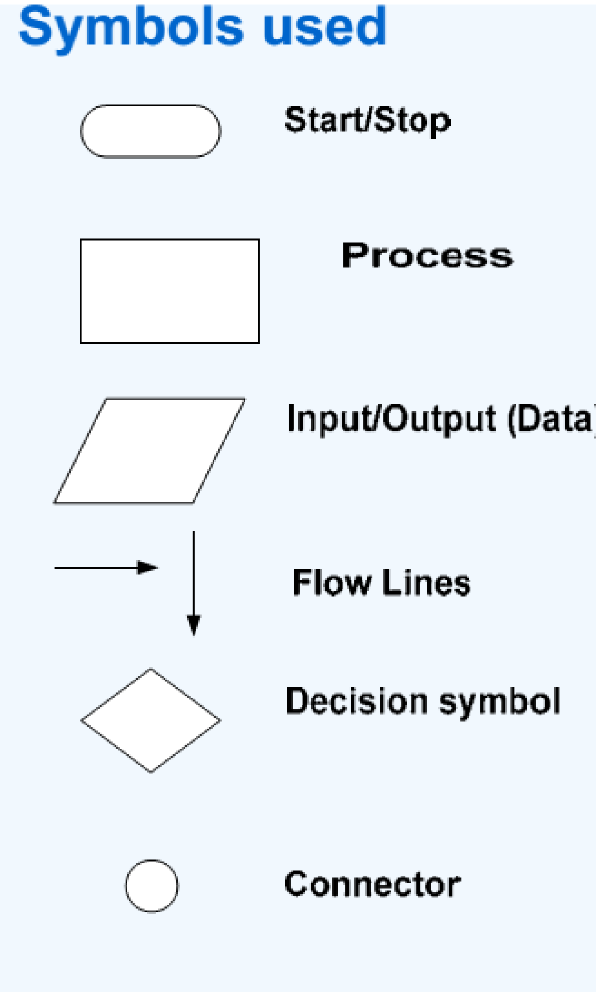
– Flowchart serves as documentation for computer program.

– The flowchart must be drawn according to definite rules and utilizes standard symbols adopted internationally.

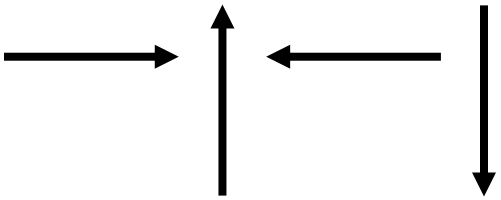
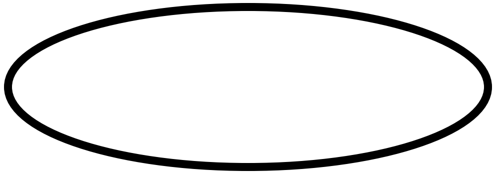
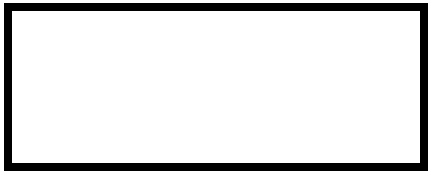
– The International Organization for Standardization (IOS) was the symbols shown below (You can draw the symbols using ready-made flowcharting template):

Flow Charts

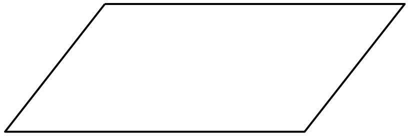
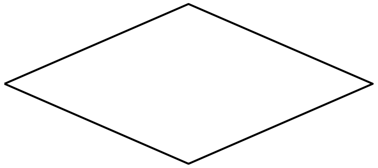
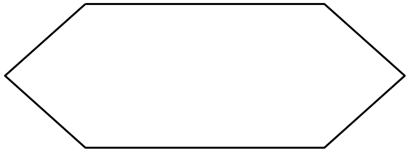
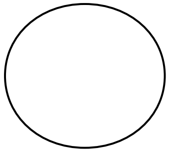
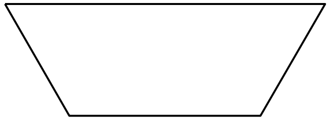
-
  - A flowchart is a diagrammatic representation of an algorithm.
  - A flow chart is an organized combination of shapes, lines and text that graphically illustrate a process or structure.



PRE-PROGRAMMING PHASE

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.

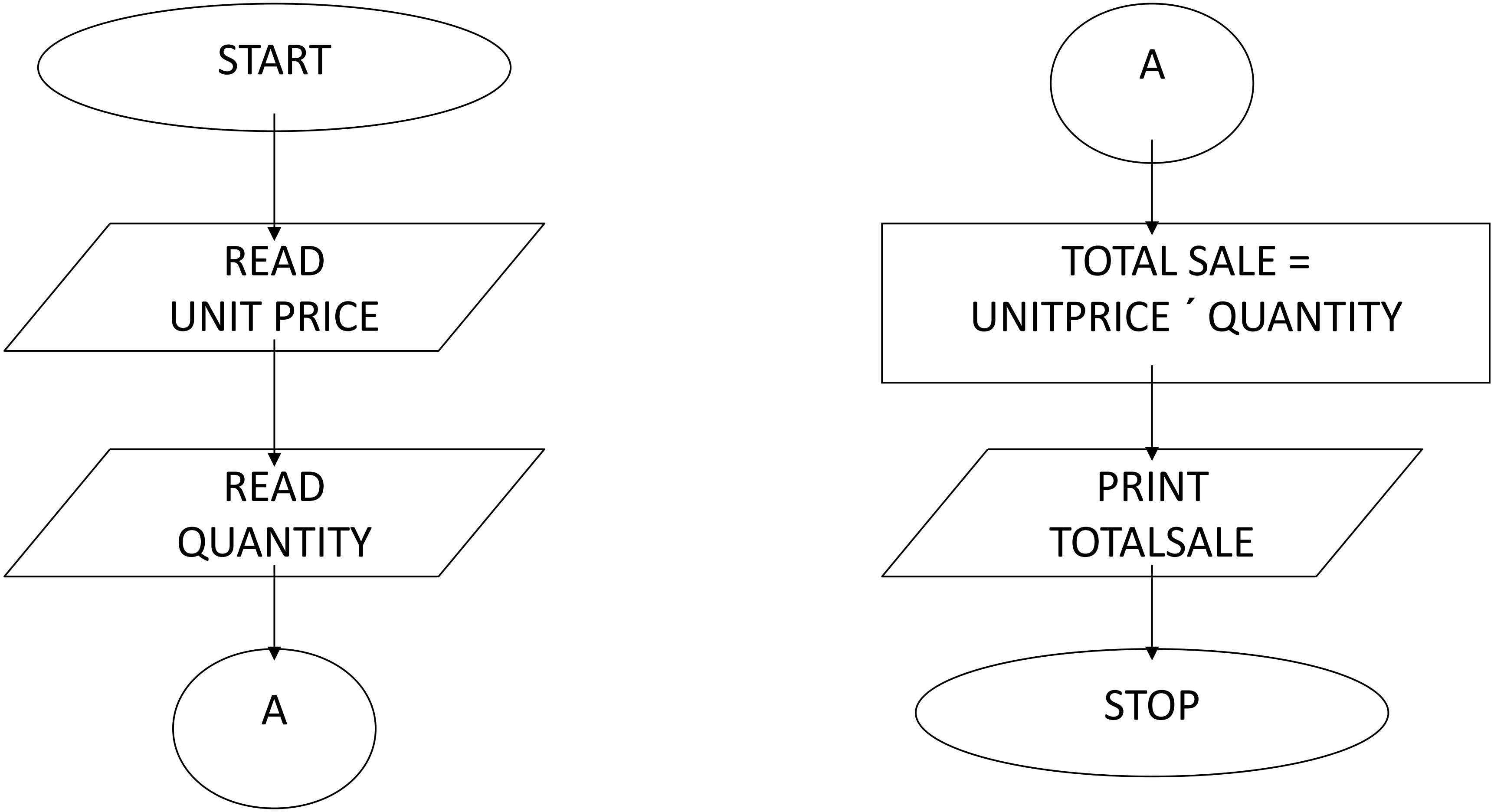
PRE-PROGRAMMING PHASE

	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.

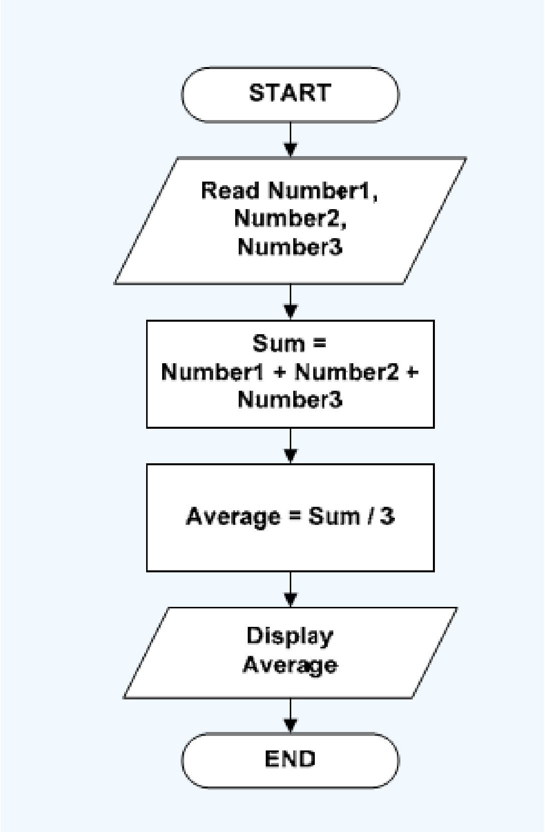
PRE-PROGRAMMING PHASE

- **Example 2.3 : Sale Problem**
  - Draw a flowchart for a problem that to read two numbers. The first number represents the unit price of a product and the second number represents the quantity of the product sold. Calculate and print the total sale.
  - Solution: Stepwise Analysis of the Sale Problem
    - Start of processing
    - Read the unit price
    - Read the quantity
    - Calculate total sale
    - Print total sale
    - Stop the processing

PRE-PROGRAMMING PHASE



Example: Flow Chart (Sequential)


START
Read Number1, Number2, Number3
Sum = Number1 + Number2 + Number3
Average = Sum / 3
Display Average
END

- Find the average of three numbers

Structuring a Program

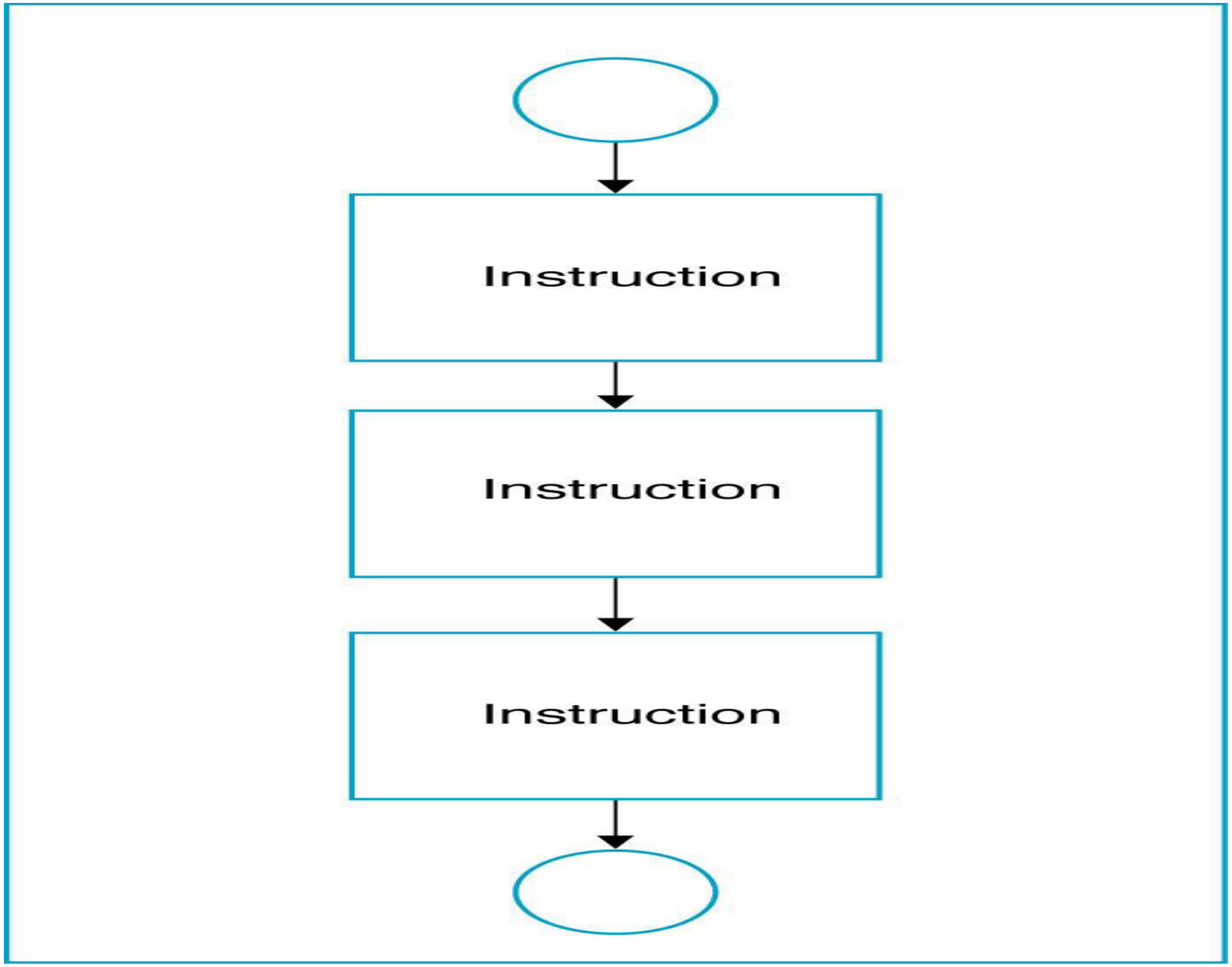
- Develop efficient computer solution to problems:
  1. Use Modules
  2. Use four logic structures
     a. Sequential structure
        - Executes instructions one after another in a sequence.

     b. Decision structure
        - Branches to execute one of two possible sets of instructions.

     c. Loop structure
        - Executes set of instruction many times.

     d. Case structure
        - Executes one set of instructions out of several sets.
  3. Eliminate rewriting of identical process by using modules.
  4. Use techniques to improve readability including four logic structure, proper naming of variables, internal documentation and proper indentation.

Sequential Logic Structure

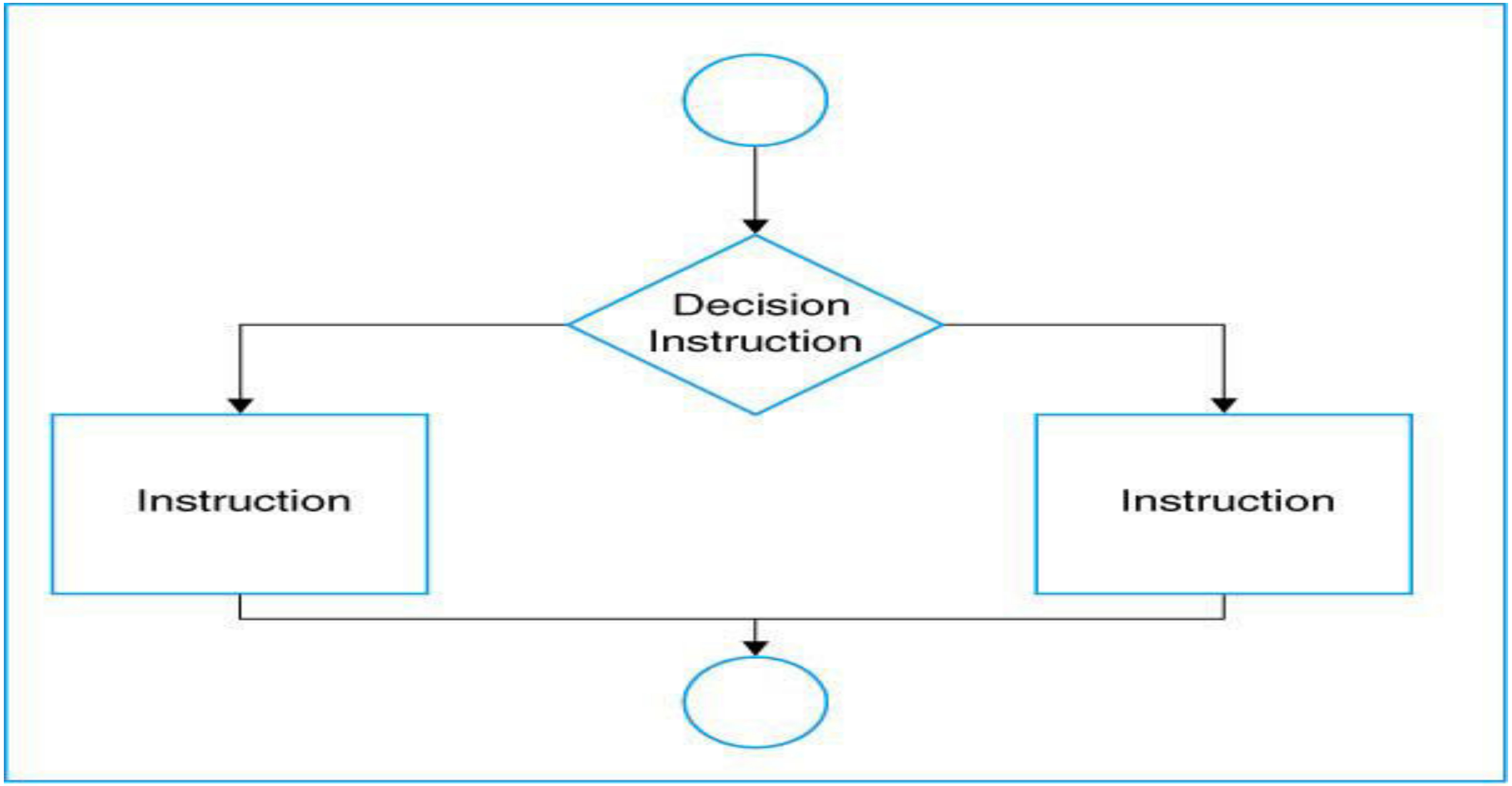


The Decision Logic Structure

- 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



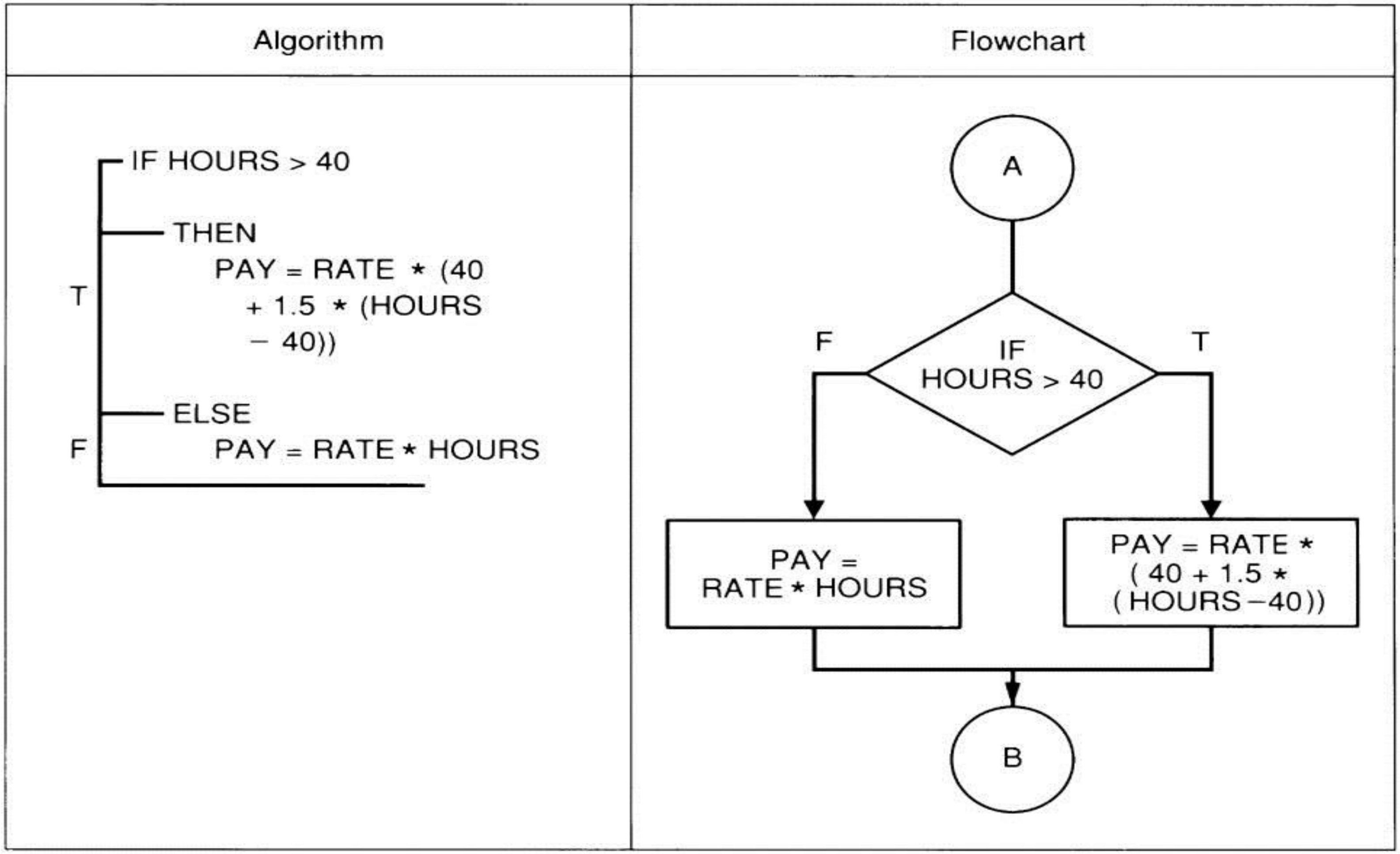
Examples of conditional expressions

- A < B (A and B are the same data type – either numeric, character, or string)
- X + 5 >= Z (X and Z are numeric data)
- E < 5 or F > 10 (E and F are numeric data)
- DATAOK (DATAOK – logical datum)

Example

- Assume your 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



Algorithm

```
IF HOURS > 40
   THEN
      PAY = RATE * (40
        + 1.5 * (HOURS
        − 40))
T
   ELSE
F     PAY = RATE * HOURS
```

Flowchart

Note: For all flowcharts with decision blocks, T = TRUE and F = FALSE

NESTED IF/THEN/ELSE INSTRUCTIONS

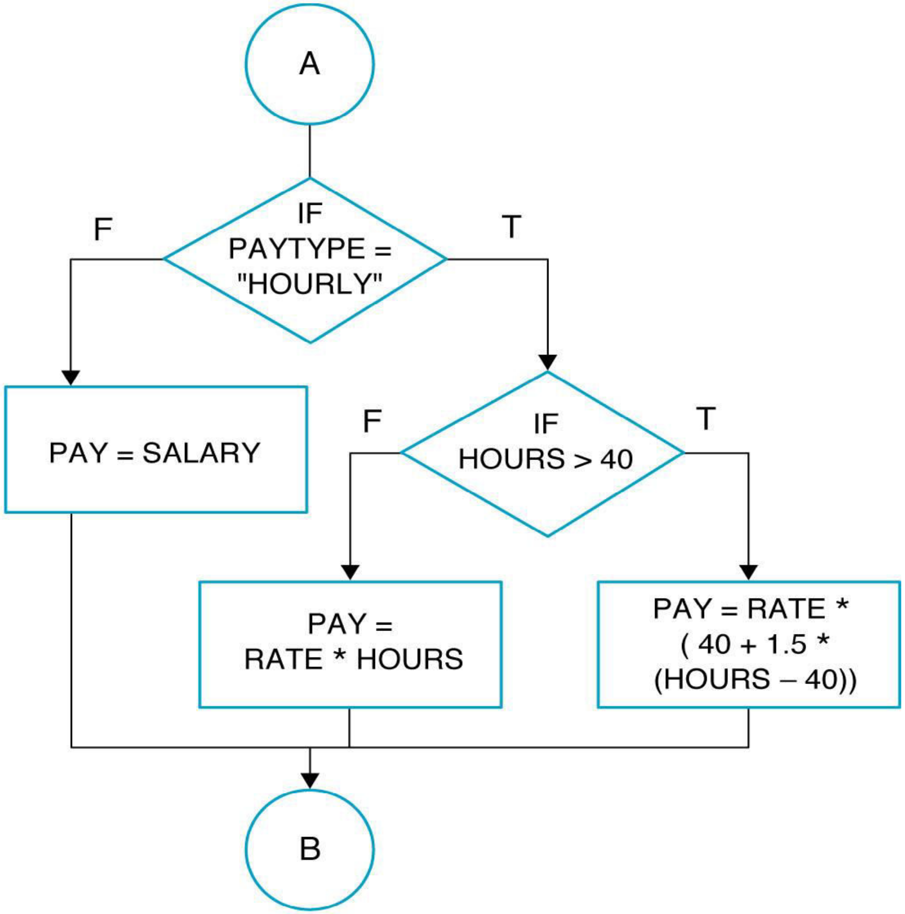
- Multiple decisions.
- Instructions are sets of instruction in which each level of a decision is embedded in a level before it.

NESTED IF/THEN/ELSE INSTRUCTIONS

Algorithm

```
         IF PAYTYPE = "HOURLY"
           THEN
              IF HOURS > 40
                THEN
                  PAY = RATE * (40
T     T              + 1.5 * (HOURS
                     – 40))
                ELSE
      F           PAY = RATE * HOURS
           ELSE
F            PAY = SALARY
```

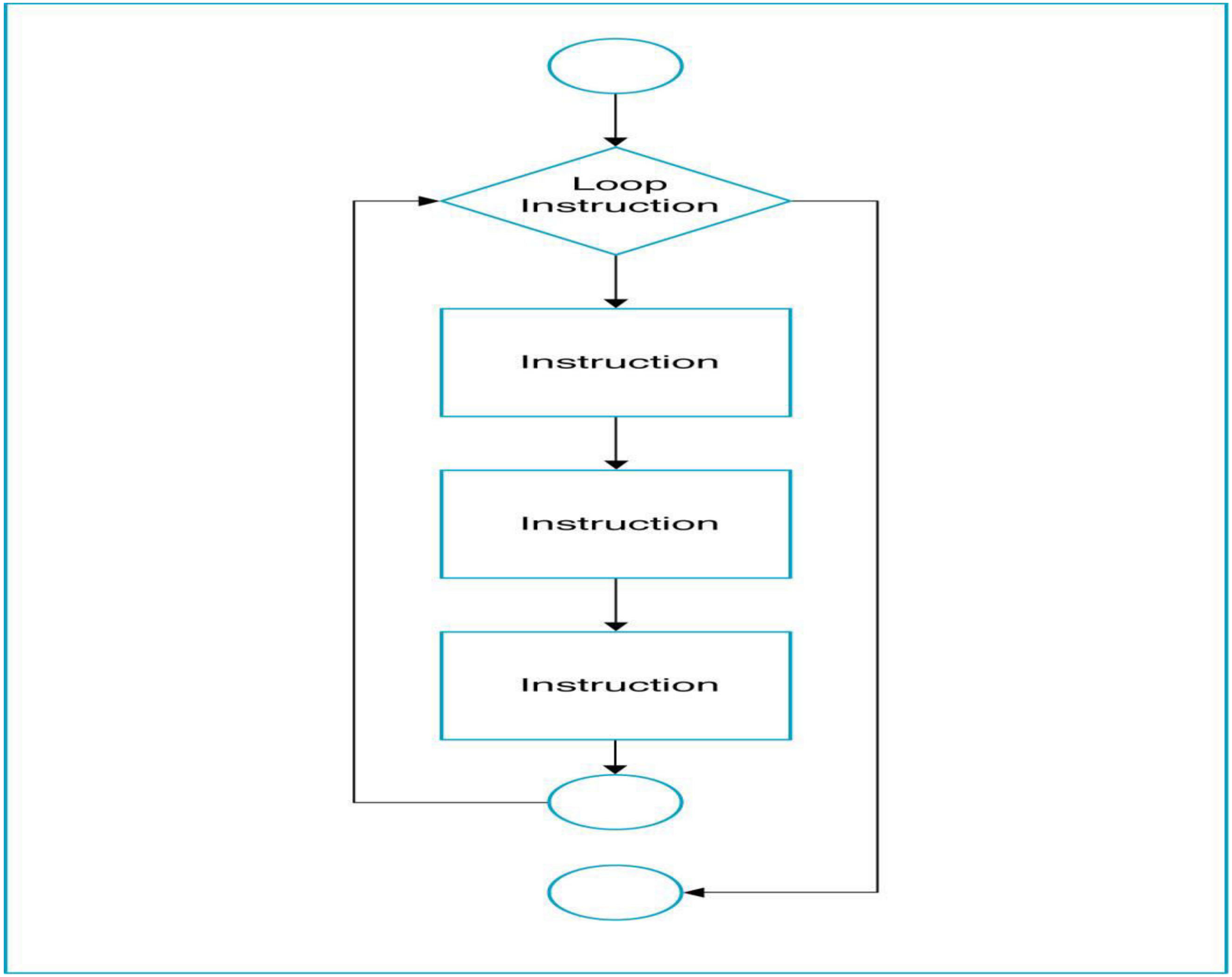
Flowchart



The Loop Logic Structure

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

Loop Logic Structure

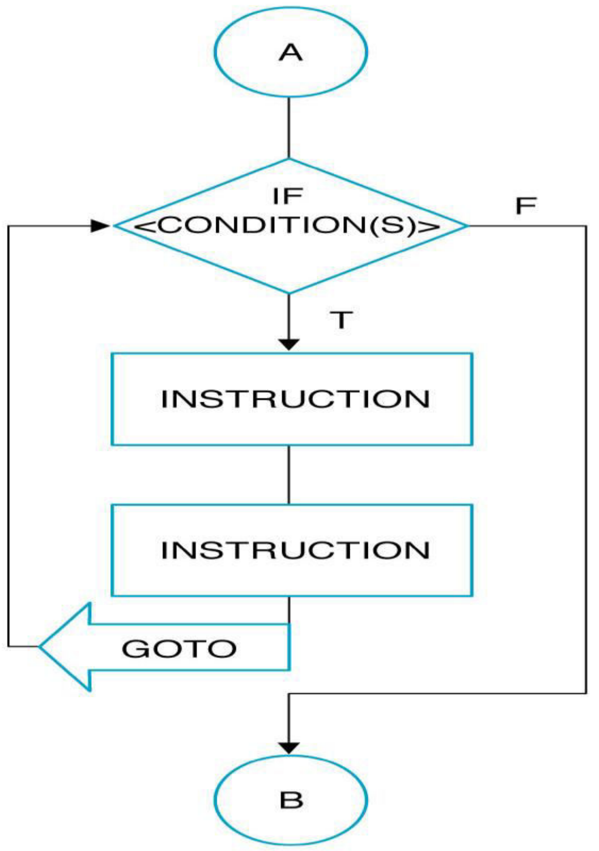


WHILE loop

Algorithm

```
100  IF <CONDITION(S)>
       THEN
 T        INSTRUCTION
          INSTRUCTION
          GOTO 100
```

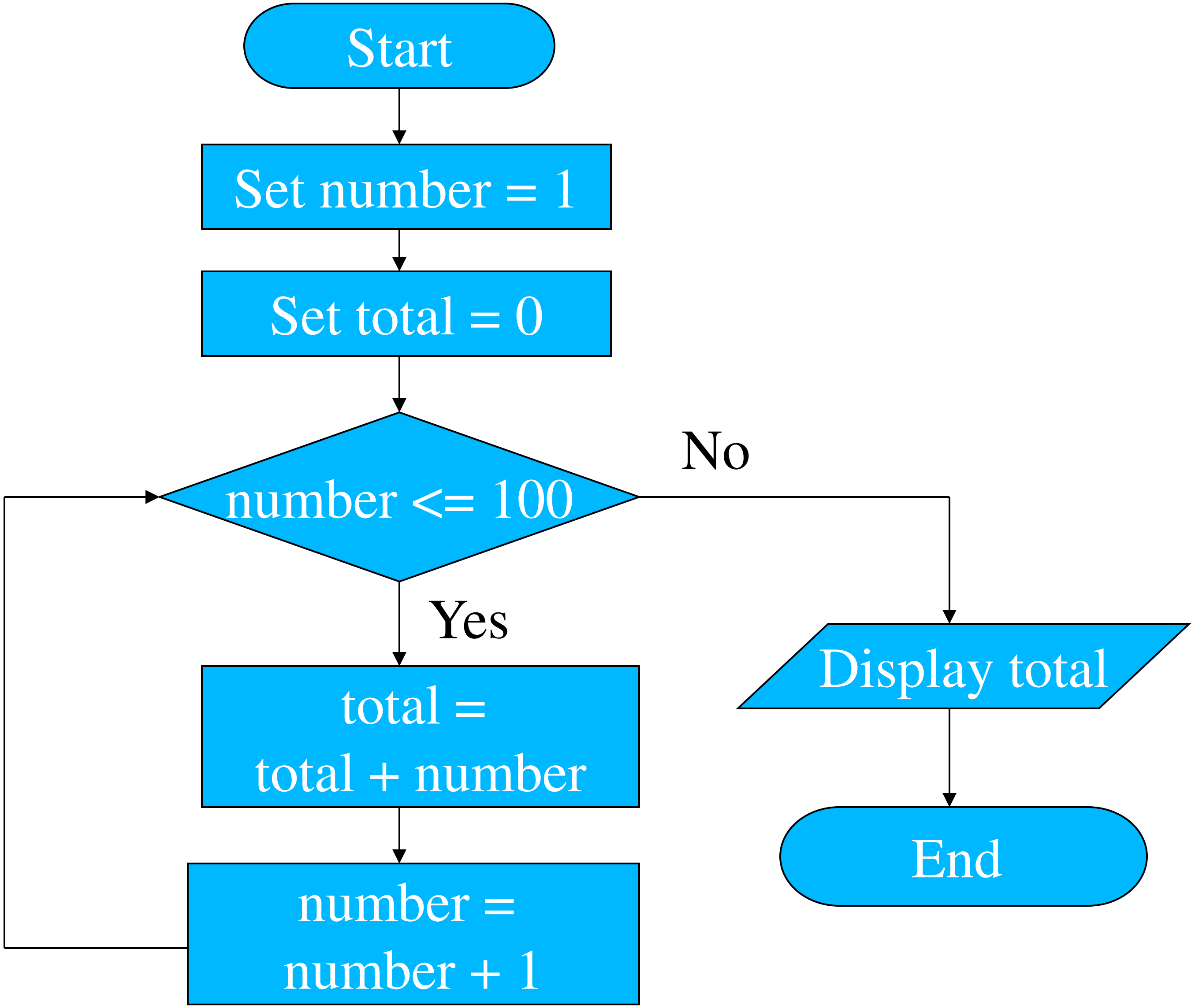
Flowchart



WHILE loop

- Do the loop body if the condition is true.
- Example: Get the sum of 1, 2, 3, …, 100.
    - Algorithm:
        - Set the number = 1
        - Set the total = 0
        - While (number <= 100)
            - total = total + number
            - number = number + 1
        - End While
        - Display total

WHILE loop


Start

Set number = 1

Set total = 0

number <= 100

No

Yes

total =
total + number

number =
number + 1

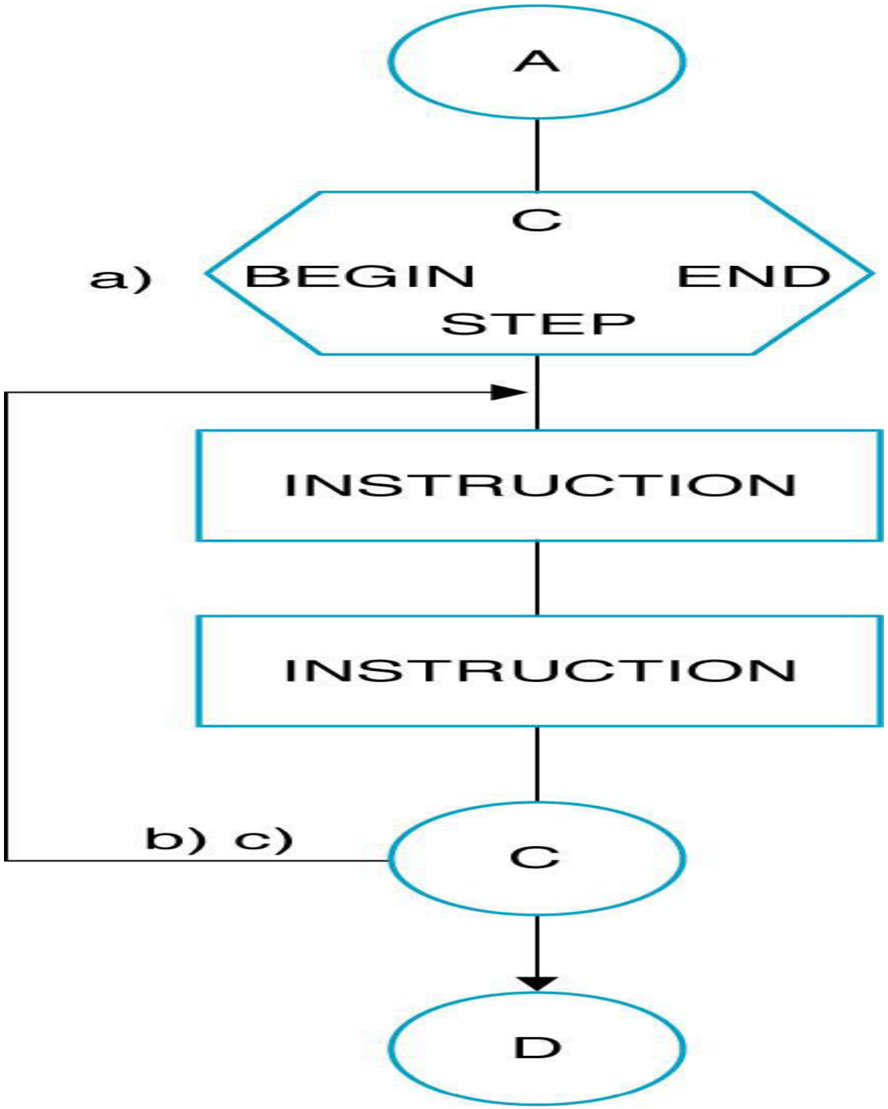
Display total

End

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

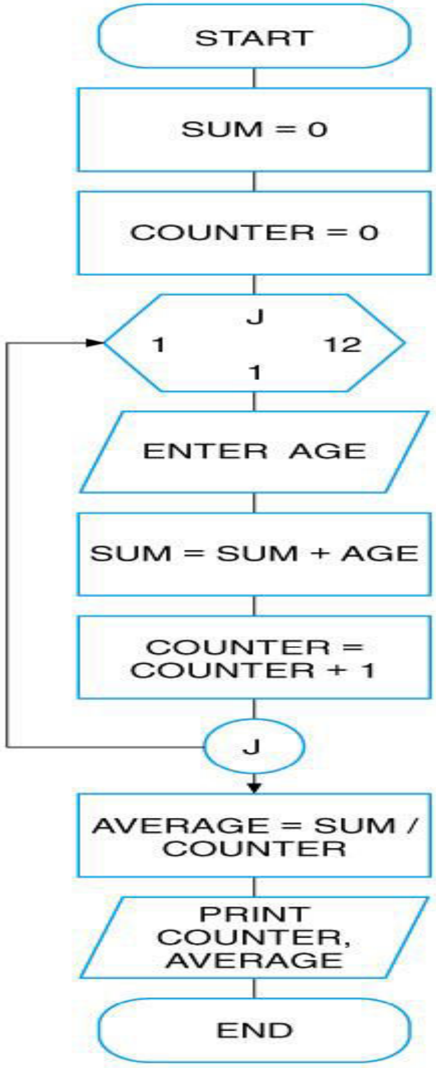


Automatic-Counter Loop

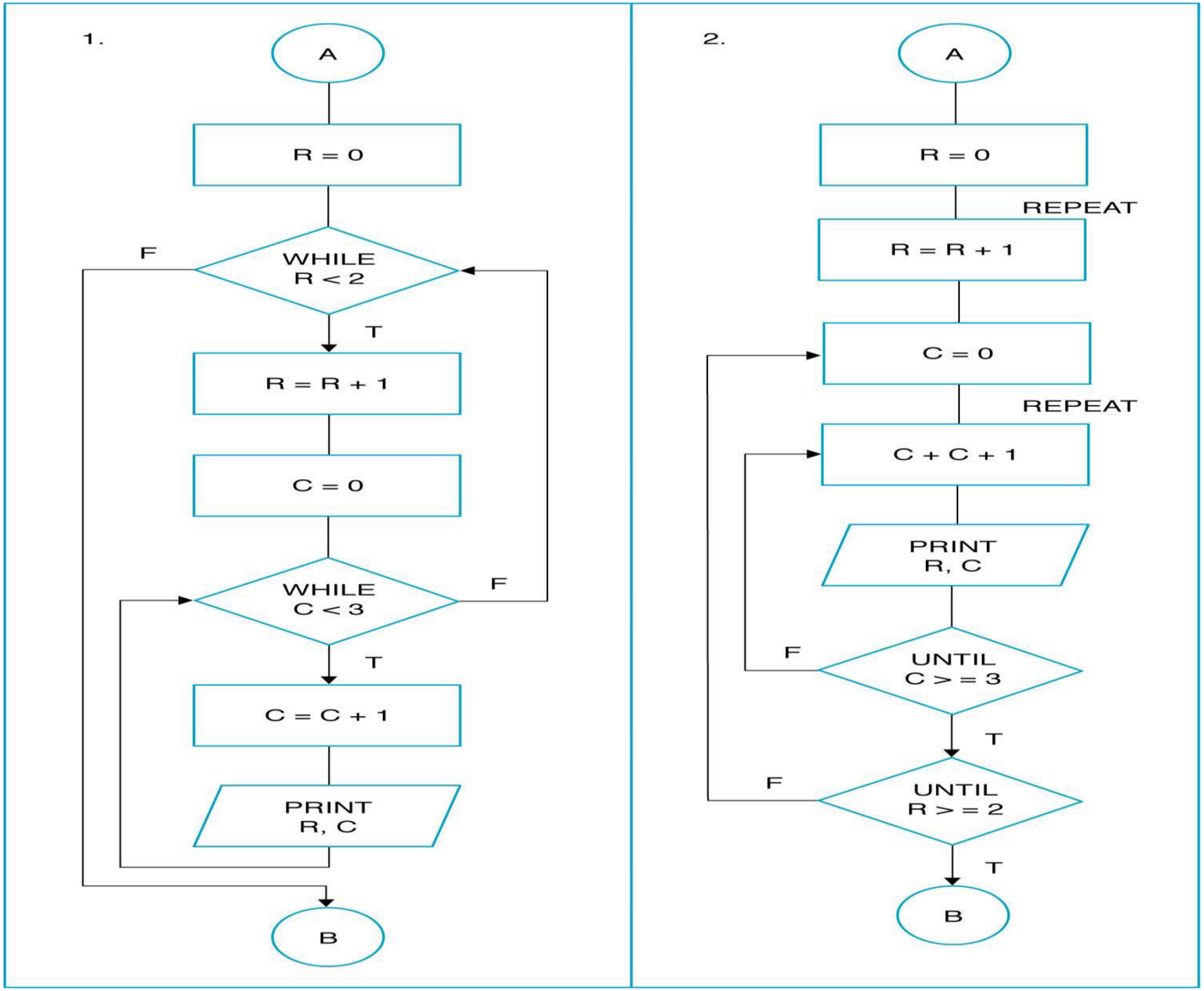
Algorithm

```
AVERAGE AGE
1. SUM = 0
2. COUNTER = 0
3. ┌LOOP: J = 1 TO 12
   │      SUM = SUM + AGE
   │      COUNTER = COUNTER + 1
   └LOOP-END: J
4. AVERAGE = SUM/COUNTER
5. PRINT COUNTER, AVERAGE
6. END
```

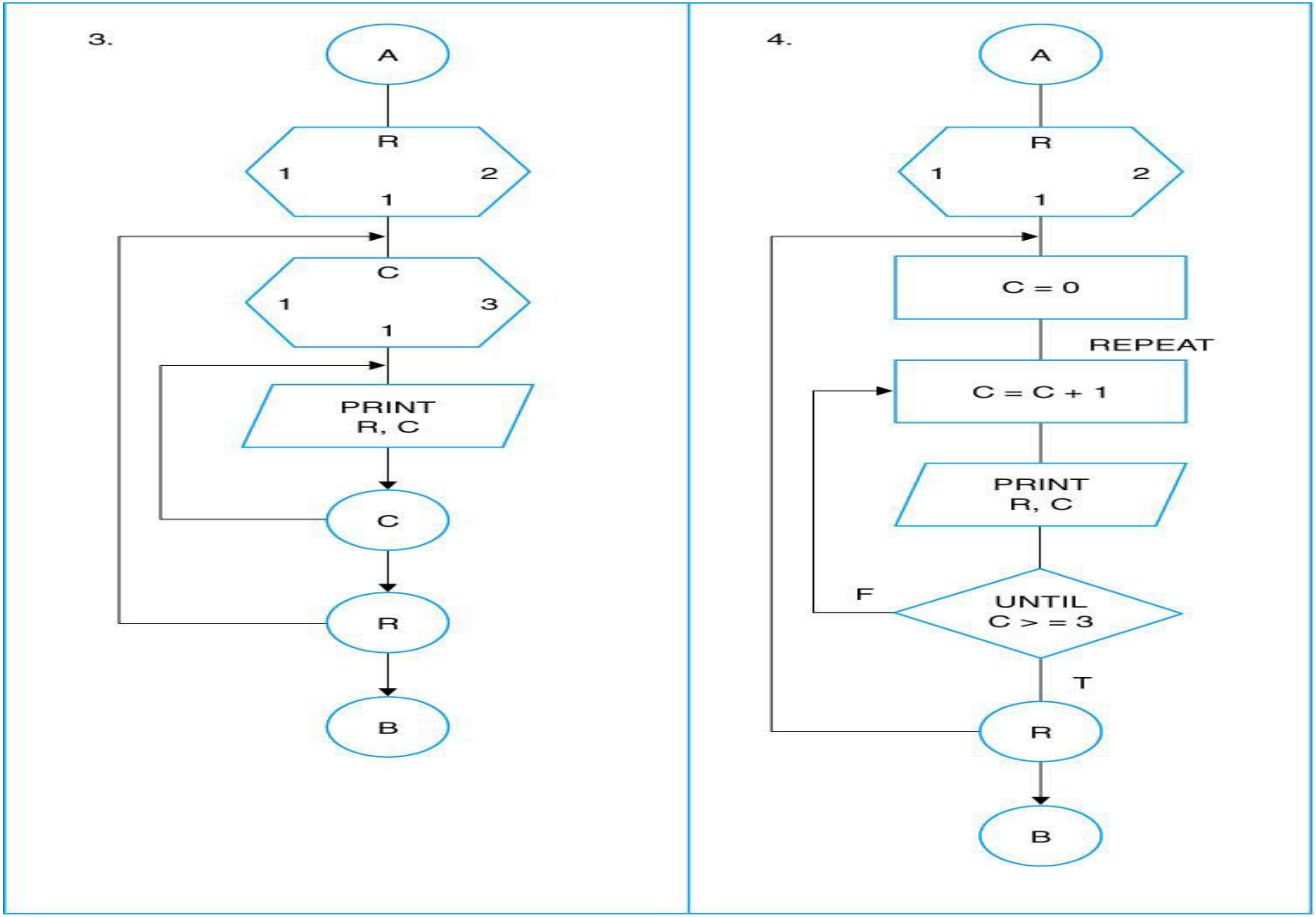
Flowchart



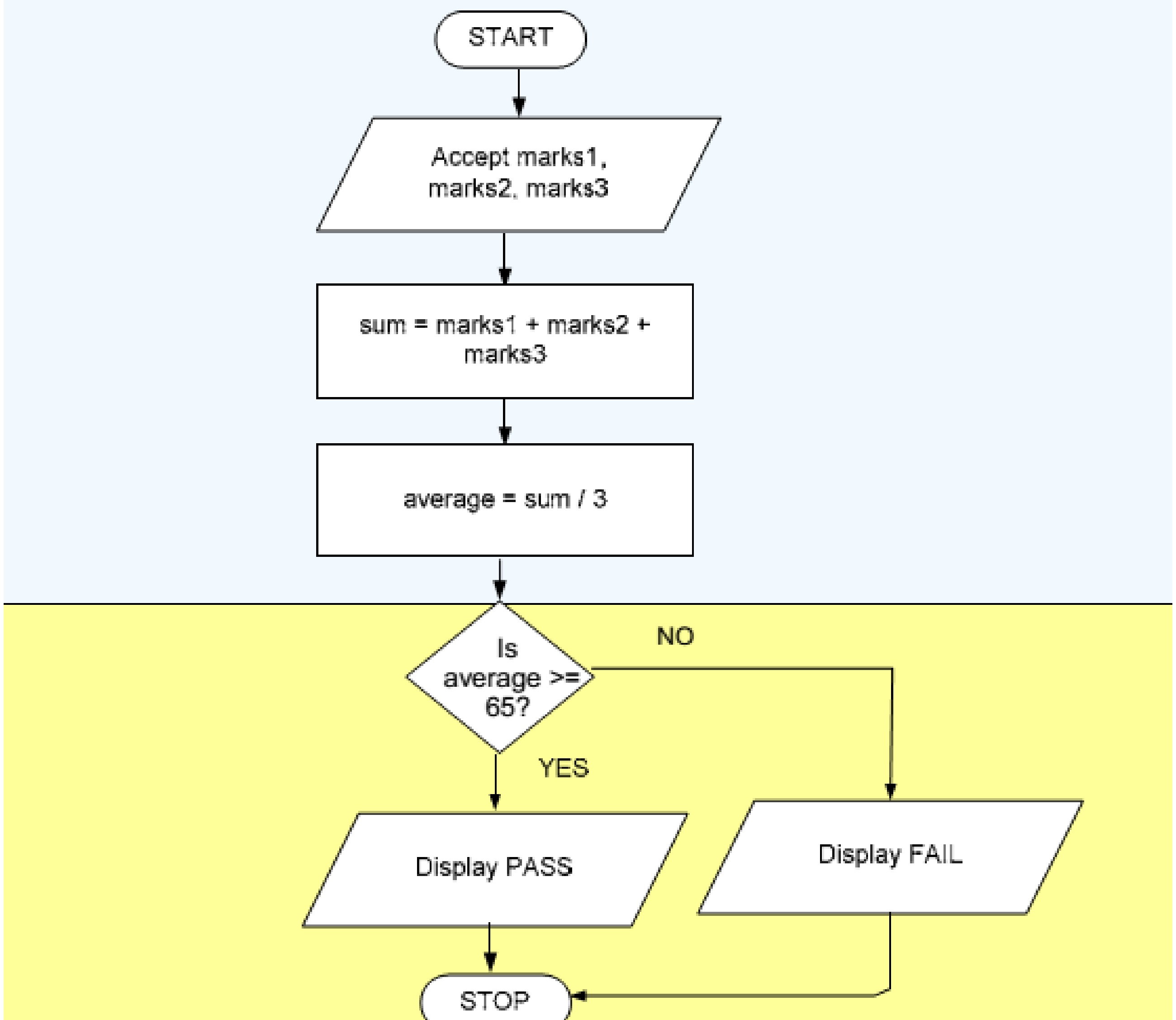
NESTED LOOP


1.
A
R = 0
F
WHILE R < 2
T
R = R + 1
C = 0
WHILE C < 3
F
T
C = C + 1
PRINT R, C
B
2.
A
R = 0
REPEAT
R = R + 1
C = 0
REPEAT
C + C + 1
PRINT R, C
F
UNTIL C > = 3
T
F
UNTIL R > = 2
T
B

NESTED LOOP


3.
A
R
1
2
1
C
1
3
1
PRINT
R, C
C
R
B
4.
A
R
1
2
1
C = 0
REPEAT
C = C + 1
PRINT
R, C
F
UNTIL
C > = 3
T
R
B

Flow Chart - Selectional



Example (Iterational)

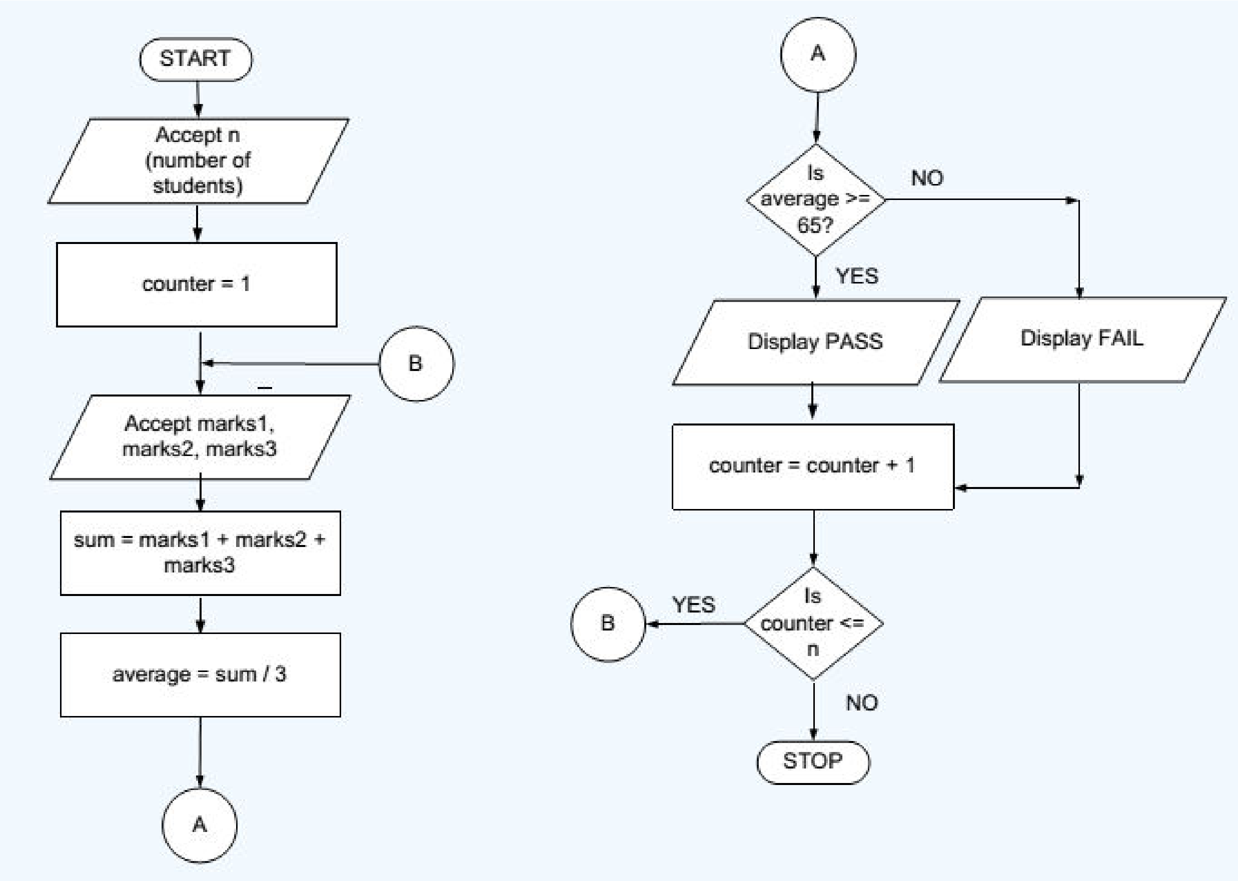
-
    - Do the following for N input values. Read N from user

      − Write a program to find the average of a student given the marks he
      obtained in three subjects.

      − Then test whether he passed or failed.

      − For a student to pass, average should not be less than 65.

Flow Chart – Example (Iterational)

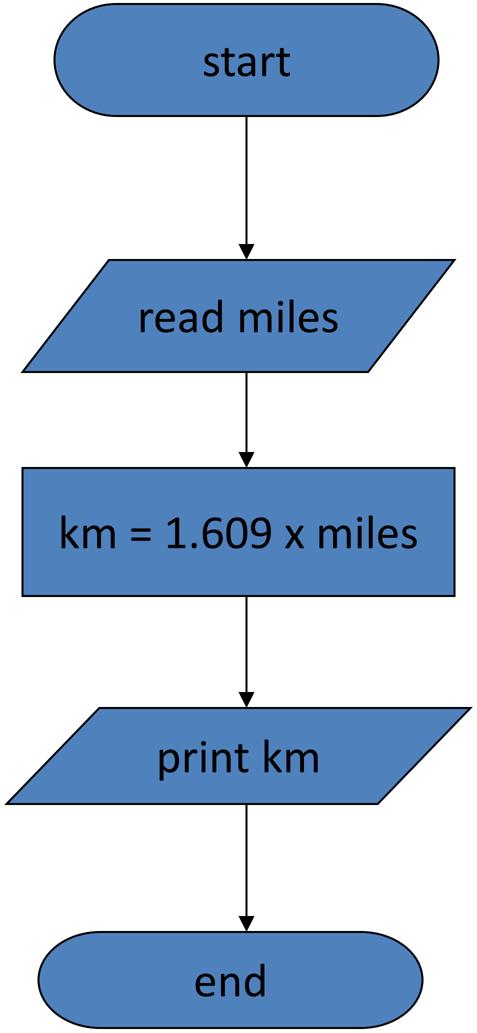


Tool demo

- Demo of Flow Chart Drawing tool
- - Visio
- -Raptor
- -Alice
- -Convert algorithm to pseudo code
- * Quiz

Problem 1

Draw a flow chart for To convert
the distance in miles to kilometers where 1.609
kilometers per mile.


start

read miles

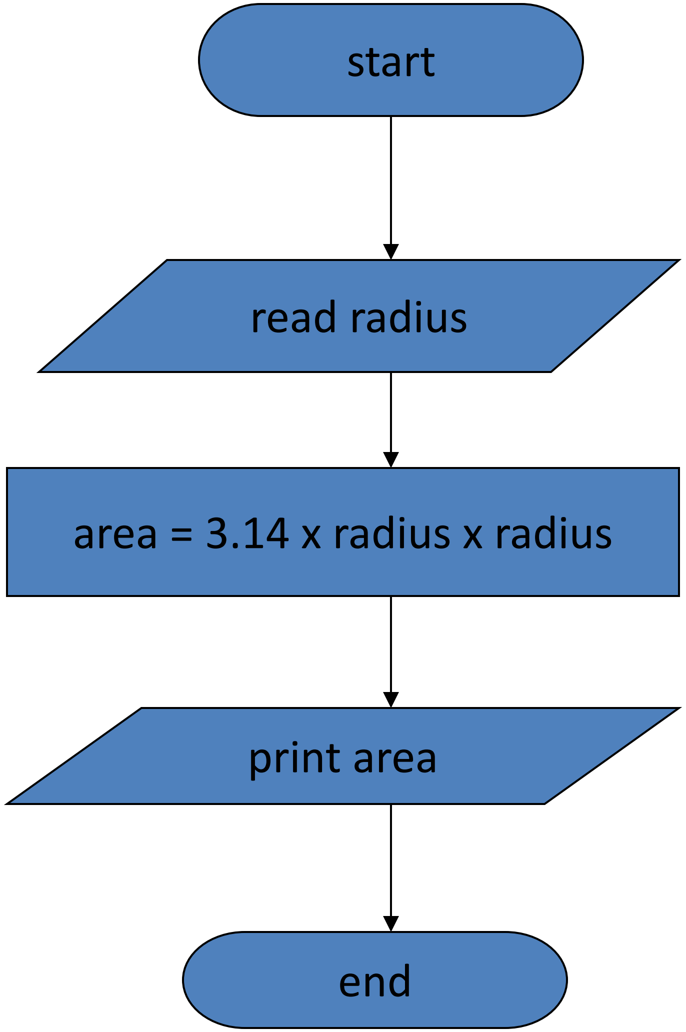
km = 1.609 x miles

print km

end

Problem 2

- Draw a flow chart for find an area of a circle where area = pi * radius * radius



start

read radius

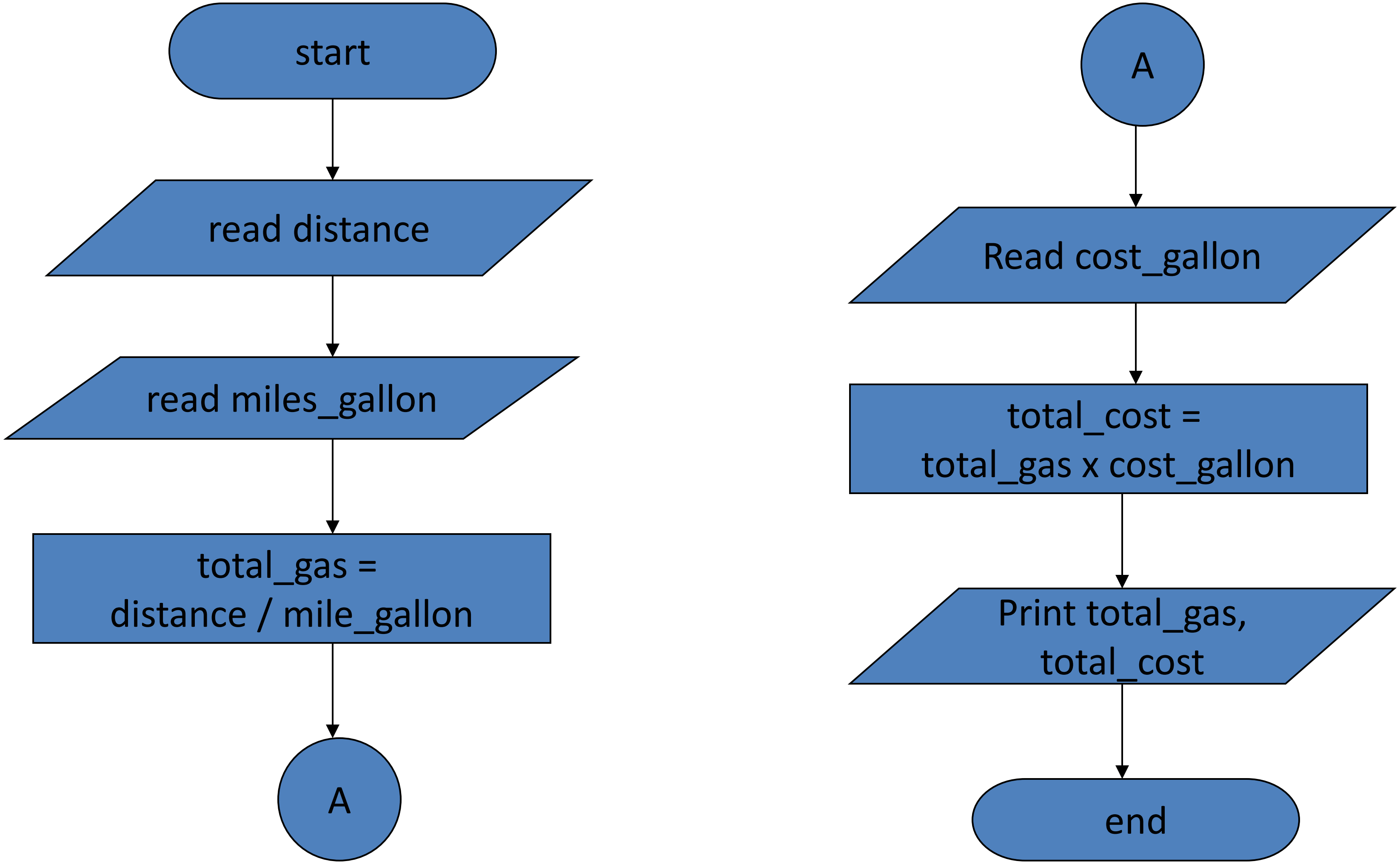
area = 3.14 x radius x radius

print area

end

Problem 3

- Draw a flow chart for user to enter the distance of a trip in miles, the miles per gallon estimate for the user’s car, and the average cost of a gallon of gas. Calculate and display the number of gallons of gas needed and the estimated cost of the trip.



start

read distance

read miles_gallon

total_gas =
distance / mile_gallon

A

A

Read cost_gallon

total_cost =
total_gas x cost_gallon

Print total_gas,
total_cost

end

Algorithm

- Representation of computation is known as algorithm
- 'In Computer Science following notations are used to represent algorithm
- ■ Flowchart: This is a graphical representation of computation
- ■ Pseudo code: They usually look like English statements but have additional qualities

Algorithm

- • A step by step procedure to solve a problem
  • Properties
  – a finite sequence of steps
  – Each step shall be explicit and unambiguous
  • Algorithms are not specific to any programming language
  • An algorithm can be implemented in any programming language
  • Use of Algorithms
  – Facilitates easy development of programs
  – Iterative refinement
  – Easy to convert it to a program
  – Review is easier

Algorithm (2 of 2)

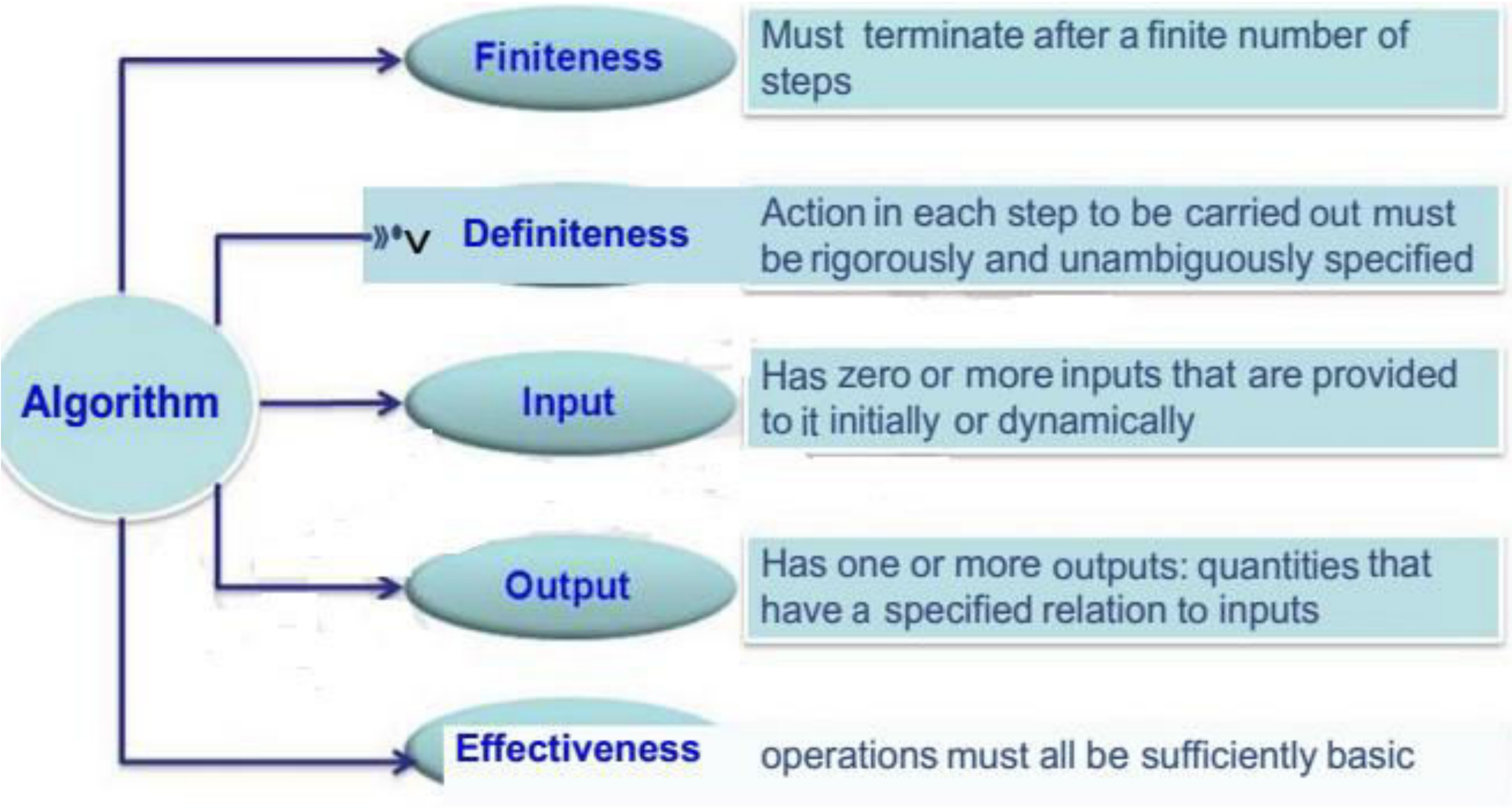
-
  - Identify the Inputs and Outputs
  - Identify any other data and constants required to solve the problem
  - Identify what needs to be computed
  - Write an algorithm

Algorithm

- **Definition:**
  • Finite set of steps to accomplish a task
  • Step-by-step, simple, mechanical procedure to compute a function on every possible input
  Example :
  Algorithm for adding two numbers

```
Begin
Step 1: Accept two numbers in A and B
Step 2: C = A + B
Step 3: Display C
END
```

Properties of an Algorithm



Algorithm

Finiteness

Must terminate after a finite number of steps

Definiteness

Action in each step to be carried out must be rigorously and unambiguously specified

Input

Has zero or more inputs that are provided to it initially or dynamically

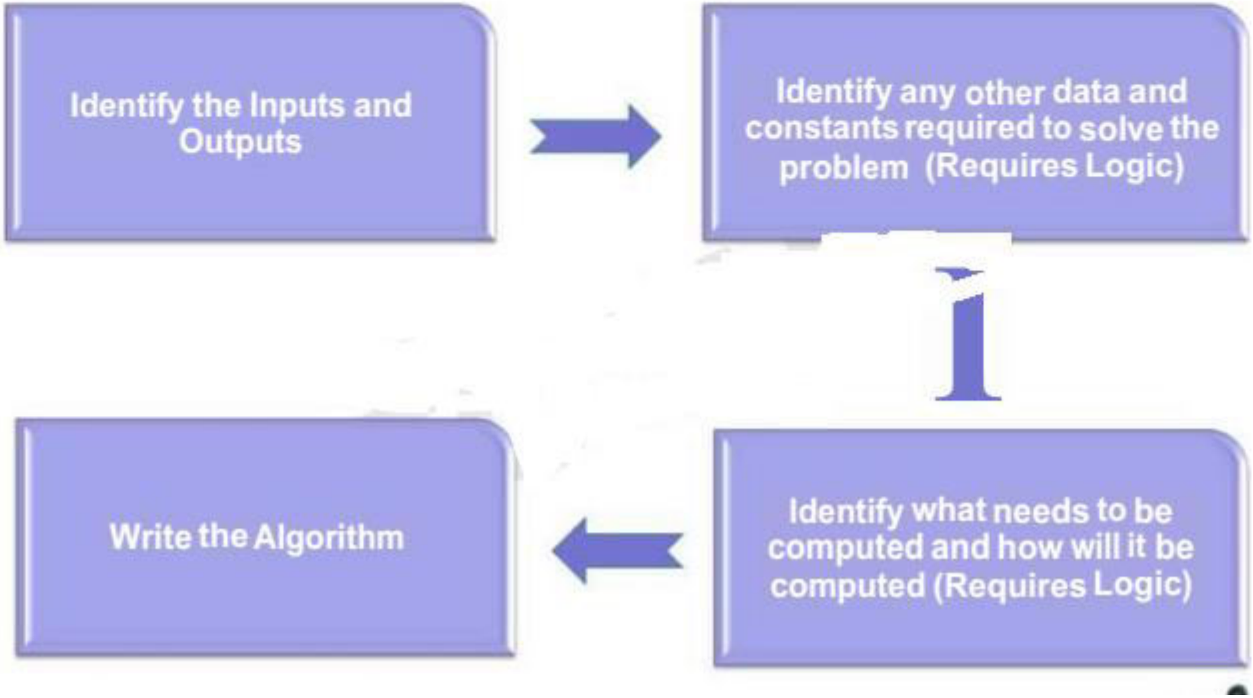
Output

Has one or more outputs: quantities that have a specified relation to inputs

Effectiveness

operations must all be sufficiently basic

Steps to Develop an Algorithm



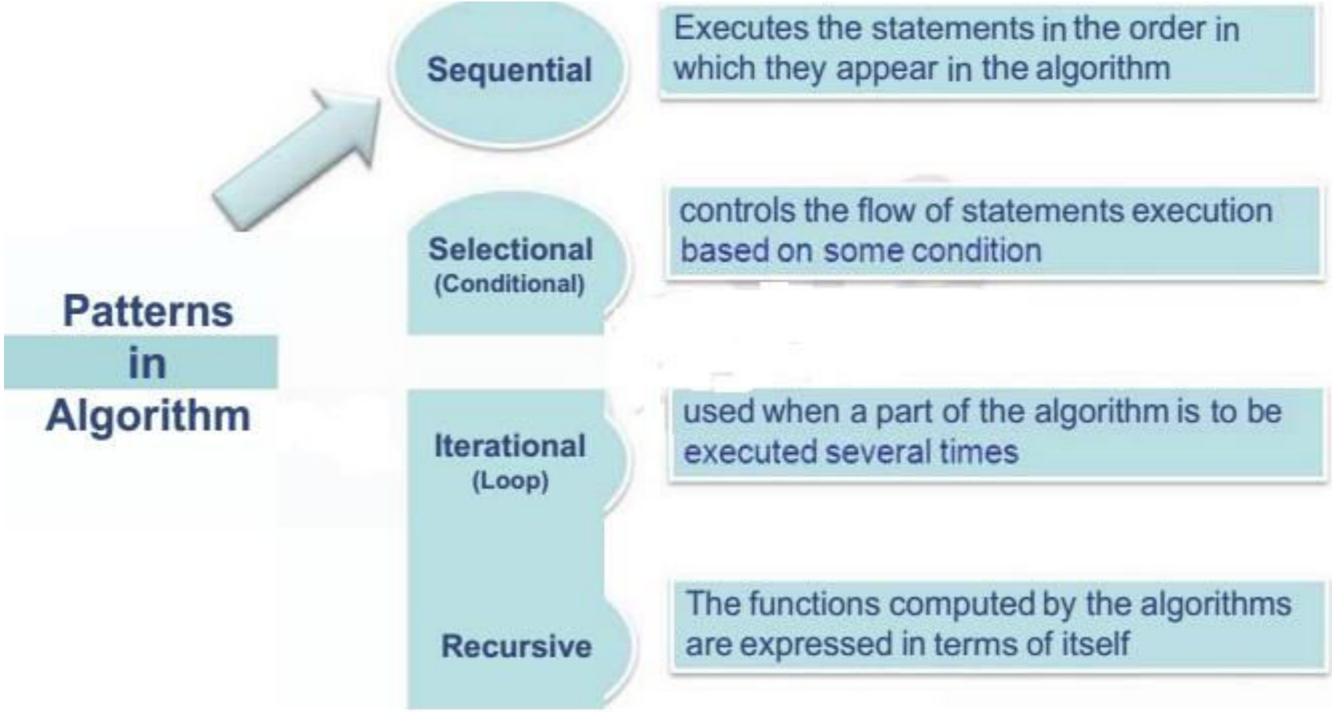
Identify the Inputs and Outputs

Identify any other data and constants required to solve the problem (Requires Logic)

Identify what needs to be computed and how will it be computed (Requires Logic)

Write the Algorithm

Different patterns in Algorithm



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

Recursive

The functions computed by the algorithms are expressed in terms of itself

Example 1 (Sequential)

- Task: Find the Area of a Circle

  Algorithm:

  ```
  BEGIN
  Stepl : Accept the RADIUS
  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

  END
  ```

Example 2(Sequential)

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

```
BEGIN

Step 1 : Accept Marks1, Marks2, Marks3
Step 2 : Sum = Marks1 + Marks2 + Marks3
Step 3 : Average = Sum / 3
Step 4 : Display Average


END
```

Example 1 - Selectional

- 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.

```
BEGIN
Step 1 : Accept Marks'!, Marks2, Marks3
Step 2 : Total = Marks'! + Marks2 + Marks3
Step 3 : Average = Total / 3
Step 4 : Set Output = “Student Passed”
Step 5 : if Average < 65 then Set Output = “Student Failed"
Step 6 : Display Output
END
```

Example 2 - Selectional

- Task:Find whether a given year is a leap year or not

  Algorithm:

  ```
  BEGIN
  Stepl : Acceptthe YEAR
  Step 2 : IF ({YEAR%4=0 AND YEAR%100!=0)OR (YEAR%400=0))
  Display "Year is a leap year"
  ELSE
  Display “Year is not a leap year"
  ENDIF
  END
  ```

  For the logic in the IF condition refer Truth Table for Leap year problem

Example - Iterational

- Find the average marks scored by ‘N’ number of students

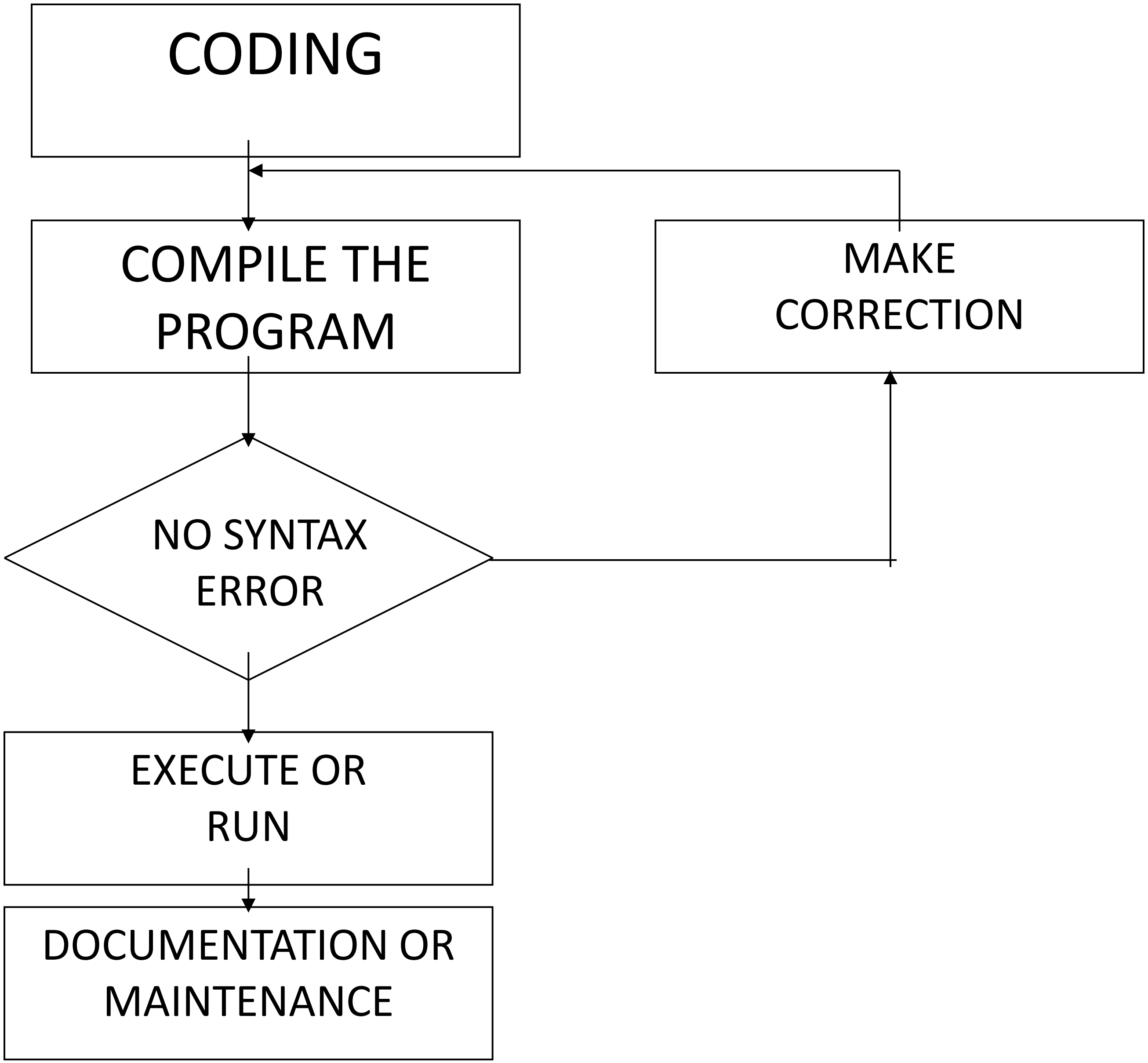
```
BEGIN
Step 1 : Accept Number Of Students
Step 2 : Counter = 1
Step 3 : Read Marks1, Marks2, Marks3
Step 4 : Total = Marks1 + Marks2 + Marks3
Step 5 : Average = Total / 3
Step 6 : Set Output = “Student Passed”
Step 7 : If (Average < 65) then Set Output = “Student Failed"
Step 8 : Display Output
Step 9 : Counter = Counter + 1
Step 10 : If (Counter <= NumberOfStudents ) then goto step 3
END
```

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

COMPILE THE PROGRAM

MAKE CORRECTION

NO SYNTAX ERROR

EXECUTE OR RUN

DOCUMENTATION OR MAINTENANCE

Programming Or Implementation Phase

- **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.
  - Many types or levels of computer programming language such as:
    - Machine language
    - Symbolic language or assembly language
    - Procedure-oriented language
  - The first two languages are also called *low-level programming language*. While the last one is called *high-level programming language*.

Programming Or Implementation Phase

- **Machine Language**
  - Machine language uses number to represent letters, alphabets or special character that are used to represent bit pattern.
  - Example:
    - an instruction to add regular pay to overtime pay, yielding total pay might be written in machine language as follows:

      16 128 64 8

    - in which 16 is a code that mean ADD to the computer. The 128 and 64 are addresses or location at which regular pay and overtime pay are stored. The 8 represents the storage location for the total pay.

Programming Or Implementation Phase

- Sometimes, bit pattern that represent letters and alphabets are used for coding.
  - Example:

Instead of:	16	128	64	8
Use:	10000	10000000	1000000	1000

  - This representation is ideal for a computer but difficult and tedious to the programmer to write a lengthy program.

Programming Or Implementation Phase

- **Symbolic Language or Assembly Language**
  - A symbolic language or assembly language is closely related to machine language in that, one symbolic instruction will translate into one machine-language instruction.
  - Contain fewer symbols, and these symbols may be letters and special characters, as well as numbers.
  - As example, a machine language instruction

    16 128 64 8

    can be rewritten in assembly language as

    ADD LOC1 LOC2 LOC3

  - Which means, add content of location LOC1 to location LOC2 and put the result in location LOC3.

Programming Or Implementation Phase

- **Procedure – Oriented Language**
  - Programmer has to know the computer hardware before he can write program in machine and assembly language. It means the language is machine dependent.
  - Using procedure – oriented language, the programmer can run the program in any computer hardware.
  - A special program called a *compiler* will translate program written using procedure – oriented language to machine language.

Programming Or Implementation Phase

- Some example of the language:
  - COBOL (COmmon Business Oriented Language)
  - FORTRAN (FORmula TRANslation)
  - Pascal
  - C
  - C++
  - BASIC, etc.
- These languages are also called *high-level programming language*

Programming Or Implementation Phase

Computer Language	Instruction Format
Machine language	16 128 64 8
Assembly language	ADD LOC1 LOC2 LOC3
BASIC	LET T = R + 0
FORTRAN	TOTAL = RPAY + OPAY
COBOL	ADD RPAY, OPAY GIVING TOTAL
Pascal	TOTAL : = RPAY + OPAY
C	TOTAL = RPAY + OPAY

Programming Or Implementation Phase

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

Programming Or Implementation Phase

- **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.

Programming Or Implementation Phase

- **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.

- Next 8 slides are for faculty to imbibe, hints on how to solve teach problem solving

Approach to Teaching Problem Solving – Faculty Hints

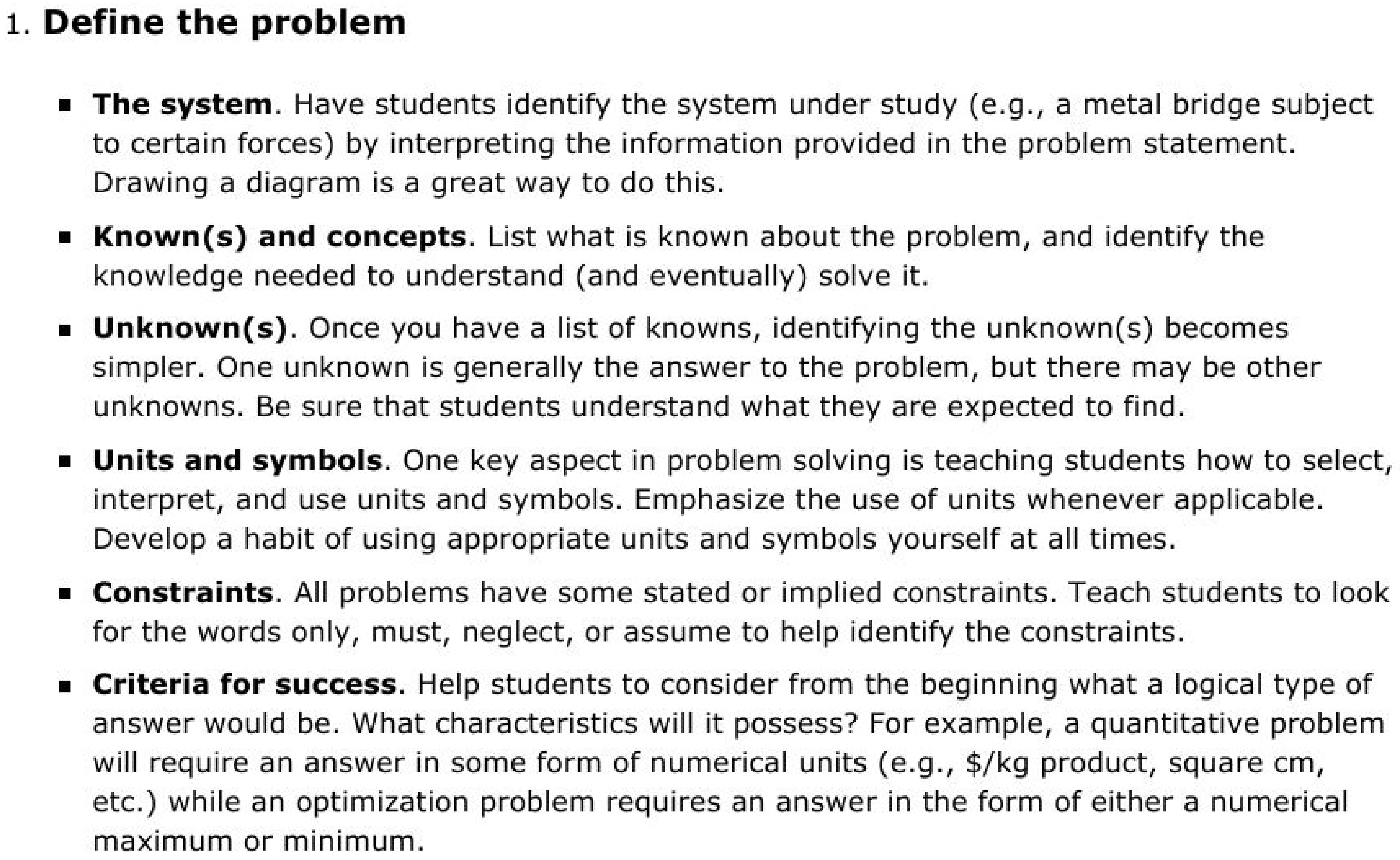
- Model a useful problem-solving method -
  - Show students by your example how to be patient and persistent and how to follow a structured method
- Teach within a specific context
  - Teach problem-solving skills in the context in which they will be used (e.g., mole fraction calculations in a chemistry course). Use real-life problems in explanations, examples, and exams. Do not teach problem solving as an independent, abstract skill.
- Help students understand the problem.
  - In order to solve problems, students need to define the end goal. This step is crucial to successful learning of problem-solving skills. If you succeed at helping students answer the questions “what?” and “why?”, finding the answer to “how?” will be easier.

Approach to Teaching Problem Solving – Faculty Hints

- Take enough time.
  - When planning a lecture/tutorial, budget enough time for: understanding the problem and defining the goal, both individually and as a class; dealing with questions from you and your students; making, finding, and fixing mistakes; and solving entire problems in a single session

- **Ask questions and make suggestions**
  - Ask students to predict “what would happen if ...” or explain why something happened. This will help them to develop analytical and deductive thinking skills. Also, ask questions and make suggestions about strategies to encourage students to reflect on the problem-solving strategies that they use
- **Link errors to misconceptions**
  - Use errors as evidence of misconceptions, not

Woods’ problem-solving model



1. **Define the problem**

- **The system**. Have students identify the system under study (e.g., a metal bridge subject to certain forces) by interpreting the information provided in the problem statement. Drawing a diagram is a great way to do this.
- **Known(s) and concepts**. List what is known about the problem, and identify the knowledge needed to understand (and eventually) solve it.
- **Unknown(s)**. Once you have a list of knowns, identifying the unknown(s) becomes simpler. One unknown is generally the answer to the problem, but there may be other unknowns. Be sure that students understand what they are expected to find.
- **Units and symbols**. One key aspect in problem solving is teaching students how to select, interpret, and use units and symbols. Emphasize the use of units whenever applicable. Develop a habit of using appropriate units and symbols yourself at all times.
- **Constraints**. All problems have some stated or implied constraints. Teach students to look for the words only, must, neglect, or assume to help identify the constraints.
- **Criteria for success**. Help students to consider from the beginning what a logical type of answer would be. What characteristics will it possess? For example, a quantitative problem will require an answer in some form of numerical units (e.g., $/kg product, square cm, etc.) while an optimization problem requires an answer in the form of either a numerical maximum or minimum.

Woods’ problem-solving model

- **“Let it simmer”**.
- Use this stage to ponder the problem. Ideally, students will develop a mental image of the problem at hand during this stage.
- Identify specific pieces of knowledge. Students need to determine by themselves the required background knowledge from illustrations, examples and problems covered in the course.
- Collect information. Encourage students to collect pertinent information such as

Woods’ problem-solving model ..

- **Plan a solution**
- Consider possible strategies. Often, the type of solution will be determined by the type of problem. Some common problem-solving strategies are: compute; simplify; use an equation; make a model, diagram, table, or chart; or work backwards.
- Choose the best strategy. Help students to choose the best strategy by reminding them again what they are required to find or calculate

Woods’ problem-solving model..

- **Carry out the plan**
- Be patient. Most problems are not solved quickly or on the first attempt. In other cases, executing the solution may be the easiest step.
- Be persistent. If a plan does not work immediately, do not let students get discouraged. Encourage them to try a different strategy and keep trying.

Woods' problem-solving model ..

- **Look back**
- Encourage students to reflect. Once a solution has been reached, students should ask themselves the following questions:
    - Does the answer make sense?
    - Does it fit with the criteria established in step 1?
    - Did I answer the question(s)?
    - What did I learn by doing this?
    - Could I have done the problem another way?

Stuff to look at

- Problem Solving Basics and Computer Programming, by By Ronald A. Pasko
- http://www.cs.iit.edu/~cs100/ProblemSolving.pdf

Problem 1

- Given the 3 dimensions of a box (length, width, and height), multiply them together to determine the volume

Steps to Program Solving and Program Development

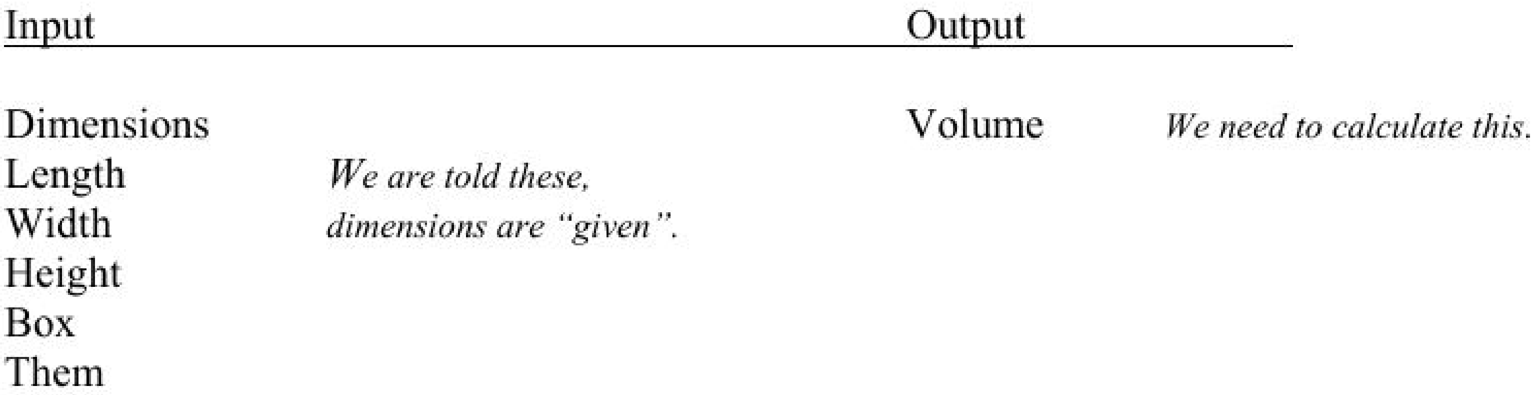
- Decomposition
- Flowchart
- Pseudocode
- Code

1. Decomposition

- Decompose the problem description
- Eg. perform syntactic analysis on the description – in 4 steps
- Identify all of the nouns in the sentence
    - Given the 3 dimensions of a box (length, width, and height), calculate the volume.

2.Segregate Input - Output

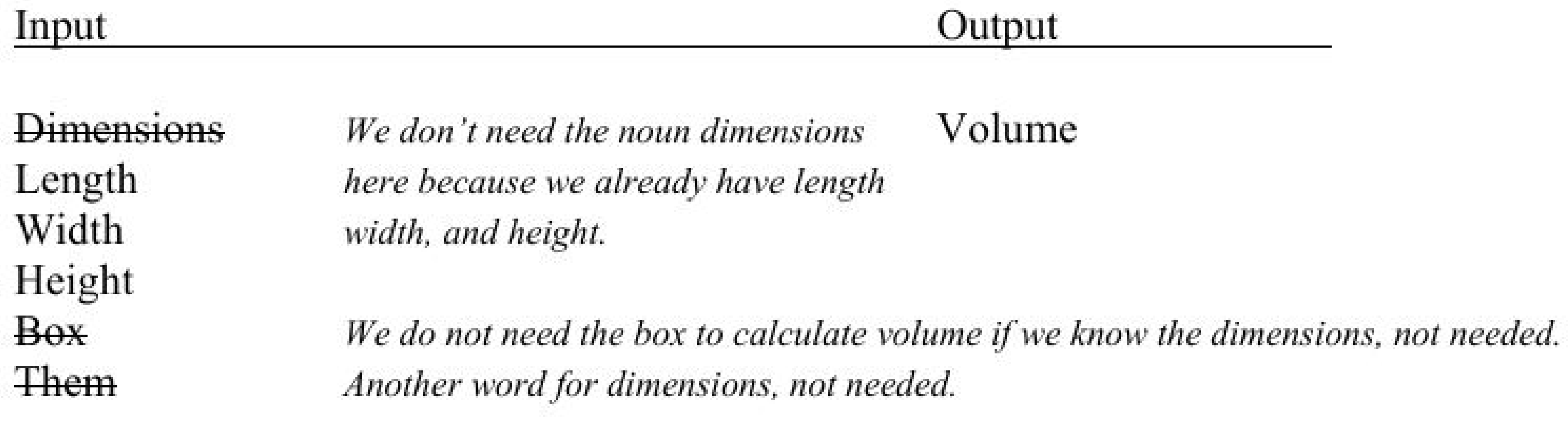
- Once these nouns are identified, they should be grouped into one of two categories:
- Input (items I either already know or am getting from the user)
- Output (items that I find out by manipulating the input)



Input		Output	
Dimensions		Volume	*We need to calculate this.*
Length	*We are told these,*		
Width	*dimensions are "given".*		
Height			
Box			
Them			

3. Eliminate redundant information

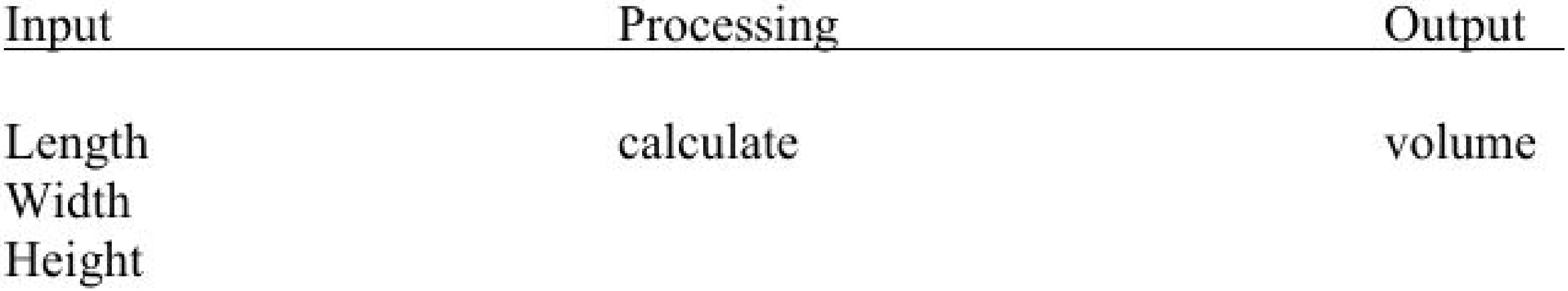
- There may be some information in the problem description that made it into our input/output chart that we really don’t need to solve the problem (that is, not all of thenouns may be relevant)
- Eliminate the most general item



Input		Output
~~Dimensions~~	*We don’t need the noun dimensions*	Volume
Length	*here because we already have length*	
Width	*width, and height.*	
Height		
~~Box~~	*We do not need the box to calculate volume if we know the dimensions, not needed.*	
~~Them~~	*Another word for dimensions, not needed.*	

4. Process

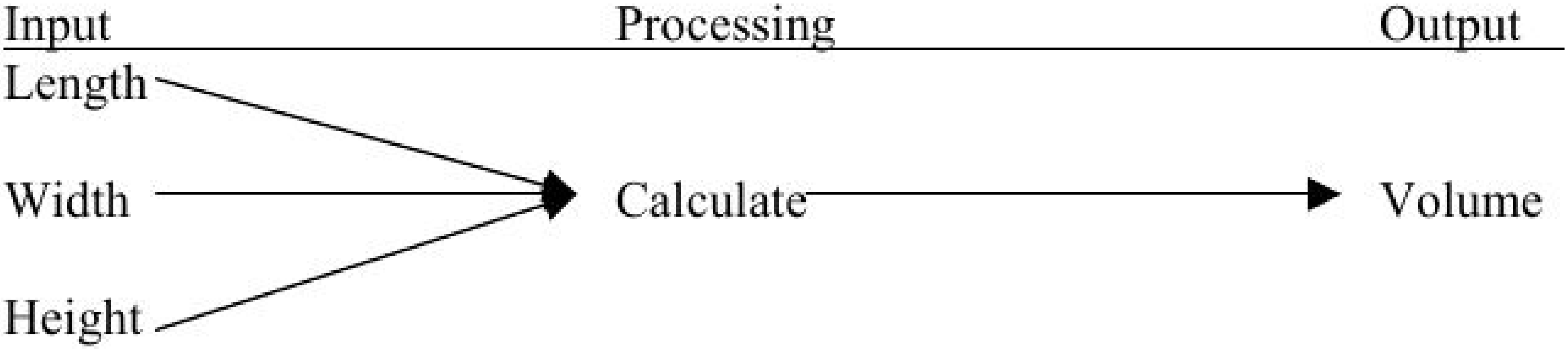
- Identify all of the verbs in the sentence
- The verbs in the problem specification identify what actions your program will need to take
- These actions, known as processing are the steps between your input and your output



Input	Processing	Output
Length Width Height	calculate	volume

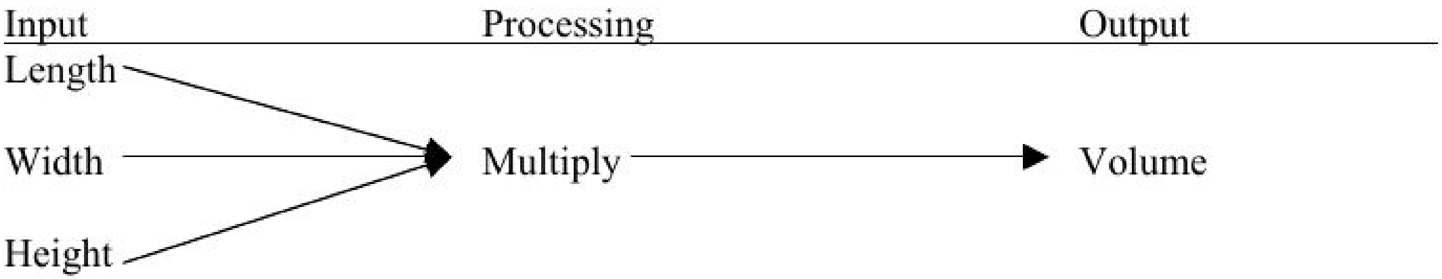
5. Processing for Output

- Link you inputs, processes, and output
- This step is as simple as drawing lines between the relevant information in your chart. Your lines show what inputs need to be processed to get the desired output
- Take our length, width, and height and multiply them, to give us our desired volume



5. Processing

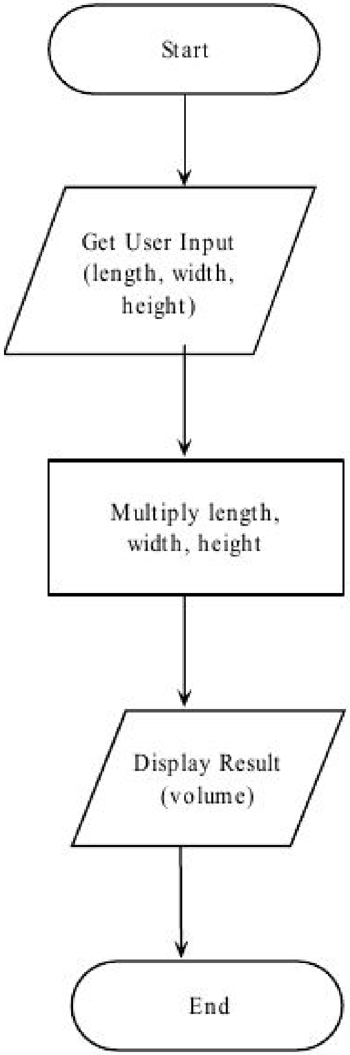
- Calculate could refer to applying some mathematical formula or other transformation to our input data in order to reach the desired output
- Here calculate stand for
- Volume = length * width * height



6. Flowcharting

- Next step is to perform flowcharting
- Flowcharting is a graphical way of depicting a problem in terms of its inputs, outputs, and processes
- Here take in three items as input (length, width, and height). And after we have the user’s input, need to process it. In this case, we must multiply the dimensions together

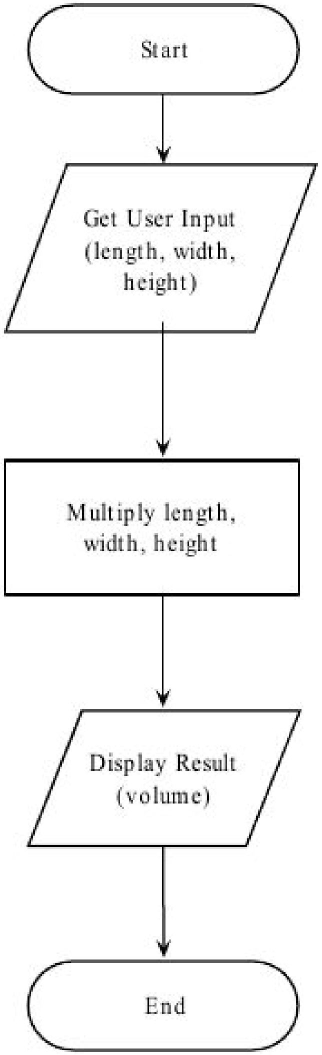
6. Flowchart for Volume


Start
Get User Input (length, width, height)
Multiply length, width, height
Display Result (volume)
End

7. Pseudocode

- Move from flowchart to pseudocode
- Pseudocode involves writing down all of the major steps you will use in the program as depicted in your flowchart

7. Pseudocode


Start
Get User Input (length, width, height)
Multiply length, width, height
Display Result (volume)
End

```
Get length, width, height
Compute volume
        volume = length * width * height
        Store volume
Display volume
```

4 Step Process

- Decomposition,
- Flowcharting, and
- Pseudocode
- Code in any programming language

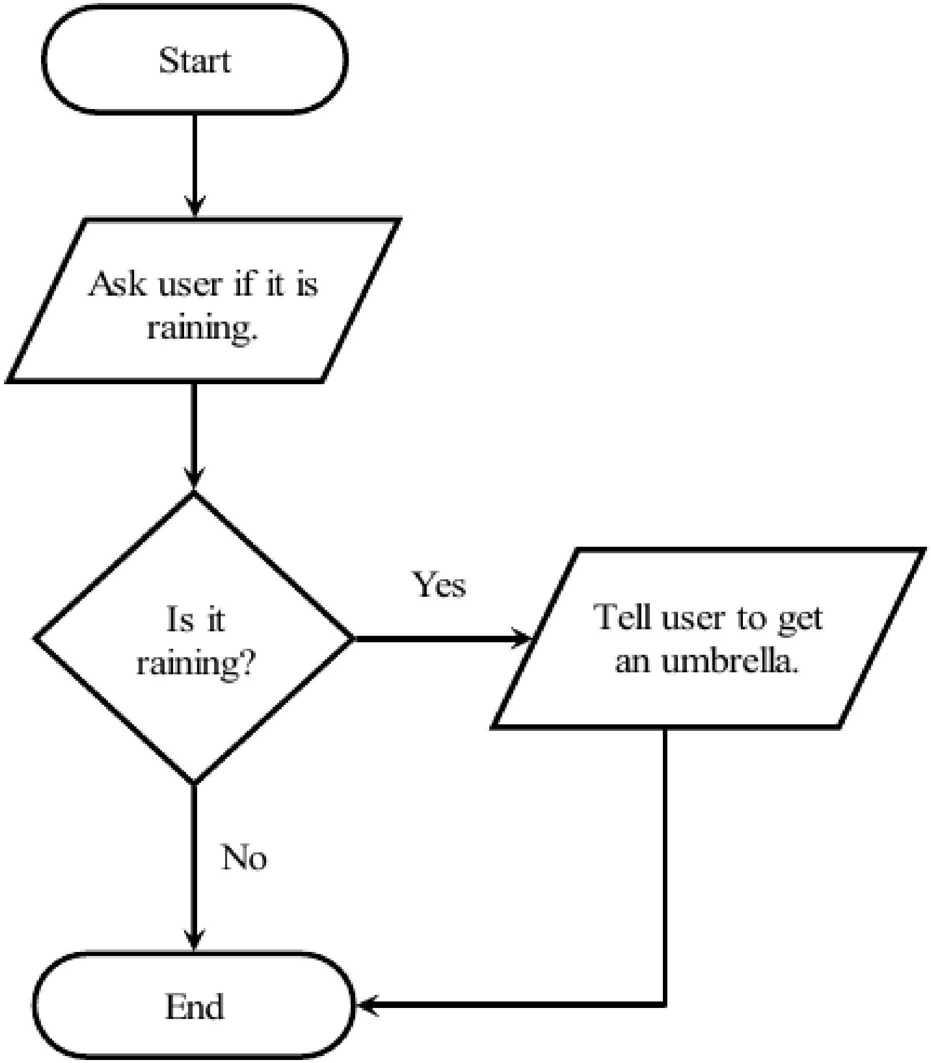
Try out -Problem 2

- You have a store that sells lemons and oranges. Oranges are $.30 each and lemons are $.15 each. Your program should get from the user the numbers of oranges and lemons he/she wants and outputs the total amount of money they owe you.

Problem 3

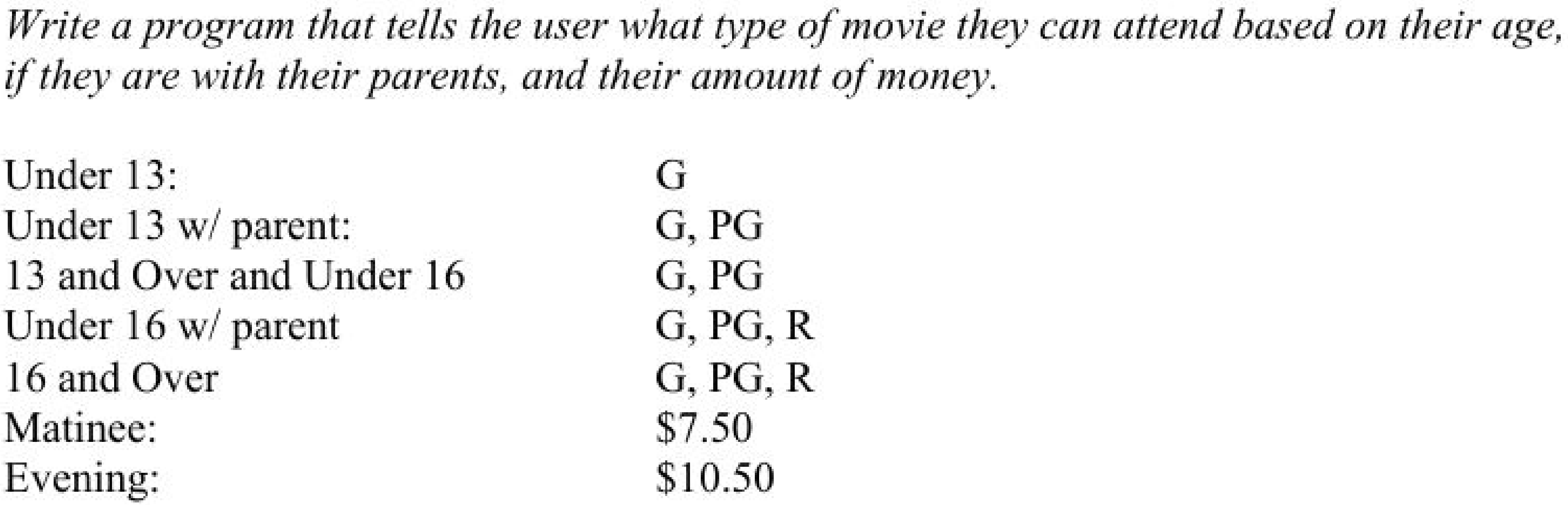
- Write a program that will accept as input from the user, an answer to the following question: Is it raining? If it is raining, tell the user to get an umbrella

Problem 3



```
Ask user if it is raining
Get answer
If answer is yes
        Display "Get an umbrella"
```

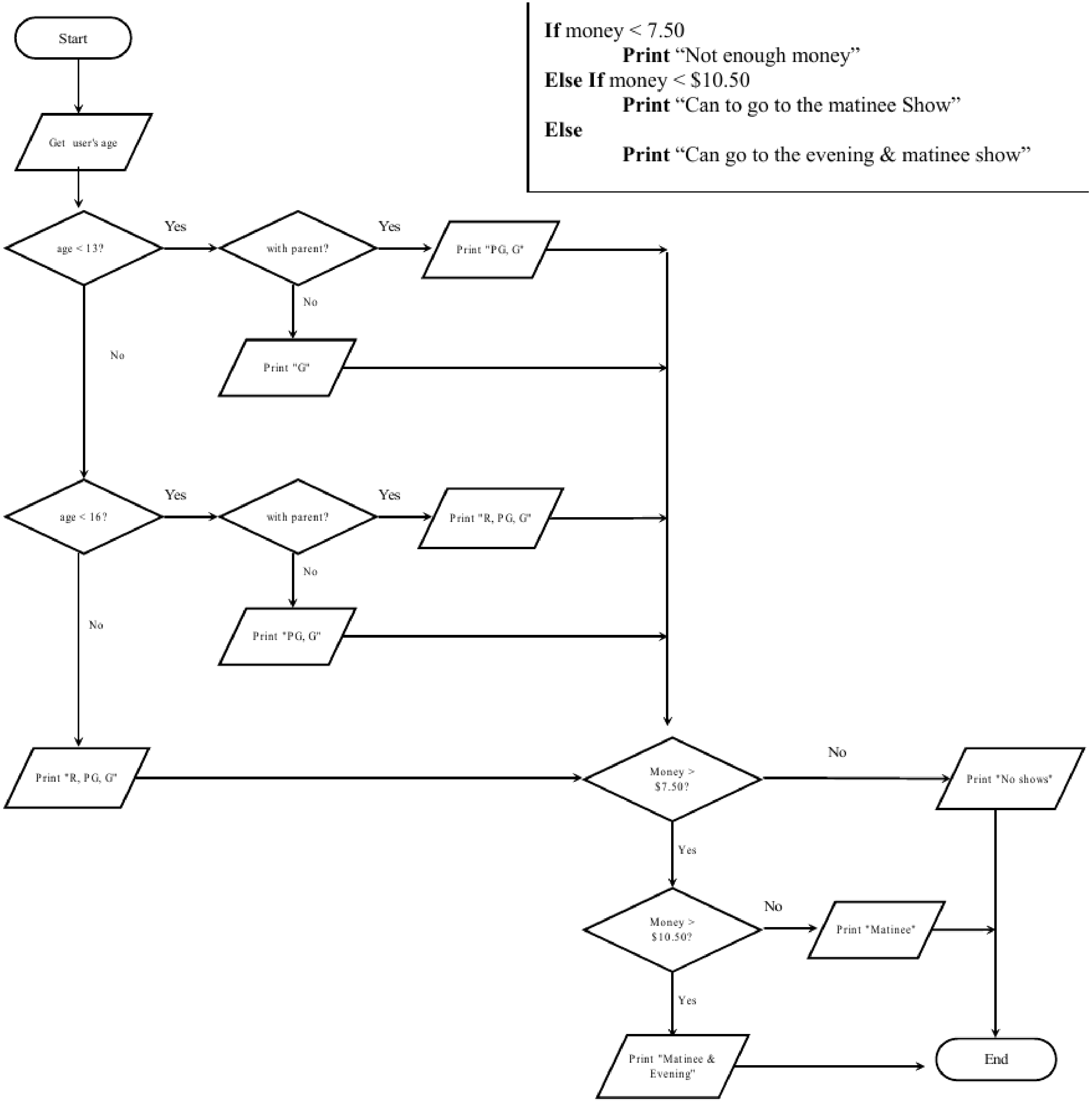
Problem 4



Write a program that tells the user what type of movie they can attend based on their age, if they are with their parents, and their amount of money.

Under 13:	G
Under 13 w/ parent:	G, PG
13 and Over and Under 16	G, PG
Under 16 w/ parent	G, PG, R
16 and Over	G, PG, R
Matinee:	$7.50
Evening:	$10.50

```
If money < 7.50
        Print "Not enough money"
Else If money < $10.50
        Print "Can to go to the matinee Show"
Else
        Print "Can go to the evening & matinee show"
```



```
If age < 13
        If with parent
                Print “Can go to G & PG show’
        Else
                Print “Can to go G show”
Else If age < 16
        If with parent
                Print “Can go to G, PG & R show’
        Else
                Print “Can to go PG & G show”
Else
        Print “Can go to G, PG, & R show”

If money < 7.50
        Print “Not enough money”
Else If money < $10.50
        Print “Can to go to the matinee Show”
Else
        Print “Can go to the evening & matinee show”
```

Next - Program

- Try, A First Program—Calculating the Drake Equation in the book “Introduction to Computer Science Using Python”   by Charles Dierbach at the link below at Page 29 which has test cases use also.
- https://www.ebooks-it.net/ebook/introduction-to-computer-science-using-python