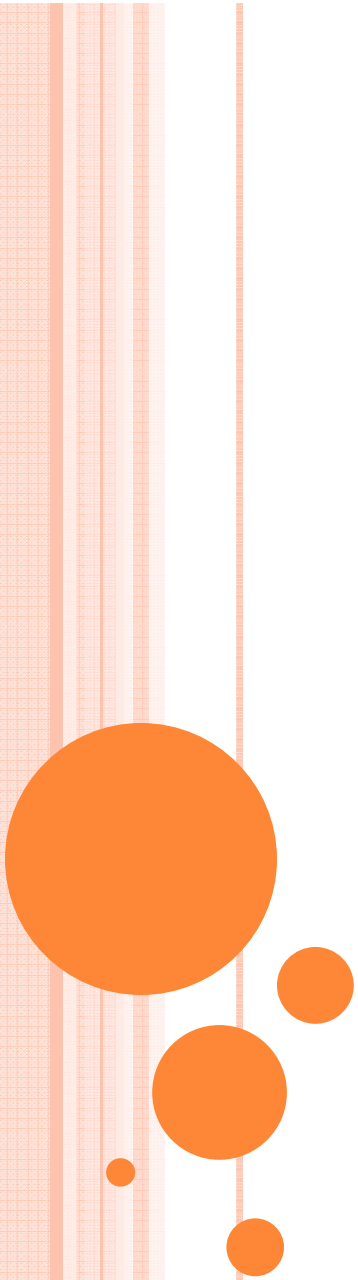


Understanding Problems and how to Solve them by using Computers

INTRODUCTION TO PROBLEM SOLVING

- Introduction to Problem Solving
- Understanding problems
- Data processing
- Writing an algorithm

CONTINUE..

Tool to solve problem :

- Software development method (SDM)
  - Specification of needs
  - Problem analysis
  - Design and algorithmic representation
  - Implementation
  - Testing and verification
  - Documentation

CONTINUE..

- What is problem solving?
- Problem solving is the process of transforming the description of a problem into a solution of that problem by using our knowledge of the problem domain and by relying on our ability to select and use appropriate problem-solving strategies, techniques and tools.

- Computers can be used to help us **solving problems**

Software development method

1. Specification of needs/requirements specification
2. Problem analysis
3. Design and algorithmic representation
4. Implementation
5. Testing and verification
6. Documentation

Specification of needs

- To understand exactly:
  - what the problem is
  - what is needed to solve it
  - what the solution should provide
  - if there are constraints and special conditions.

Problem Analysis

- In the analysis phase, we should identify the following:
  - Inputs to the problem, their form and the input media to be used
  - Outputs expected from the problem, their form and the output media to be used
  - Special constraints or conditions (if any)
  - Formulas or equations to be used

Design and Algorithmic Representation

- An algorithm is a sequence of a finite number of steps arranged in a specific logical order which, when executed, produces the solution for a problem.
- An algorithm must satisfy these requirements:
  - It must have an **input**
  - It must have an **output**
  - It should not be **ambiguous**-it must be clear on what to do and how many to executed (there should not be different interpretations to it)
    - Every step in algorithm must be clear as what it is supposed to do

Design and Algorithmic Representation Cont…

- It must be **general** (it can be used for different inputs)
- It must be **correct** and it must solve the problem for which it is designed
- It must execute and terminate in a **finite** amount of time
- It must be **efficient** enough so that it can solve the intended problem using the resource currently available on the computer

- An algorithm can be represented using pseudocodes or flowcharts.

PSEUDOCODES

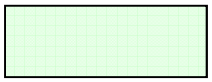
- A pseudocode is a semiformal, English-like language with limited vocabulary that can be used to design and describe algorithms.
- Criteria of a good pseudocode:
  - Easy to understand, precise and clear
  - Gives the correct solution in all cases
  - Eventually ends

FLOWCHARTS

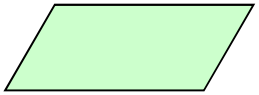
- Flowcharts is a graph used to depict or show a step by step solution using **symbols** which represent a task.
- The symbols used consist of geometrical shapes that are connected by **flow lines**.
- It is an alternative to pseudocoding; whereas a pseudocode description is verbal, a flowchart is graphical in nature.

SYMBOLS

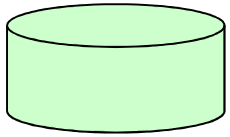
Terminal symbol - indicates the beginning and end points of an algorithm.



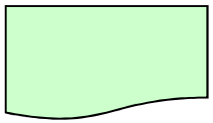
Process symbol - shows an instruction other than input, output or selection.



Input-output symbol - shows an input or an output operation.

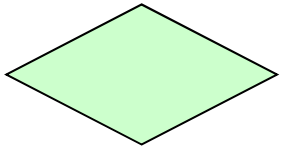


Disk storage I/O symbol - indicates input from or output to disk storage.

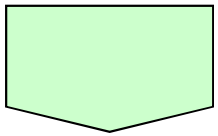


Printer output symbol - shows hardcopy printer output.

SYMBOLS CONT…

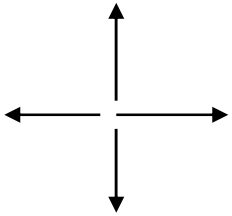


Selection symbol - shows a selection process for two-way selection.



Off-page connector - provides continuation of a logical path on another page.

On-page connector - provides continuation of logical path at another point in the same page.



Flow lines - indicate the logical sequence of execution steps in the algorithm.

EXAMPLE 1:

- Write a pseudo code and a flowchart to determine a student's final grade and indicate whether it is passing or failing. The final grade is calculated as the average of four marks.

EXAMPLE 1:

Pseudocode:

- *Input a set of 4 marks*
- *Calculate their average by summing and dividing by 4*
- *if average is below 50*

  Print "FAIL"

  else

  Print "PASS"

EXAMPLE 1:

- Detailed Algorithm
- Step 1: Input M1,M2,M3,M4

  Step 2: GRADE ← (M1+M2+M3+M4)/4

  Step 3: if (GRADE < 50) then

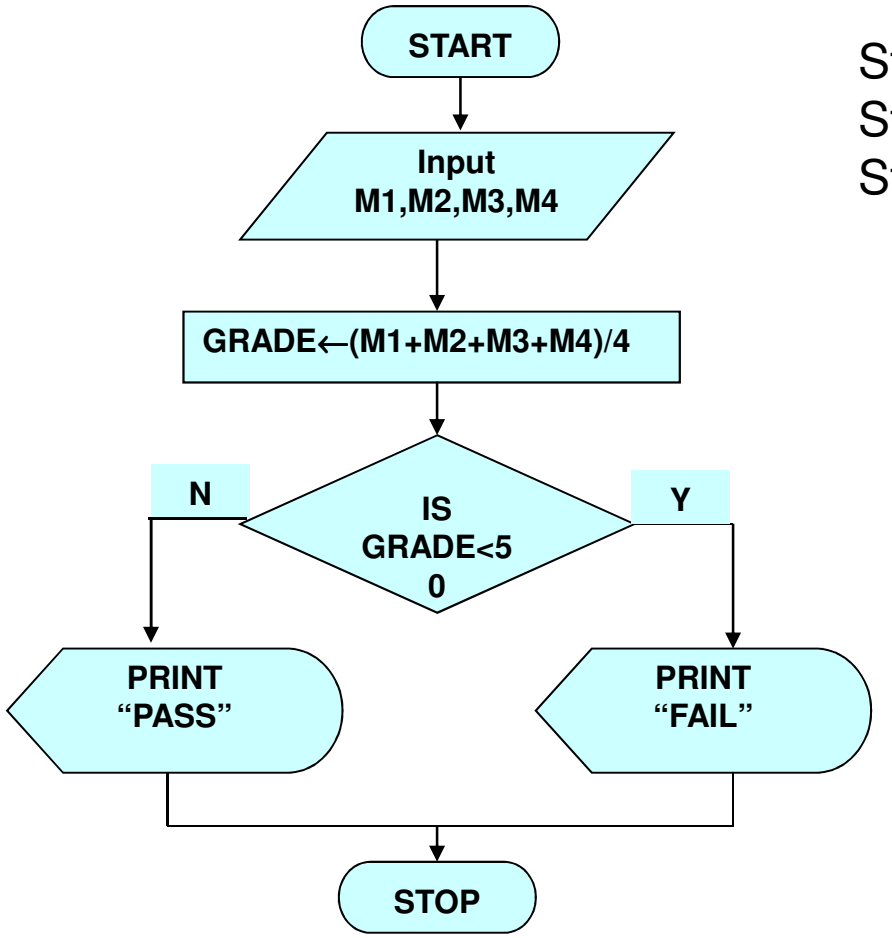
  Print “FAIL”

  else

  Print “PASS”

  endif

EXAMPLE 1: FLOWCHART



Step 1: Input M1,M2,M3,M4
Step 2: GRADE ← (M1+M2+M3+M4)/4
Step 3: if (GRADE <50) then
Print “FAIL”
else
Print “PASS”
endif

EXAMPLE 2

- Write an algorithm and draw a flowchart to convert the length in feet to centimeter.

Pseudocode:

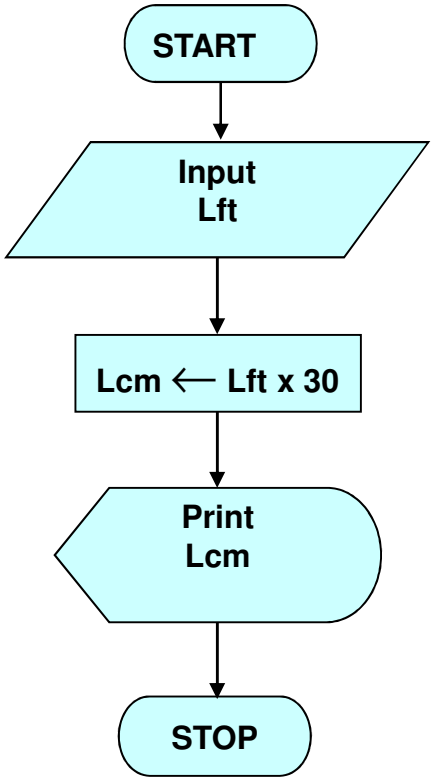
- *Input the length in feet (Lft)*
- *Calculate the length in cm (Lcm) by multiplying LFT with 30*
- *Print length in cm (LCM)*

EXAMPLE 2

Algorithm

- Step 1: Input Lft
- Step 2: Lcm ← Lft x 30
- Step 3: Print Lcm

Flowchart



EXAMPLE 3

Write an algorithm and draw a flowchart that will read the two sides of a rectangle and calculate its area.

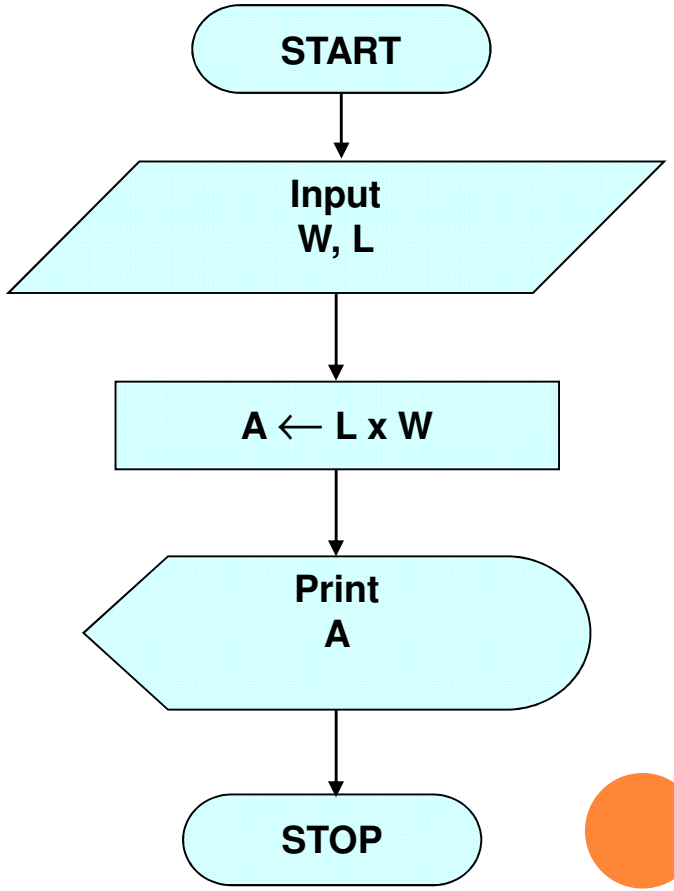
Pseudocode

- *Input the width (W) and Length (L) of a rectangle*
- *Calculate the area (A) by multiplying L with W*
- *Print A*

EXAMPLE 3

Algorithm

- Step 1: Input W,L
- Step 2: A ← L x W
- Step 3: Print A



EXAMPLE 4

- Write an algorithm and draw a flowchart that will calculate the roots of a quadratic equation $ax^2 + bx + c = 0$

- Hint: **d** = sqrt ($b^2 - 4ac$), and the roots are: ***x*1** = $(-b + d)/2a$ and ***x*2** = $(-b - d)/2a$

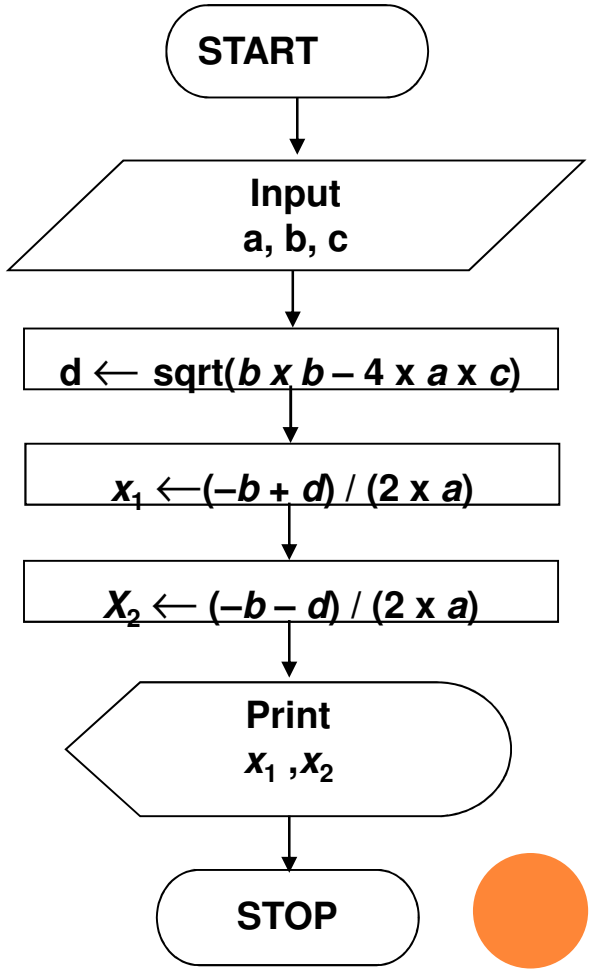
EXAMPLE 4

Pseudocode:

- *Input the coefficients (a, b, c) of the quadratic equation*
- *Calculate* **d**
- *Calculate* ***x*1**
- *Calculate* **x2**
- *Print* x*1 and* x2

EXAMPLE 4

- **Algorithm:**
  - Step 1: Input a, b, c
  - Step 2: $d \leftarrow \text{sqrt}\ (b \times b - 4 \times a \times c\ )$
  - Step 3: $x1 \leftarrow (-b + d) / (2 \times a)$
  - Step 4: $x2 \leftarrow (-b - d) / (2 \times a)$
  - Step 5: Print $x1$, $x2$



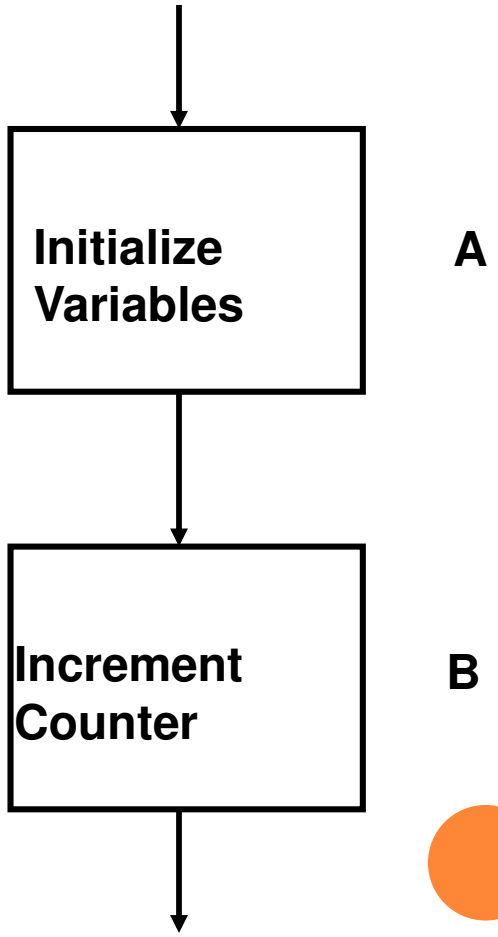
CONTROL STRUCTURE

- In order to tackle a problem, we need
  - a correct algorithm
  - to apply the algorithm at the 'good' moment
  - to decide which algorithm to apply (sometimes there are more than one, depending on conditions)
  - to know if a certain operation must be repeated

  →In short: we need a suitable ***Control Structure***
- In 1966, two researchers, C. Bohn and G. Jacopini, demonstrated that **any algorithm** can be described using only **3 control structures**: **sequence**, **selection** and **repetition**.

Sequence Construct

- Single steps or actions in the program logic
- Statements executed in the order of appear-ance, with control passing unconditionally from one statement to the next.
  - The program executes Statement A followed by Statement B.



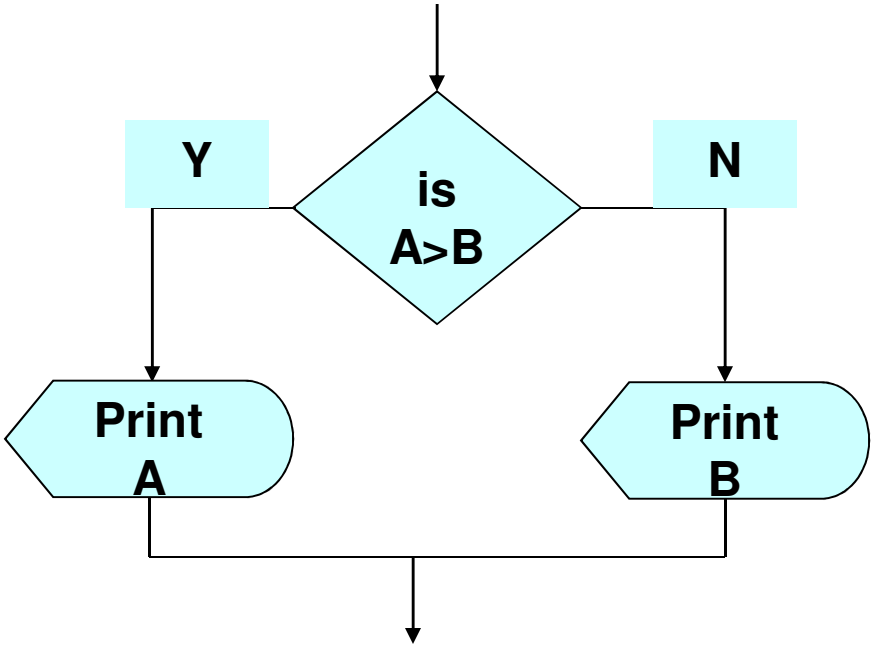
Selection Construct

- A decision point in a procedure in which the outcome of a stated condition determines which of two actions will be taken.
- Tests a condition and executes one of the two alternative instruction sets based on the results of the test.

SELECTION STRUCTURE

- The expression A>B is a logical expression
- *it describes a* ***condition*** *we want to test*
- ***if A>B is true (if A is greater than B)*** *we take the action on left*
- print the value of A
- ***if A>B is false (if A is not greater than B)*** *we take the action on right*
- print the value of B

SELECTION STRUCTURE



Selection Construct Continued

IF Hours is greater than 40

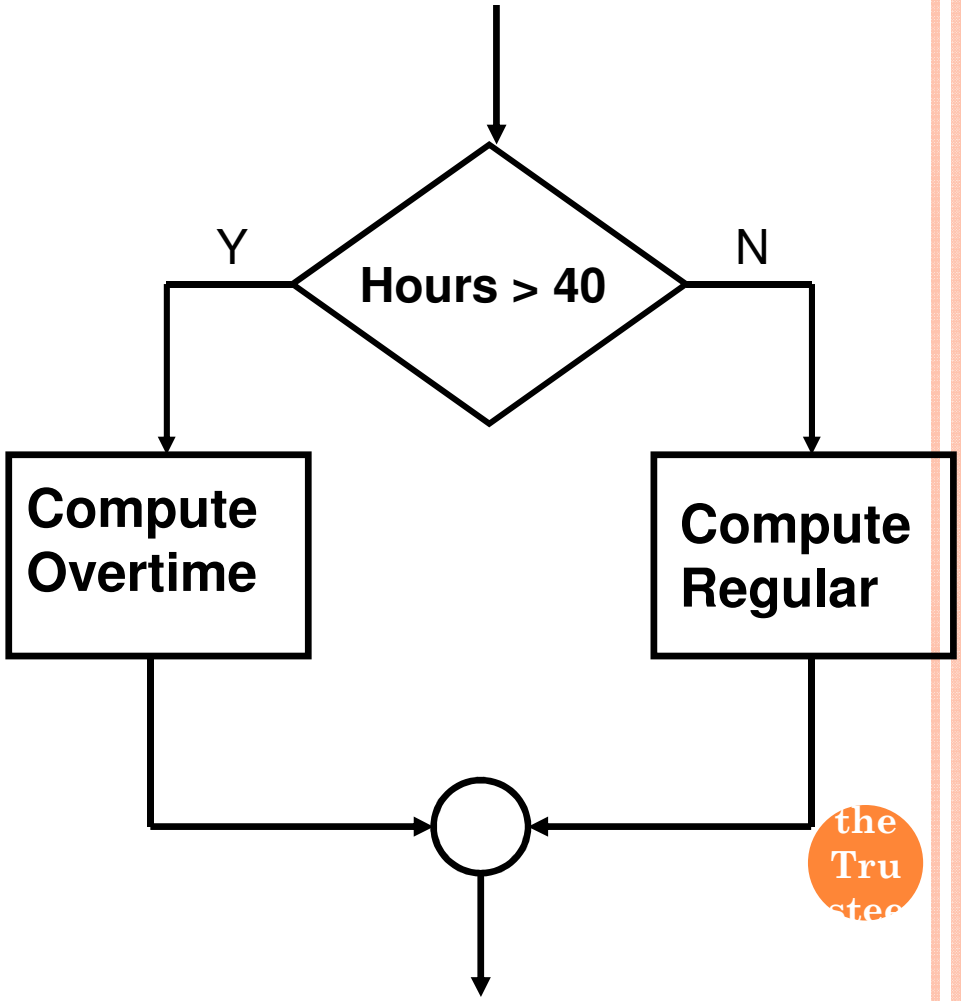
THEN

Compute Overtime Pay

ELSE

Compute Regular Pay

ENDIF



Iteration Construct

- The logic pattern in programming in which certain actions are repeated whenever a specified condition occurs.

- The cycle repeats until such time as the specified condition no longer occurs

REPETITION/ITERATION STRUCTURE

- Specifies a block of one or more statements that are repeatedly executed until a condition is satisfied.
- The keyword used is *while*.
- Format:

```
while condition
      loop-body
end_while
```

Example 1: print statements "hello world " 2 times

```
Begin
set number equal to 1
while number is less than 3
    print hello world
       add 1 to number
   end_while
end
```

IMPLEMENTATION

- The process of ***implementing*** an algorithm by writing a computer program using a programming language (for example, using Python language)
- The output of the program **must** be the solution of the intended problem
- The program must **not** do anything that it is **not** supposed to do
  - *(Think of those many **viruses, buffer overflows, trojan horses,** etc. that we experience almost daily. All these result from programs **doing more** than they were intended to do)*

CONTINUE..

- Testing and verification
  - Program verification
    - Is the process of ensuring that a program meets user requirements
  - Program testing
    - Is the process of executing a program to demonstrate its correctness

- Documentation
  - Document what the program do
  - Should have the following elements:
    - A concise requirements specifications
    - Descriptions of problem inputs, expected outputs, constraints, and applicable formula
    - A psedocode/flowchart for its algorithm
    - A source program listing
  - Example - Comments in the program

PROBLEM SOLVING - EXAMPLE

- Consider for each of the following as requirements specifications for a problem. Complete the analysis and design steps of the software development. Indicate clearly the problem inputs, outputs, constraints and formulas.

EXAMPLE 1

- Problem

  Your summer job requires you to study some maps that give distances in kilometers and some that use miles. You and your coworkers prefer to deal in metric measurements.

- Analysis
  - Convert one system to another (convert miles to kilometers)
  - Problem input – distance in miles
  - Problem output – distance in kilometers
  - Relevant formula to convert miles to km
    - 1 mile = 1.609km

CONTINUE..

- Design
  - Algorithm
    - Get the distance in miles
    - Convert the distance to kilometers
    - Display the distance in kilometers

EXAMPLE 2

- Problem

  You need to create a converter that can convert the temperature from degrees Fahrenheit to degrees Celsius.

- Analysis
  - Convert temperature from Fahrenheit to Celsius
  - Program input – temperature in Fahrenheit
  - Program output – temperature in Celsius
  - Relevant formula – Celsius =5/9 (Fahrenheit – 32)
- Design
  - Algorithm
    - Get the input in Fahrenheit
    - Convert the temperature to Celsius
    - Display the temperature in Celsius

EXERCISE 1-SEQUENCE STRUCTURE

- Problem

  Get the radius of a circle. Compute and display the circle's area and circumference.

  PI=3.14159

Do the rest :

EXERCISE 2 – SEQUENCE STRUCTURE

- Problem

  An employee earns N 500.00 per hour for the first 40 hours and N800.00 per hour for each hour over 40. Design a program to calculate the daily wage of this employee

EXERCISE 3 – REPETITION STRUCTURE

- Problem

  Design a program that can print the statement " I love programming " 5 times.

EXERCISE 4-SEQUENCE AND REPETITION

- Problem

  Design a program that reads 5 numbers and computes and outputs their arithmetic average.