Problem Solving

(CS 1002)

Dr. Muhammad Aleem,

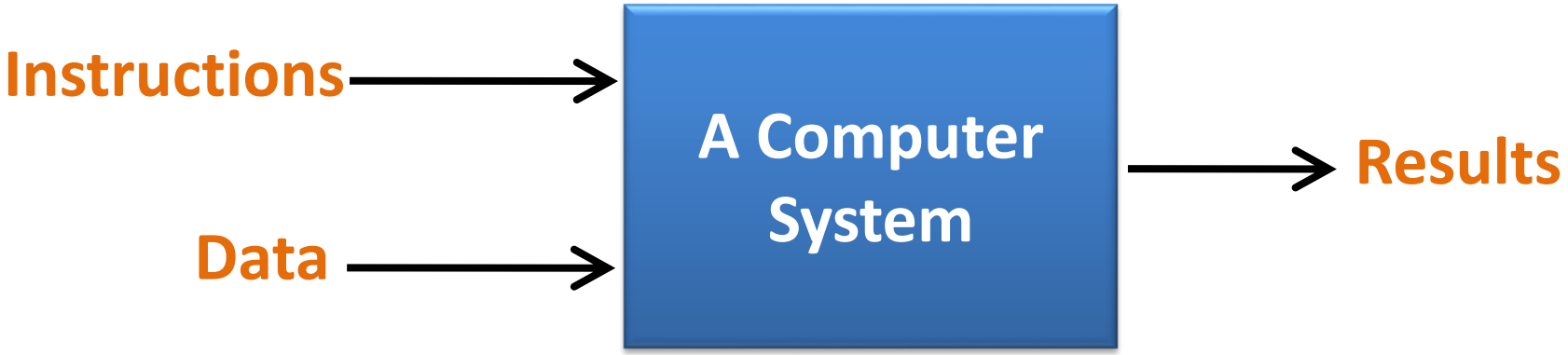
Department of Computer Science,

National University of Computer & Emerging Sciences,

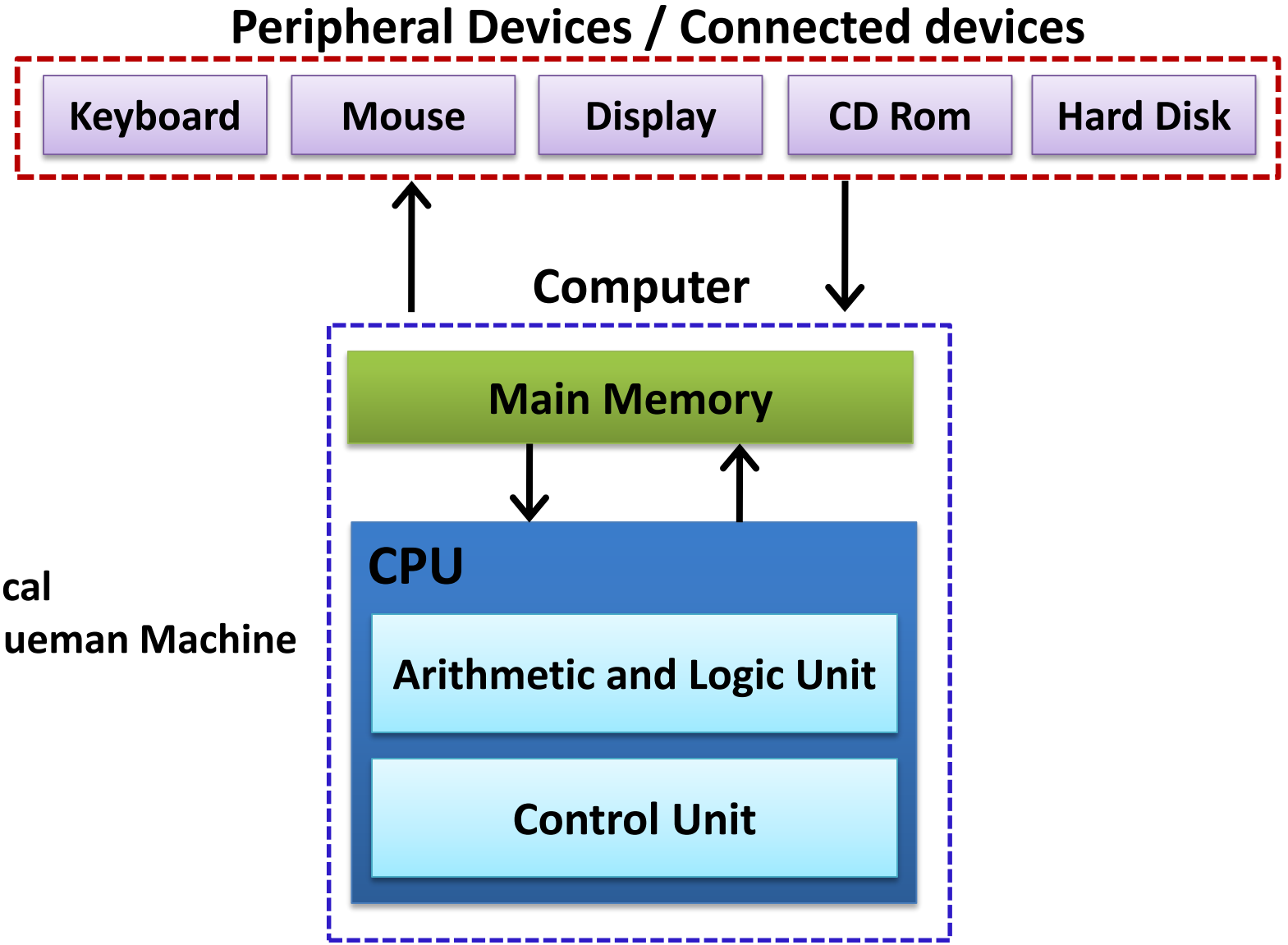
Islamabad Campus

What is a Computer?

- A computer is a **electro-mechanical device** that works **semi-automatically** to **process input data** according to the **stored set of instructions** and produces **output** or resultant data.



Components of a Computer System



A Typical
Von-Nueman Machine

Computer Instructions and Programs

- **Instruction**: A computer instruction is a command or directive given to a computer to perform specific task.

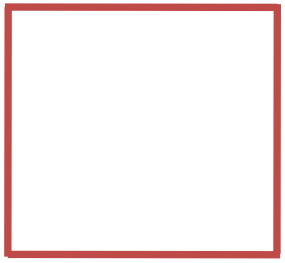
  Examples: Add 2 and 5, Print “Hello World”

- **Program**: A **program** is **sequence** of **instructions written** in **programming language** that directs a **computer** to **solve a problem**

  Examples: Draw a square, etc.

Computer Instructions and Programs

- **Program**: "**Draw a square**"



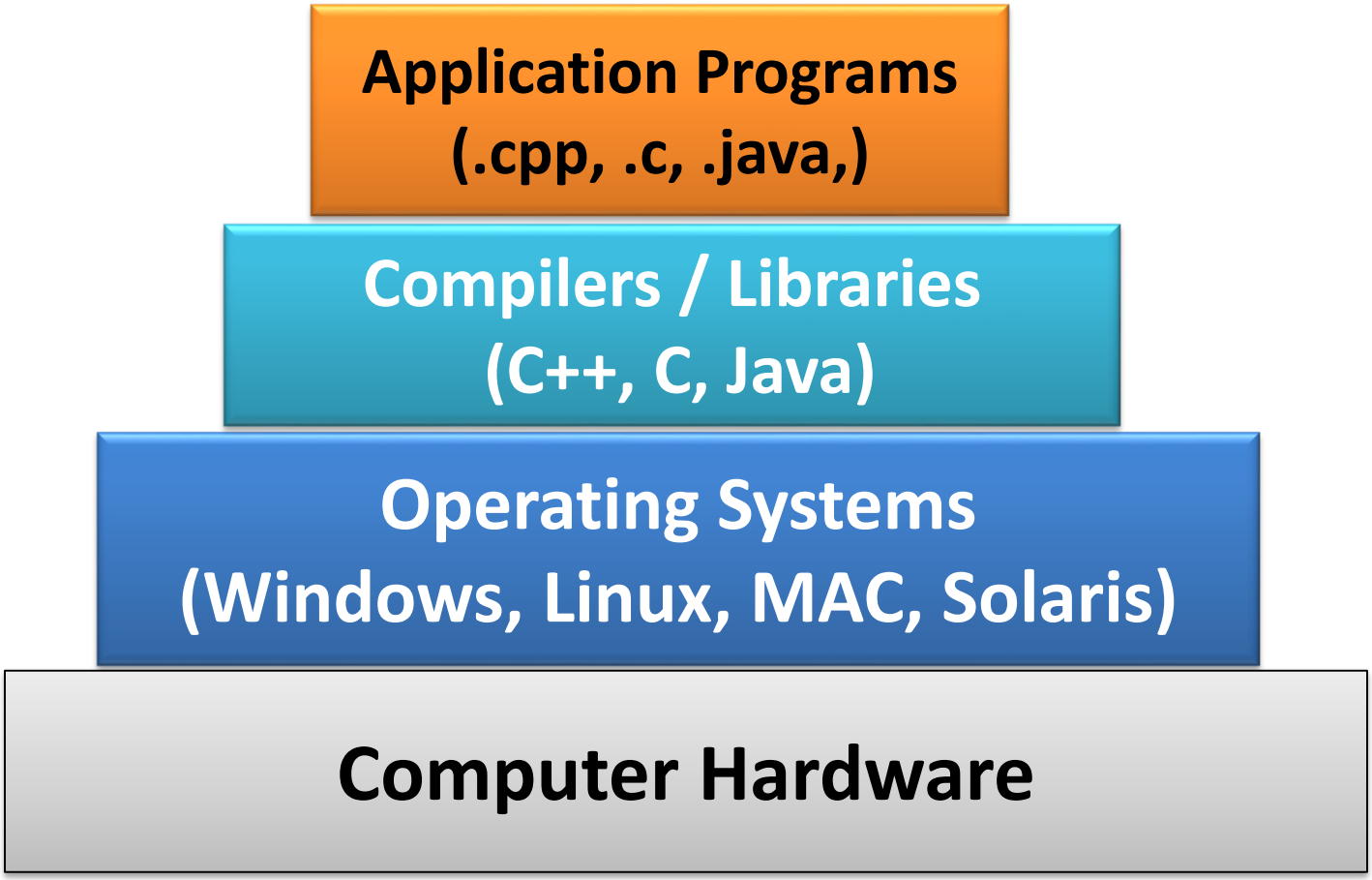
1 – Draw a vertical line of length *n* inches

2 – Draw a horizontal line of *n* inches

3– Draw a vertical line of length *n* inches

4 – Draw a horizontal line of *n* inches

Computer Software System



Programming Languages

Classification of programming languages:

1. Machine language
2. Low-level languages
3. High-level languages

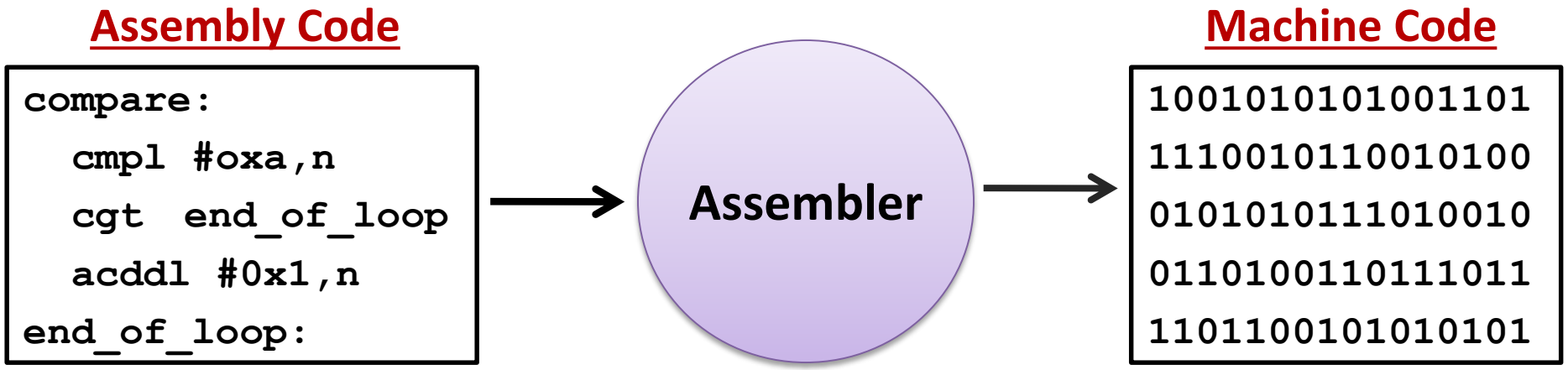
1. Machine level languages

- A **computer understands** only **sequence of bits or 1's and 0's** (the smallest piece of information)

- A **computer program** can be **written** using **machine languages** (**01001101010010010….**)
  - **Very fast execution**
  - **Very difficult** to **write** and **debug programs**
  - **Machine specific** *(different codes on different machines)*

2. Low level languages

- **English encrypted words** **instead of codes**
- **More understandable** (*for humans*)
- Example: **Assembly language**
- Requires: **"Translation"** from **Assembly** code to **machine code**



3. High level languages

- **Mostly machine independent**
- Close to **natural language** (English like keywords)
- **Easy to write** and **understand** programs
- **Easy to debug** and **maintain** code
- **Requires compilers** to **translate** to machine code
- **Slower** than **low-level languages**

3. High level languages

- Many popular High-Level languages

https://c1.staticflickr.com/4/3912/15090961835_c4f26e4890_b.jpg

Problem Solving Steps

1. **Understand** the problem
2. **Plan** the logic
3. **Code** the program
4. **Test** the program
5. **Deploy** the program into production

1. Understanding the Problem

- **Problems** are often **described** in **natural language like English**.

- **Identify the requirements**

  1. **Inputs** or given **data-items**
  2. Required **output(s)** or desired results
  3. **Indirect inputs** (may not be given directly, you have to calculate or assume)

1. Understanding the Problem

- Example: **Calculate** the **area of a circle** having the radius of 3 cm
  - Inputs:

    Radius=3
  - Output:

    Area
  - Indirect Inputs:

    Pi=3.14

Area = 3.14 * (3*3) = 28.27

2. Plan the Logic

- **Identify/Outline** small **steps** in sequence, to **achieve** the **goal** (or desired results)

- Tools such as ***flowcharts*** and ***pseudocode*** can be used:
  1. **Flowchart**: a **pictorial representation** of the **logic** steps
  2. **Pseudocode**: **English-like representation** of the **logic**

Advice: Walk through the logic before coding

3. Code the Program

- **Code the program:**
  - **Select** the programming **language**
  - **Write** the **program instructions** in the selected programming language
  - Use the **compiler** software to translate the program **into machine understandable code**
  - **Syntax errors** (Error in **program instructions**) are **identified** by the **compiler** during **compilation** and **can be corrected**.

4. Test the Program

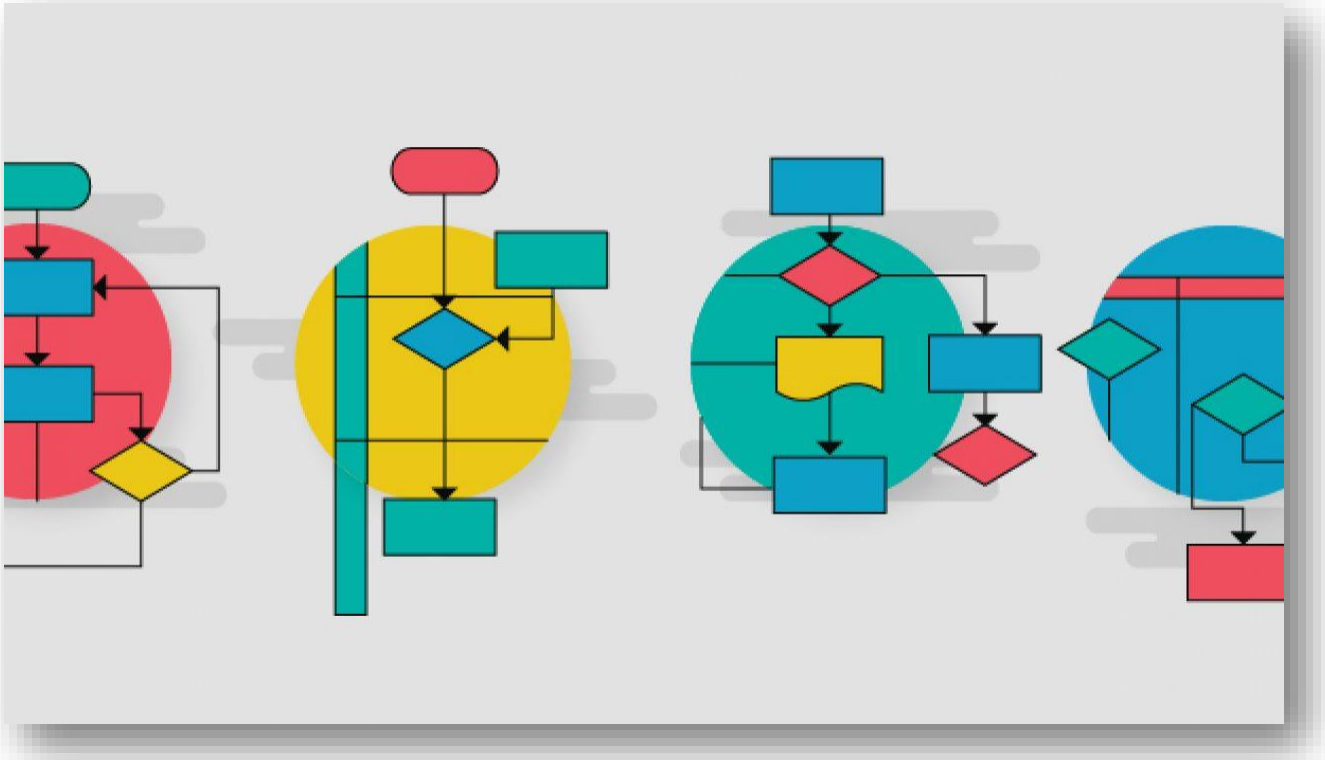
- **Testing the program**
  - **Execute** using **sample data** and **check** the **results**
  - **Identify logic errors** if any (***undesired results or output***) and correct them

5. Deploy the Program

- **Putting the program into production**
  - Do this after **testing is complete** and all **known errors** have been **corrected**

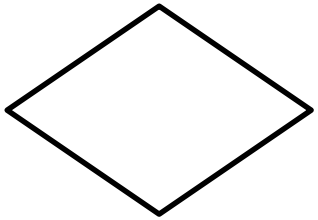
Program Logic: Flowcharts

- "**A graphic representation** of a **sequence** of **operations** to **represent** a computer **program**"
  - Shows **steps of the solution**
  - Shows **individual steps** and their **interconnections**

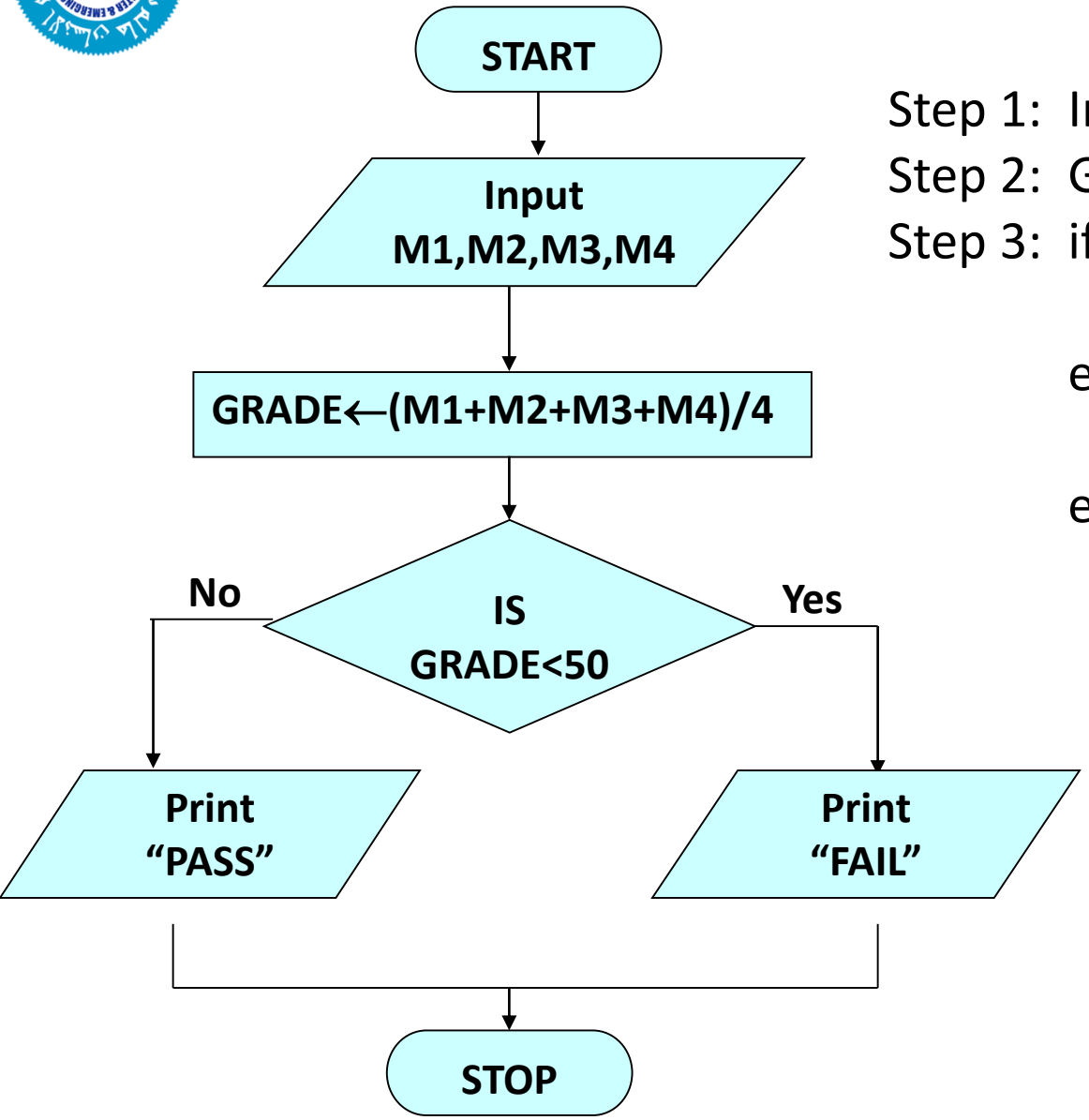


https://d3n817fwly711g.cloudfront.net/uploads/2017/07/flowchart-feature-image-01-1280x720.jpg

Basic Flowchart Symbols

Name	Symbol	Description
Oval		**Beginning** or **End** of the Program
Parallelogram		**Input** / **Output** Operations
Rectangle		**Processing** for example, *Addition*, *Multiplication*, *Division*, etc.
Diamond		Denotes a **Decision** (or branching) for example IF-Then-Else
Arrow		Denotes the **Direction** of **logic flow**

Example 1



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

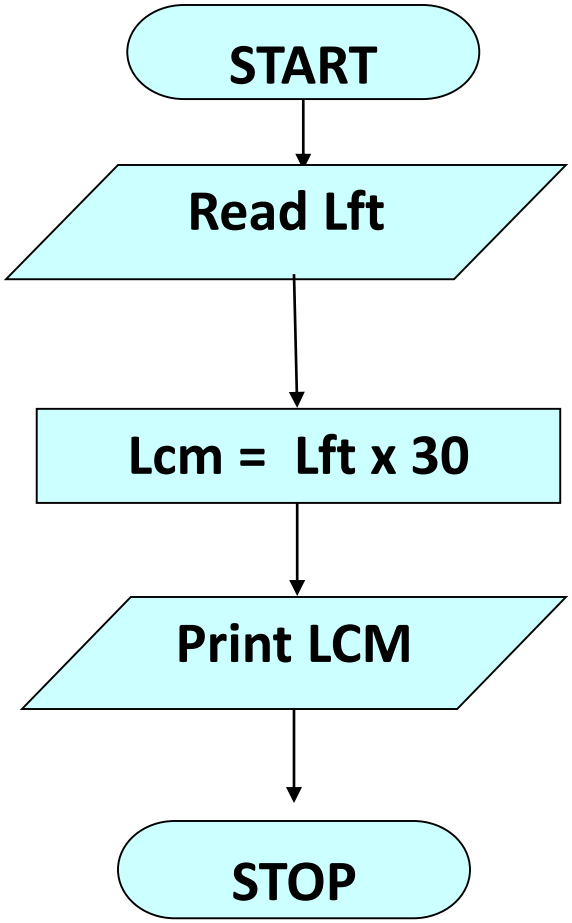
- Write an algorithm and draw a flowchart to convert the length in feet to centimeter.

Example 2

Algorithm

- Step 1: Read Lft
- Step 2: Lcm = Lft x 30
- Step 3: Print Lcm

Flowchart



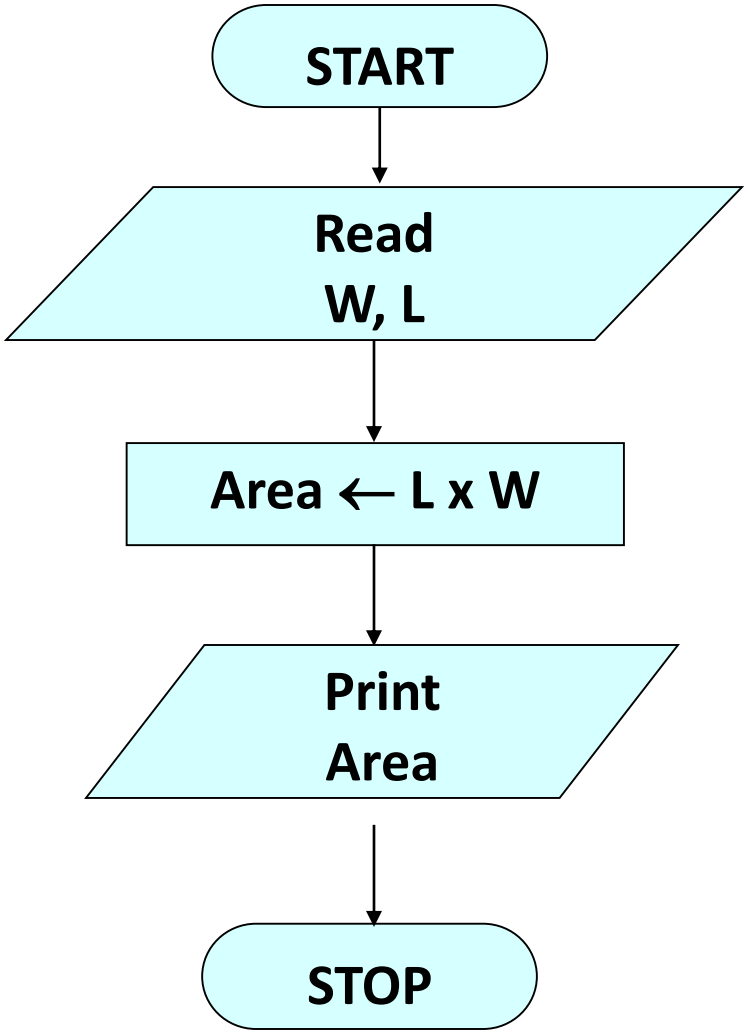
Example 3

- **Write an algorithm and draw a flowchart that will read the Length and Width of a rectangle and calculate its area.**

Example 3

Algorithm

- Step 1: Read W,L
- Step 2: Area = L x W
- Step 3: Print A



Decision Structures

- 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 a action on left*
- **Print** the value of **A**
- ***if A>B is false (if A is not greater than B)** we take a action on right*
- **Print** the value of **B**

IF–THEN–ELSE STRUCTURE

- The algorithm for the flowchart is as follows:

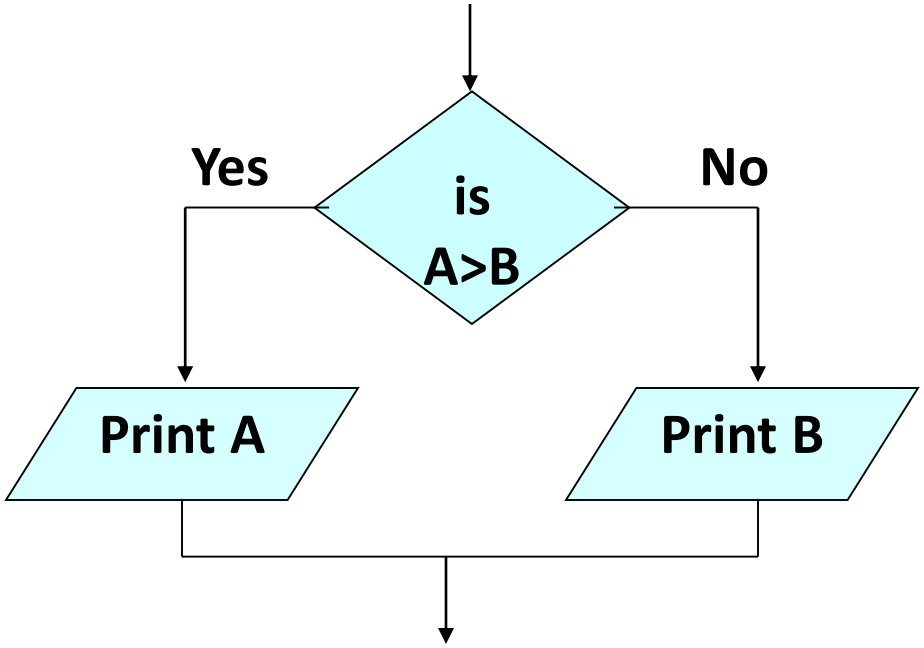
If A>B then

print A

else

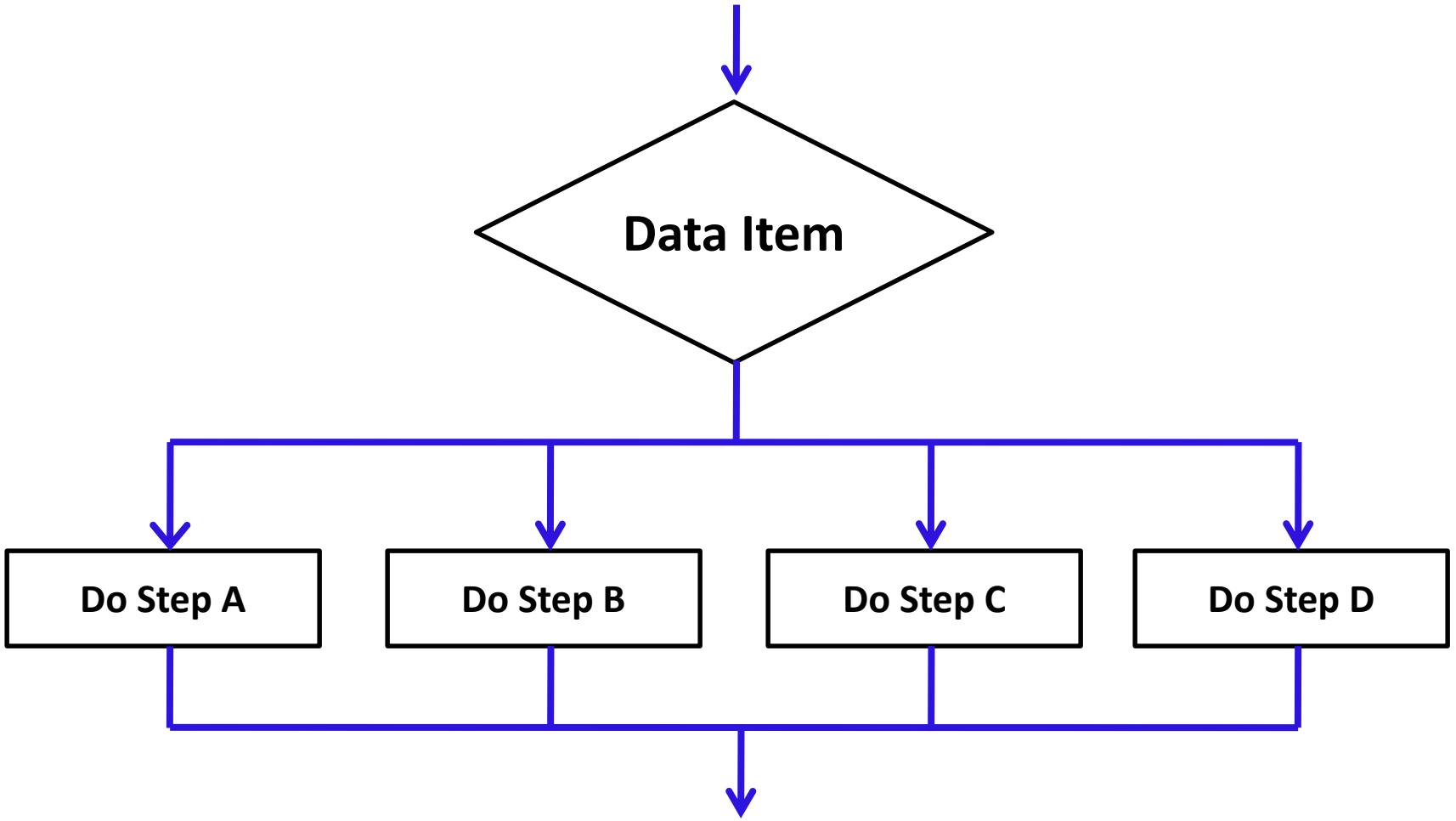
print B

endif



CASE Structure

- **Multiple branching based on a single data item**



Relational / Logical Operators

Relational Operators	
Operator Symbol (Pseudocode)	**Description**
>	Greater than
<	Less than
=	Equal to
≥	Greater than or equal to
≤	Less than or equal to
≠	Not equal to

Example 4

- Write an algorithm that **reads two values**, **finds largest value** and then **prints** the **largest value**.

ALGORITHM

Step 1: ***Read*** **VALUE1, VALUE2**

Step 2: ***if (*****VALUE1 > VALUE2)** *then*

MAX ← VALUE1

else

MAX ← VALUE2

endif

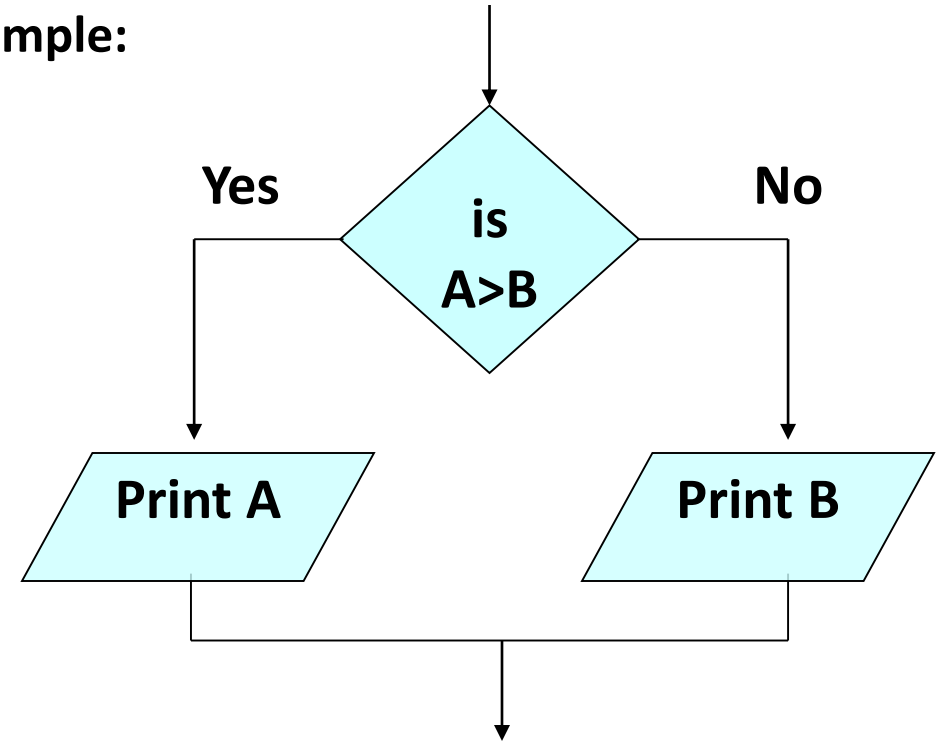
Step 3: ***Print*** *"The largest value is", MAX*

-- DRAW the Flow Chart for the Program

Selection Structure

- A Selection structure can be based on
  1. **Dual-Alternative** (*two code paths*)
  2. **Single Alternative** (*one code path*)

Dual Alternative Example:

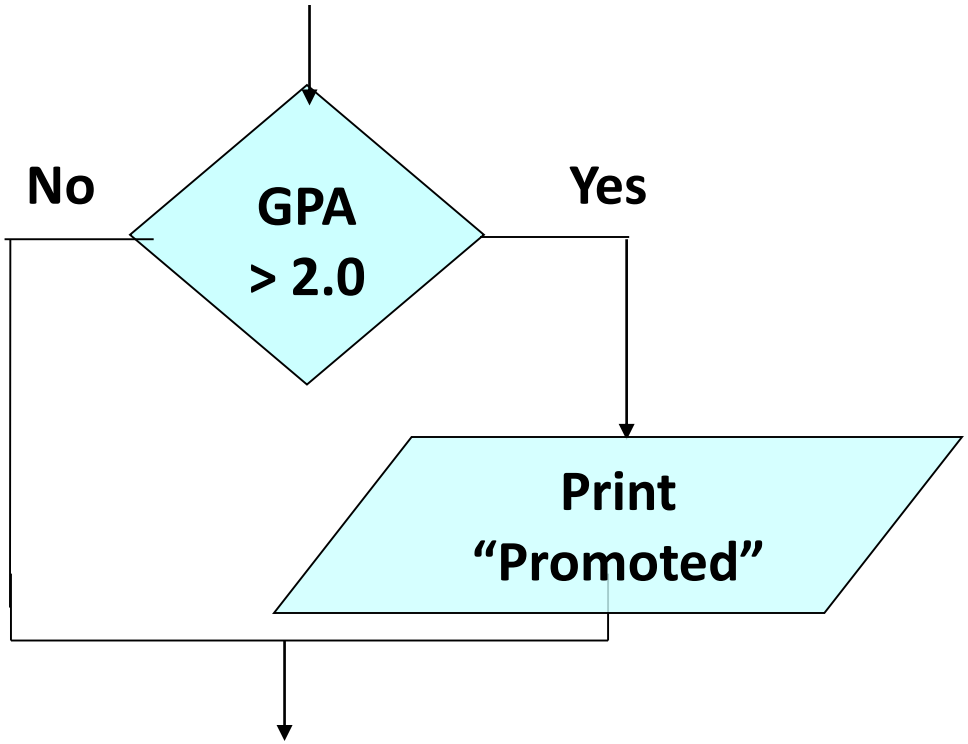


Selection Structure

Single Alternative Example

Pseudocode:

```
IF  GPA is greater than 2.0 Then
        Print "Promoted "
End IF
```



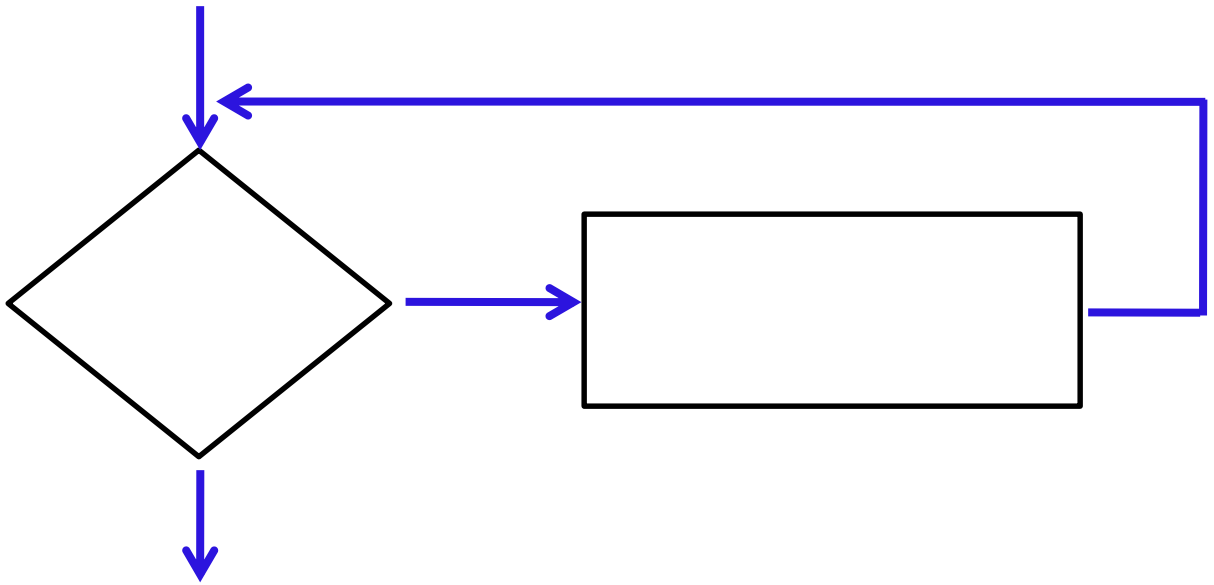
Loop Structure

- **Repetition (WHILE structure)**
  - Repeats a set of actions based on the answer to a *question/condition*

pseudocode: ***DoWHILE*** *<Some-True-Condition>*

Do Something

ENDDO



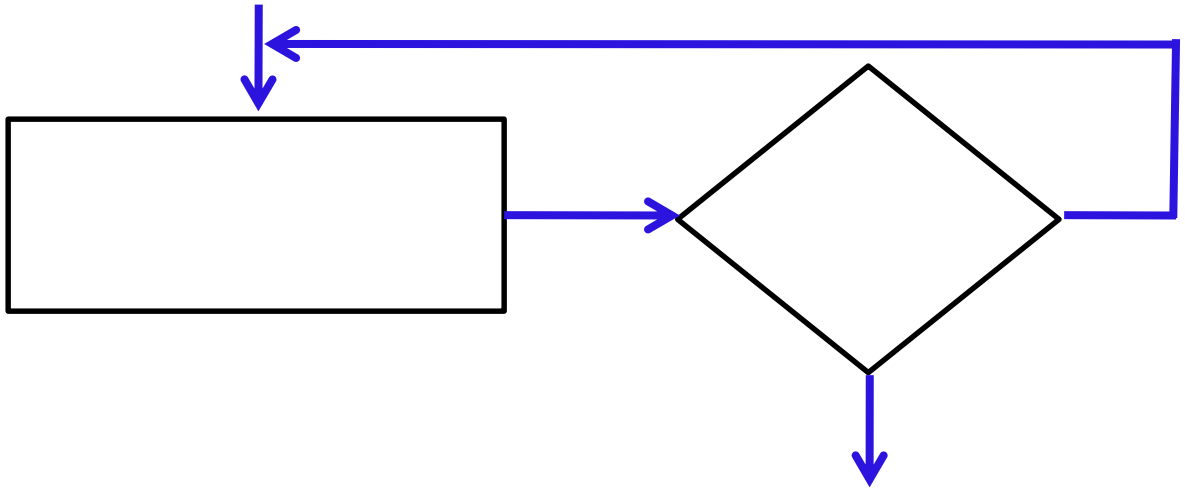
Loop Structure

- **REPEAT-UNTIL structure**
  - Repeats a set of actions until a condition remains True

  - **pseudocode: REPEAT**

Do-Something

UNTIL *<Some True Condition>*



https://www.google.com.pk/url?sa=i&url=https%3A%2F%2Fimgflip.com%2Fi%2Fnp6r7&psig=AOvVaw0hSaZQ4QperYmr4Q-0159s&ust=1630516336101000&source=images&cd=vfe&ved=2ahUKEwijzbDp4NvyAhX3AmMBHQ7ZB7QQjRx6BAgAEAk