First Year First Semester Course
M.Tech. (CS) [Batch 2021-23]

Lecture #01

Introduction to Programming

Basic concepts of Programming and
Programming Paradigms

Course Instructor:

Dr. Monidipa Das

DST-INSPIRE Faculty

Machine Intelligence Unit (MIU), Centre for Artificial Intelligence and Machine Learning (CAIML)

Indian Statistical Institute (ISI) Kolkata, India

Topics of Discussion

- Concept of Programming
  - What and Why
  - Basics of Computer Architecture
  - Algorithm and Its Representations
- Programming Paradigms
  - Programming languages
  - Concept of language translator
- Course Details
  - Course Structure/Syllabus
  - Course Materials
  - Evaluation/Tests

Concept of Programming

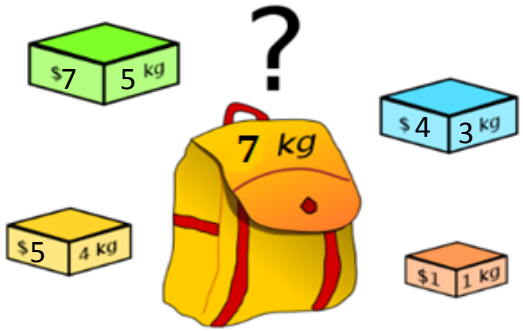
- What and Why
- Basics of Computer Architecture
- Algorithm and Its Representations

Programming: What and Why

- **Programming:**

  →letting the computer know how to ***solve a problem***

Example Problem: "Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible."



Programming: What and Why [contd.]

How to solve a problem using computer?

Algorithm

First, based on your logic, design a step by step procedure to solve the problem

Algorithm Design

- Initial Solution Generation ← Correctness
- Initial Solution Refinement ← Efficiency
  - Time requirement
  - Space requirement

Translated to a set of instructions in some computer language

Programming

Final Design
(preferably the most efficient Algorithm)

Programming: What and Why [contd.]

Processing *(based on instructions given)*

Data (input)

Information (Output)

Algorithm

Translated to a set of ***instructions*** in some ***computer language***

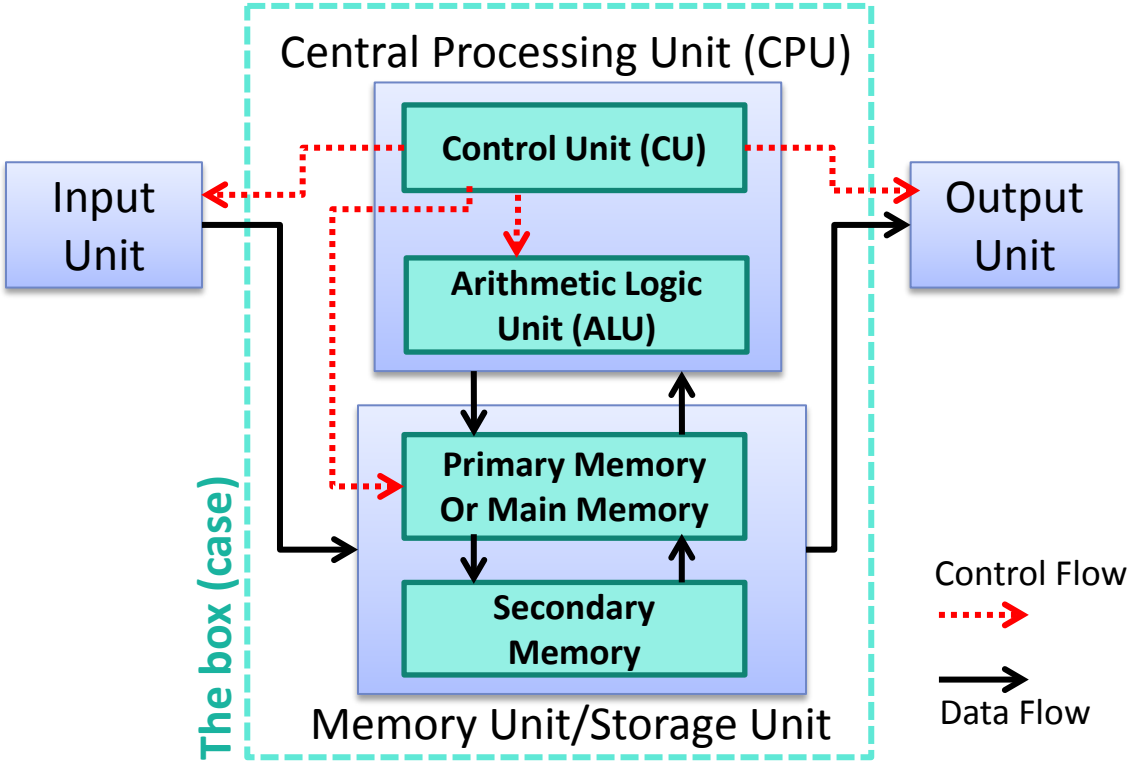
Program or Code

Programming or Coding

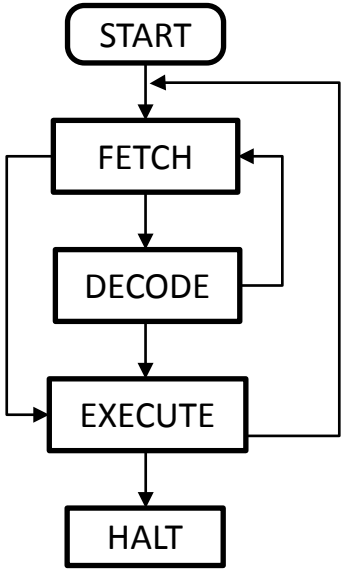
→ letting the computer know how to solve a problem

→ Writing instructions in computer language

Basics of Computer Architecture

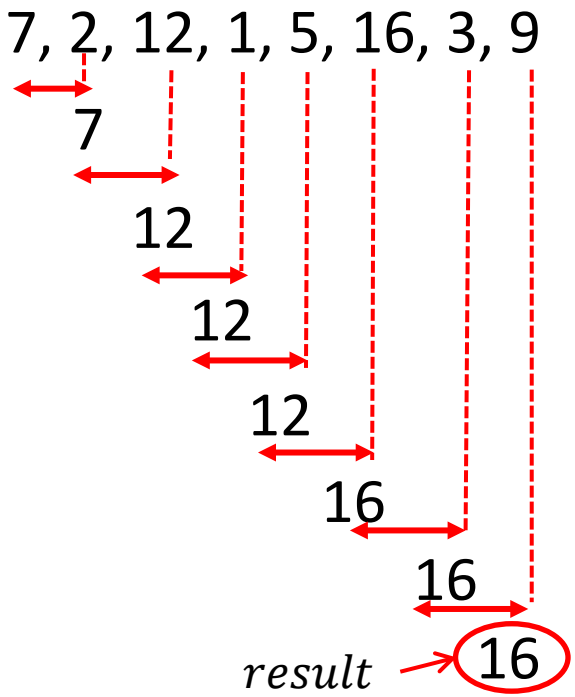


Basic Instruction Processing Cycle



Problem Solving: Examples

Example-1: Find the maximum number from a given list of 8 numbers

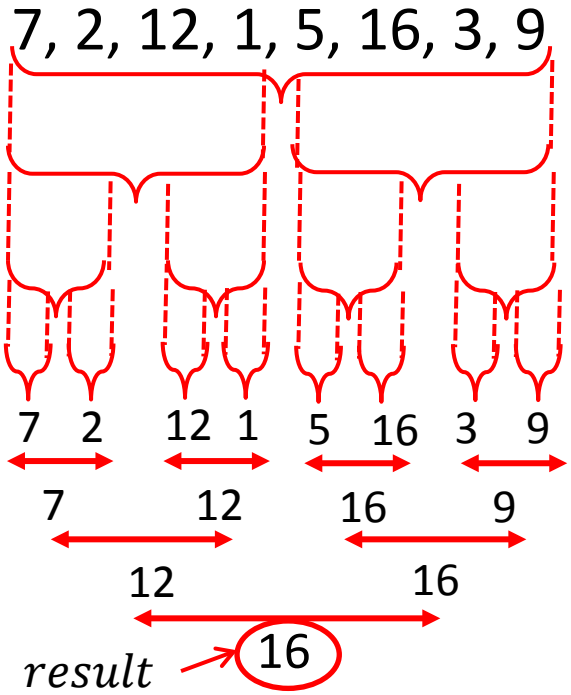


Algorithm-1

Step-1: $n \leftarrow$ READ the count of numbers
Step-2: $max \leftarrow$ READ the 1st number
Step-3: REPEAT for n numbers in the list
$x \leftarrow$ READ number
IF $x > max$ THEN
$max \leftarrow x$
Step-4: $result \leftarrow max$
Step-5: WRITE $result$
Step-6: Stop

Problem Solving: Examples [contd.]

Example-1: Find the maximum number from a given list of 8 numbers



Algorithm-2: ***Find max value from a list of numbers***

Step-1: $n \leftarrow$ READ the count of numbers
Step-2: IF n *is* 1 THEN RETURN the number
Step-3: $max1 \leftarrow$ Find max value from first half of the list
Step-4: $max2 \leftarrow$ Find max value from second half of the list
Step-5: IF $max1 > max2$ THEN
$result \leftarrow max1$
ELSE $result \leftarrow max2$
Step-6: RETURN $result$
Step-7: Stop

MORE Space requirement!

Problem Solving: Examples [contd.]

Example-2: Compute the sum of first 100 natural numbers

1, 2, 3, 4, 5, 6, 7, …, 100

$sum = 0$
$sum = sum + 1 = 1$
$sum = sum + 2 = 3$
$sum = sum + 3 = 6$
$sum = sum + 4 = 10$
$sum = sum + 5 = 15$
$sum = sum + 6 = 21$
$sum = sum + 7 = 28$
.....
$sum = sum + 100 = 5050$

$result$

Algorithm-1

Step-1: $n \leftarrow$ READ the count of numbers
Step-2: $i \leftarrow 1$
Step-3: $sum \leftarrow 0$
Step-4: REPEAT for n times
$sum \leftarrow sum + i$
$i \leftarrow i + 1$
Step-5: $result \leftarrow sum$
Step-6: WRITE $result$
Step-7: Stop

Problem Solving: Examples [contd.]

Example-2: Compute the sum of first 100 natural numbers

1, 2, 3, 4, 5, 6, 7, …, 100

$$Sum\ of\ first\ n\ natural\ numbers = \frac{n \times (n+1)}{2}$$

$$result = \frac{100 \times (100+1)}{2} = 5050$$

$result$

Algorithm-2

Step-1: $n \leftarrow$ READ the count of numbers
Step-2: $result \leftarrow n \times (n+1)/2$
Step-3: WRITE $result$
Step-4: Stop

Efficient in terms of both Time requirement and Space requirement!

Algorithm

Algorithm: ***Sequence of steps to be followed to solve a problem***

Also considered as the ***logic of a program***

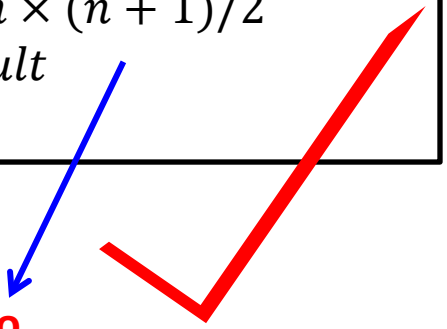
Properties of an Algorithm:

- ***Input:*** An algorithm should have some inputs
- ***Output:*** At least one output should be returned by the algorithm after the completion of the specific task based on the inputs given
- ***Definiteness:*** Every statement of the algorithm should be clear and unambiguous
- ***Finiteness:*** No infinite loop should be allowed in an algorithm
- ***Effectiveness:*** Performance can be judged in finite time using pen-paper

Examples

- Example-1

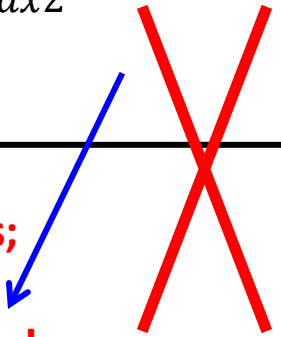
Step-1: $n \leftarrow$ READ the count of numbers
Step-2: $result \leftarrow n \times (n+1)/2$
Step-3: WRITE $result$
Step-4: Stop



Sequence of steps to compute the sum of first n natural numbers

- Example-2

Step-1: $x_1 \leftarrow$ READ the 1st number
Step-2: $x_2 \leftarrow$ READ the 2nd number
Step-3: IF $x_1 > x_2$ THEN
$result \leftarrow x_1$ or $result \leftarrow 2x_1$
ELSE $result \leftarrow max2$
Step-4: RETURN $result$
Step-5: Stop



Ambiguous; Lack of definiteness!

Algorithm [contd.]

- Any complex algorithm (or logic of a program) can be expressed using only the following ***three*** *simple logical constructs*
  - *Sequence logic*
    - Used for executing instructions one after another in sequence
  - *Conditional logic*
    - Used for making decision (e.g. “IF…….. THEN”)
  - *Iteration (or looping) logic*
    - Used when the same instruction to be executed several times (e.g. “REPEAT…….UNTIL”, “DO…….WHILE”)

Representations of Algorithm

- **Pseudocodes**
  - Pseudo→ False;
  - Code → A set of instructions written in a programming language
  - Pseudocode → Set of instructions written in a ***natural language*** to describe the logic or steps to solve the problem;

    Cannot be directly understood by the computer;

- **Flowcharts**
  - Pictorial representations of algorithm
  - Uses different geometric shapes to denote different types of instruction

Pseudocodes

- Set of instructions written in a natural language to *describe the logic or steps* to solve the problem;

- How to write pseudocode to express various logics?

```
Process Step-1
Process Step-2
....
Process Step-n
```

(a) Sequence logic

```
IF Condition
THEN
    Process Steps…
ELSE
    Process Steps…
ENDIF
```

```
IF Condition
THEN
    Process Steps…
ENDIF
```

(b) Conditional/Decision logic

```
REPEAT
    Process Steps…
UNTIL Condition
```

```
DO WHILE Condition
    Process Steps…
ENDDO
```

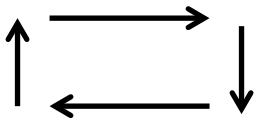
(c) Iteration logic

Flowcharts

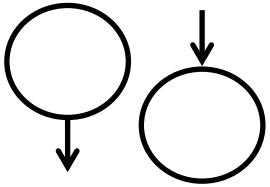
- **Pictorial representations** of algorithm
- Uses **standard geometric shapes** to denote different types of instruction

Flowchart symbols

- **Basic Flowchart Symbols:**

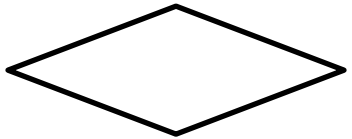


Flow lines



Connectors

Process step

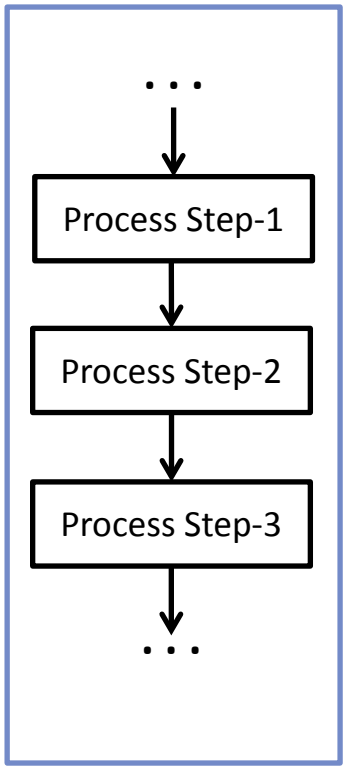


Decision

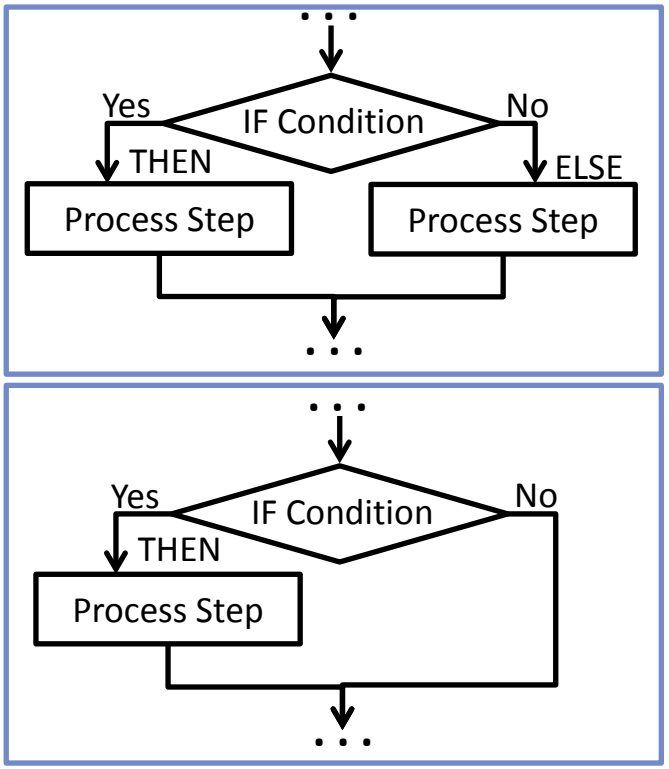
Terminal

Input/Output

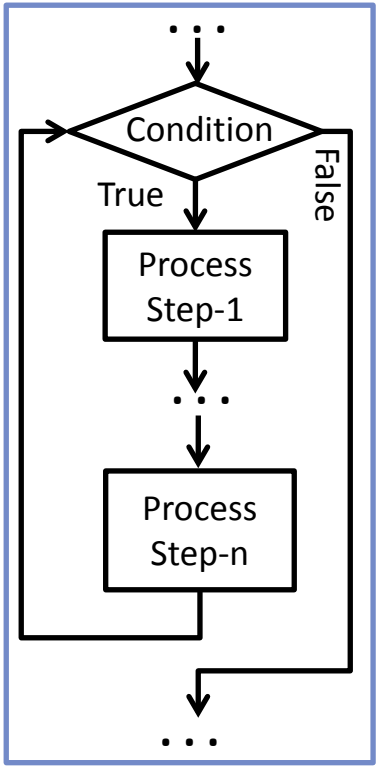
Flowcharts for Simple Logical Constructs

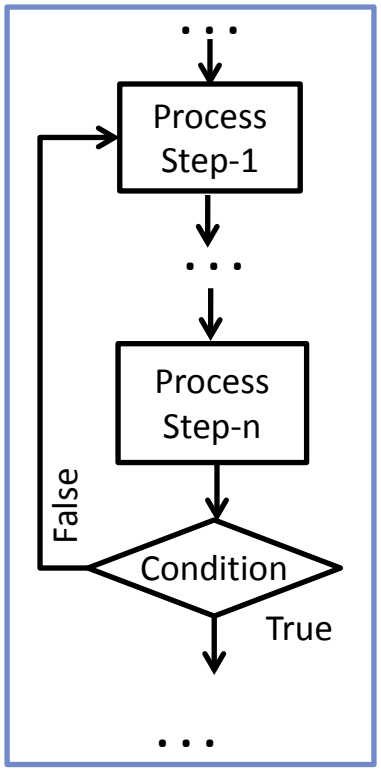


(a) Sequence logic



(b) Conditional/Decision logic

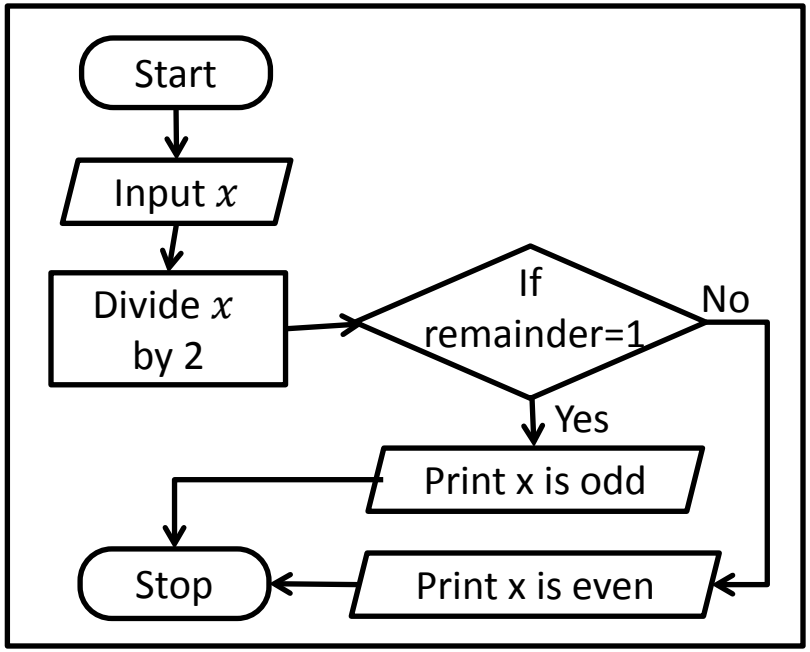




(c) Iteration logic

Examples

Give a flowchart for an algorithm determining whether a number x is odd or even

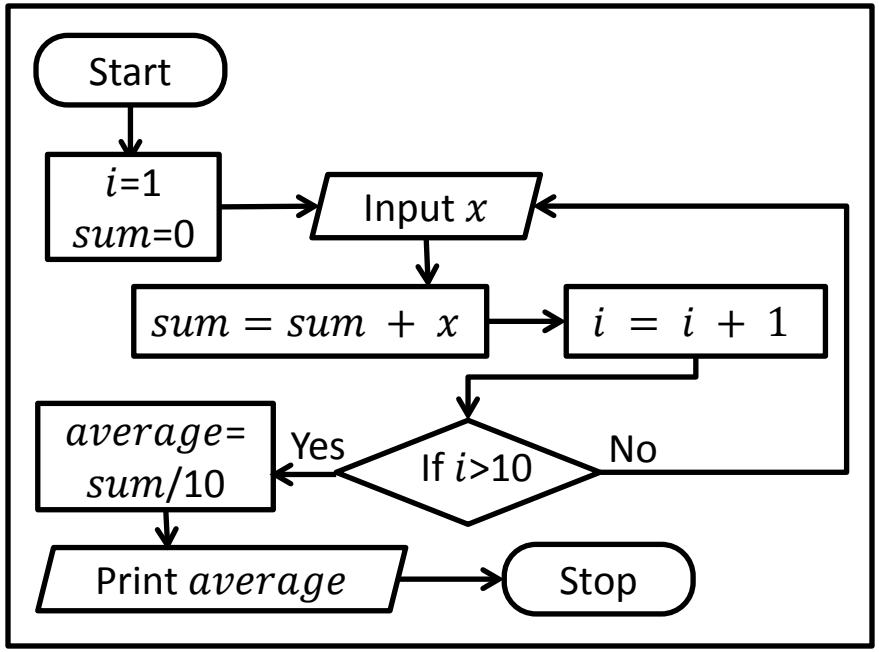


Pseudocode

1. Read the number x
2. Divide x by 2
3. IF remainder is 1 THEN
   Print x is odd
   ELSE
   Print x is even
   ENDIF
5. Stop

Examples [contd.]

Give a flowchart for an algorithm determining the average of 10 numbers



Pseudocode

1. Set i as 1
2. Set sum as 0
3. REPEAT
   - Read number
   - Add the number to sum
   - Increase i by 1

   UNTIL $i > 10$
4. Divide the sum by 10 to get the $average$
5. WRITE $average$
6. Stop

Programming Paradigms

- Programming languages
- Concept of language translator

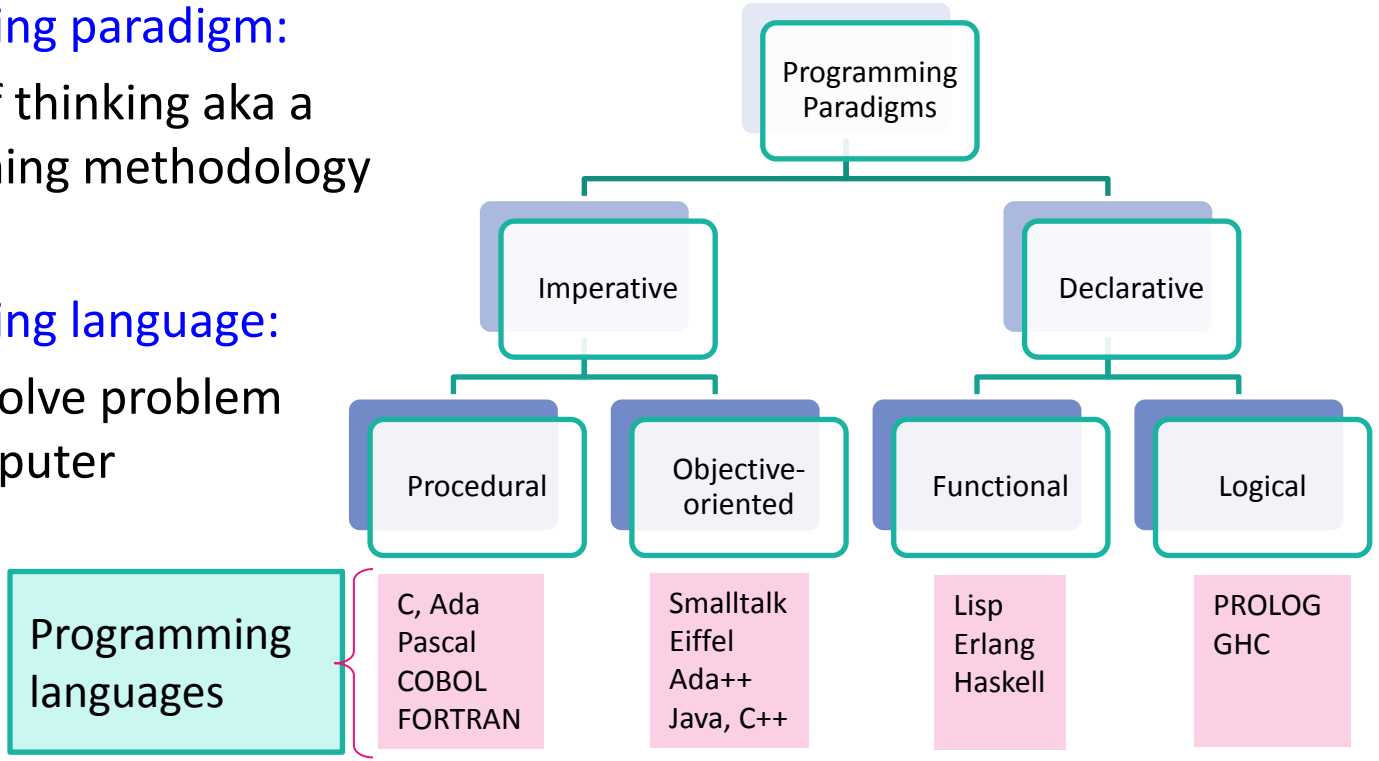
Programming Paradigms

- **Programming paradigm:**

  A mode of thinking aka a programming methodology

- **Programming language:**

  A tool to solve problem using computer



Declarative vs. Imperative Programming

Declarative vs. Imperative Programming	
A programming paradigm that expresses the logic of a computation without describing its control flow.	A programming paradigm that uses statements that changes the program's state.
Main Focus	
Focuses on what the program should accomplish.	Focuses on how the program should achieve the result.
Flexibility	
Provides less flexibility.	Provides more flexibility.
Complexity	
Simplifies the program.	Increase the complexity of the program.
Categorization	
Functional, Logic, Query programming falls into declarative programming.	Procedural and Object Oriented programming falls into imperative programming.

Procedural vs. Object-oriented Programming (OOP)

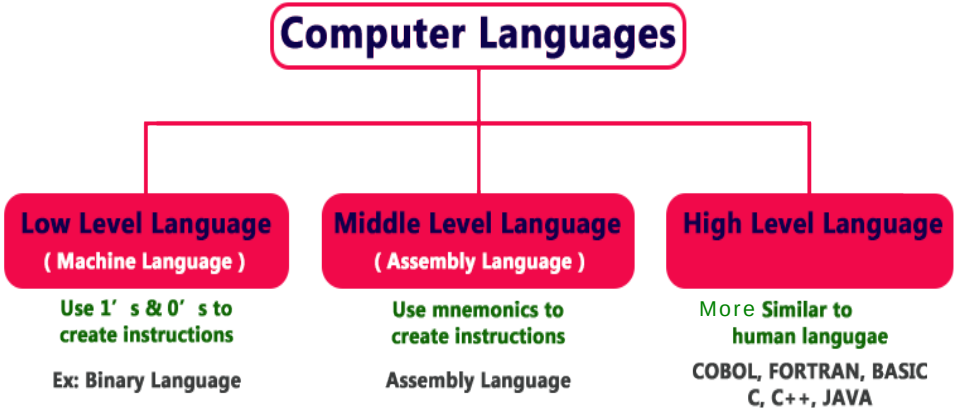
Procedural programming	Object-oriented Programming (OOP)
In Procedural programming, a program is divided into small programs that are referred to as functions.	In OOP, a program is divided into small parts that are referred to as objects.
It follows a top-down approach	It follows a bottom-up approach
It treats data and methods separately	It encapsulates data and methods together
It is less secure than OOPs	It is more secure than procedural programming

Functional vs. Logical Programming

Functional Programming	Logical Programming
Programs are composed of functions	Programs are composed of facts and rules
Program evaluation is one-way	Program evaluation can be two-way
Helps increasing modularity	Helps representing and extracting knowledge

Levels of Computer Programming Languages

- ***Programming language:*** Language required to communicate with a computer; Also called ***Computer language***
- Computer language can be broadly classified into ***three*** categories:
  - **Machine Language**
  - **Assembly Language**
  - **High-level Language**



Language Translator

- Assembler

- Compiler

- Interpreter*

*(Works instruction-wise; Does not produce object code)

Interpreter vs. Compiler

Interpreter	Compiler
Interpreter translates just one statement of the program at a time into machine code.	Compiler scans the entire program and translates the whole of it into machine code at once.
Takes very less time to analyze the source code. However, the overall time to execute the process is much slower.	Takes a lot of time to analyze the source code. However, the overall time taken to execute the process is much faster.
Does not generate an intermediary code. Hence, an interpreter is highly efficient in terms of its memory.	Generates an intermediary object code. It will need further linking. Hence more memory is needed.
Keeps translating the program continuously till the first error is confronted. If any error is spotted, it stops working and hence debugging becomes easy.	Generates the error message only after it scans the complete program and hence debugging is relatively harder while working with a compiler.
Used by programming languages like Ruby and Python for example.	Used by programming languages like C and C++ for example.

Course Details

- Course Structure/Syllabus
- Course Materials
- Course Evaluation/Tests

Course Structure/Syllabus

- Introduction to Programming

[Writing, compiling, and running basic programs]

  - Introduction to an imperative language (preferably C); syntax and constructs.
  - Functions and parameter passing, call by value, call by reference; recursion.
  - Basics of object-oriented programming: Introduction to object oriented programming
  - Classes and methods, polymorphism, inheritance; basics of C++ or Java.
  - Basics of functional programming, logic programming.
  - Efficiency issues.

Course Materials

- References Texts
  - B. W. Kernighan and D. M. Ritchie: The ‘C’ Programming Language
  - B. Gottfried: Programming in C
  - B. Stroustrup: The C++ Programming Language, Addison-Wesley,
  - Cay S. Horstmann: Core Java Volume I – Fundamentals, 11th ed., Prentice Hall, 2018.
  - B.W. Kernighan and R. Pike: The Practice of Programming, Addison-Wesley.
  - B.W. Kernighan and P.J. Plauger: The Elements of Programming Style, McGraw-Hill.
  - B.W. Kernighan and R. Pike: The Unix Programming Environment, Prentice Hall.

Evaluation/Tests

- Distribution of Marks:

✓	Assignment:	50%
✓	Exam:	50%

Questions?