



ICDT 1201Y (3)

Computer Programming (week 2)



Algorithm and Design - Problem Solving Concepts



Sub-topics

- Problem Solving
- Abstraction
- Concept of Algorithms
- Program Design using Flowchart



Learning Outcomes & Assessment Criteria

Learning Outcomes:

- Principles of Problem-Solving:
 - Identify and explain problem-solving techniques and their importance in programming.
- Terminology:
 - Define and use terms like algorithm, abstraction, and pseudocode.
- Problem-Solving Tools:
 - Recognize and use tools such as flowcharts and pseudocode
- Flowchart Design:
 - Design and interpret flowcharts for program development.

Assessment Criteria:

- Problem-Solving Techniques:
 - Identify and explain various techniques.
 - Describe their importance in programming.
- Terminology:
 - Define and use key terms correctly.
- Algorithms:
 - Define algorithms and explain their characteristics.
- Flowcharts:
 - Design flowcharts and explain their symbols and use.



Problem Solving Concepts

- Importance of Problem Solving
 - Helps in breaking down complex problems into manageable parts
 - Fundamental for effective programming
- **Techniques:**
 - **Understanding the Problem** - Grasp the problem requirements and constraints.
 - **Devising a Plan** - Develop a step-by-step solution (algorithm).
 - **Executing the Plan** - Implement the solution in a programming language.
 - **Evaluating the Solution** - Test and refine the solution to ensure it solves the problem.



Main Phases of Problem Solving

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 a number of steps:
 - Analyzing the problem.
 - Developing the Hierarchy Input Process Output (HIPO) chart or Interactivity Chart (IC) along with the Input-Process-Output (IPO)Chart (where applicable).
 - Writing the algorithms. *
 - Drawing the Program flowcharts.*



Complexities of Problems

- Problems' complexities vary
- It might not be necessary to understand all the details to solve a problem.
 - Example: Your car suffers from a flat tire; Is it necessary for you to know how the tire was manufactured, which raw materials were used, which country it was from etc? Or you just need to know the steps to change the tire? By overlooking the unnecessary details, we end up solving the problem faster...(imagine yourself watching hours of videos how tires are made).
 - This was an example of abstraction!



Abstraction

- Basic Definition
 - Simplifying complex systems by focusing on the main aspects and ignoring the details.
- Levels of Abstraction
 - High-level (e.g., overview of a system)
 - Mid-level (e.g., detailed design)
 - Low-level (e.g., specific implementation details)



Examples of Abstraction

- Example 1: Car as a system
 - High-level: Car as a means of transportation
 - Mid-level: Car's components (engine, wheels, etc.)
 - Low-level: Detailed design of the engine
- Example 2: Computer program
 - High-level: Software application
 - Mid-level: Modules and functions
 - Low-level: Code implementation



Importance of Abstraction

- Helps manage complexity
- Enhances code readability and maintainability
- Facilitates modular design



Analyzing the Problem (1)

- 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.

Analyzing the Problem (2)

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

Data	Processing	Output
Given in the problem or provided by the User.	List of processing required or Procedures.	Output requirements.

- **Further Reading:** In case the problem is big and complex, the processing can be divided into sub-tasks called modules; with each module accomplishing one function. This interaction between modules can presented using a Hierarchy Input Process Output Chart (HIPO) or an Interactivity Chart (IC)



Concept of Algorithms

- An algorithm
 - A finite sequence of well-defined instructions to solve a specific problem.
- Characteristics
 - Clear and unambiguous
 - Finite number of steps
 - Well-defined inputs and outputs
 - Effective and efficient
- Few Examples
 - Common algorithms such as sorting and searching
 - Finding the maximum number in a list.



Algorithms Vs Programs

- Algorithm: Step-by-step solution
- Program: Implementation of the algorithm in a programming language



Algorithm Examples (1)

- Calculate the total (sum) of two numbers
 1. Input the two numbers
 2. Take first number and add to the second number
 3. Display the value obtained in 2.



Algorithm Examples (2)

- Finding the larger number between two numbers
 1. Input the two numbers
 2. Take first number and check if it is larger than the second number
 3. If first number is larger than second
 - 3.1 Display first number is larger
 4. Otherwise
 - 4.1 Display second number is larger



Algorithm Examples (3)

- Finding the largest number from a group of numbers
 1. Input all the numbers as a list
 2. Assume the first number in the list is the largest
 3. Compare the value of the largest with the next number from the list
 - 3.1 If the next number is larger, update largest value
 - 3.2 Continue comparison till end of the list is reached
 4. Display/Return the value of largest



Class Exercise

1. Write an algorithm to find the minimum number in a list.
2. Write an algorithm to find the average value of three numbers.
3. Write an algorithm to find the average value of a list of numbers.



Pseudocode

- Pseudocode is a way of writing out an algorithm in plain language using a structure that resembles programming code.
- Purpose: Helps in understanding and designing algorithms without worrying about syntax
- Characteristics:
 - Readable and easy to understand
 - Uses plain language
 - Structured similarly to Computer Programming Code
 - bridges the gap between the algorithm and the actual code implementation




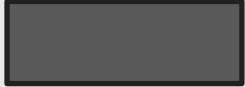

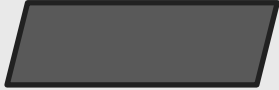

Algorithm Written as Pseudocode

1. Input all the numbers as a list
2. Assume the first number in the list is the largest
3. Compare the value of the largest with the next number from the list
 - 3.1 If the next number is larger, update largest value
 - 3.2 Continue comparison till end of the list is reached
4. Display/Return the value of largest

1. Input: list_of_numbers
2. max = list_of_numbers[0]
3. For each number in list_of_numbers :
 - a. If number > max:
 - i. max = number
4. Output: max

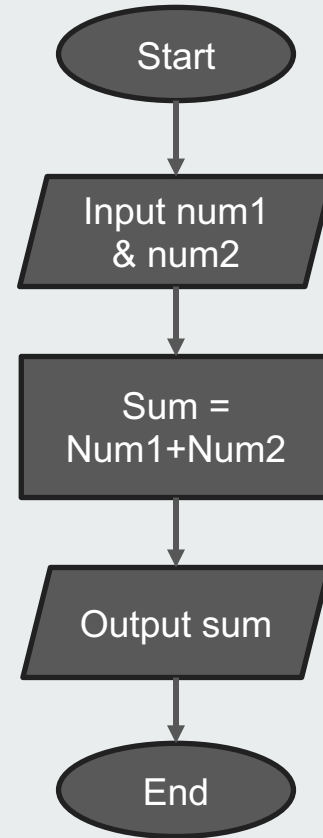
Program Design Using Flowchart

- Visual representations of the steps in an algorithm
- Symbols used in Flowcharts

Symbol Name	Symbol	Representation
Oval		Start / End
Rectangle		Process
Diamond		Decision
Parallelogram		Input / Output
Directional Arrows		Flow of Control

Flowchart Examples – Sum of two numbers

1. Start
2. Input two numbers
3. Calculate the sum
4. Output the sum
5. End





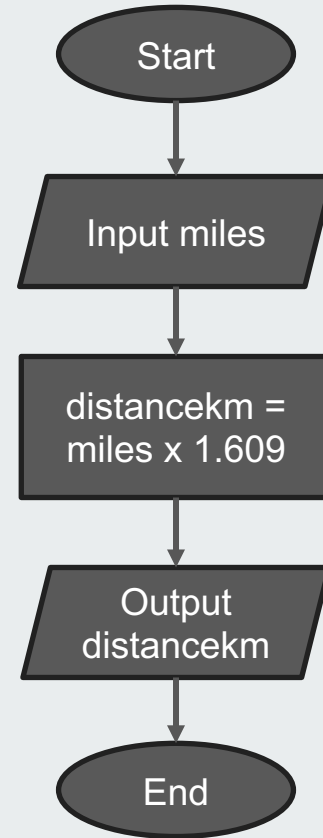
Flowchart Best Practices

- Use standard symbols
- Keep it simple and readable
- Clearly define inputs and outputs
- Ensure logical flow and correctness

Flowchart Examples – Distance Conversion

1. Write the pseudocode and draw the Flow Chart and to convert the distance in miles to kilometers where 1.609 kilometers is equivalent to 1 mile.

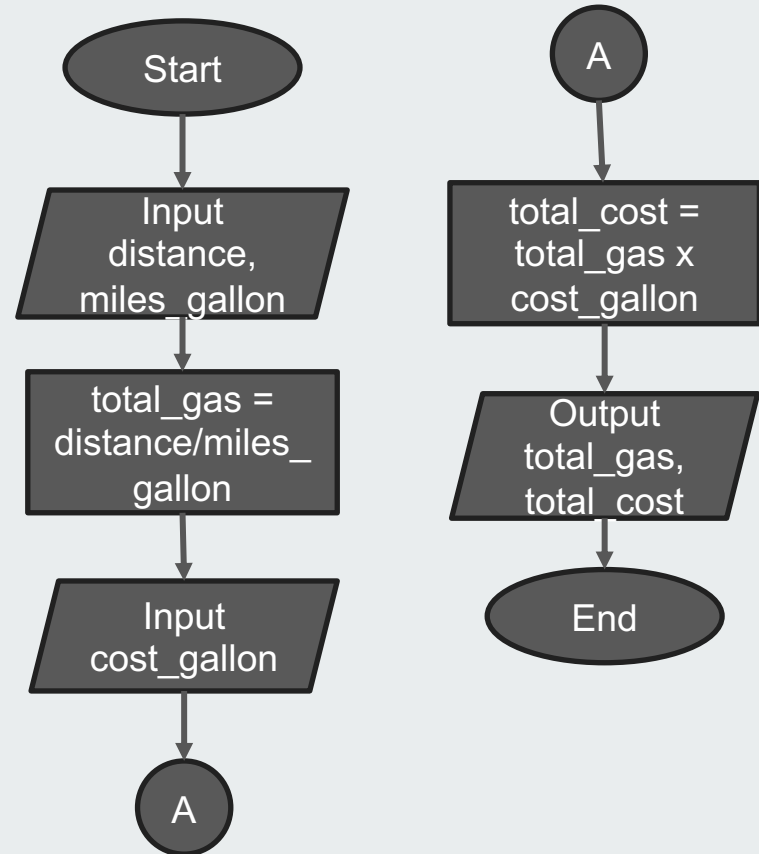
1. Start
2. Input miles
3. $\text{distancekm} = \text{miles} \times 1.609$
4. Output distancekm
5. End



Flowchart Examples – Circle Area

Write the pseudocode and draw the Flow Chart 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.

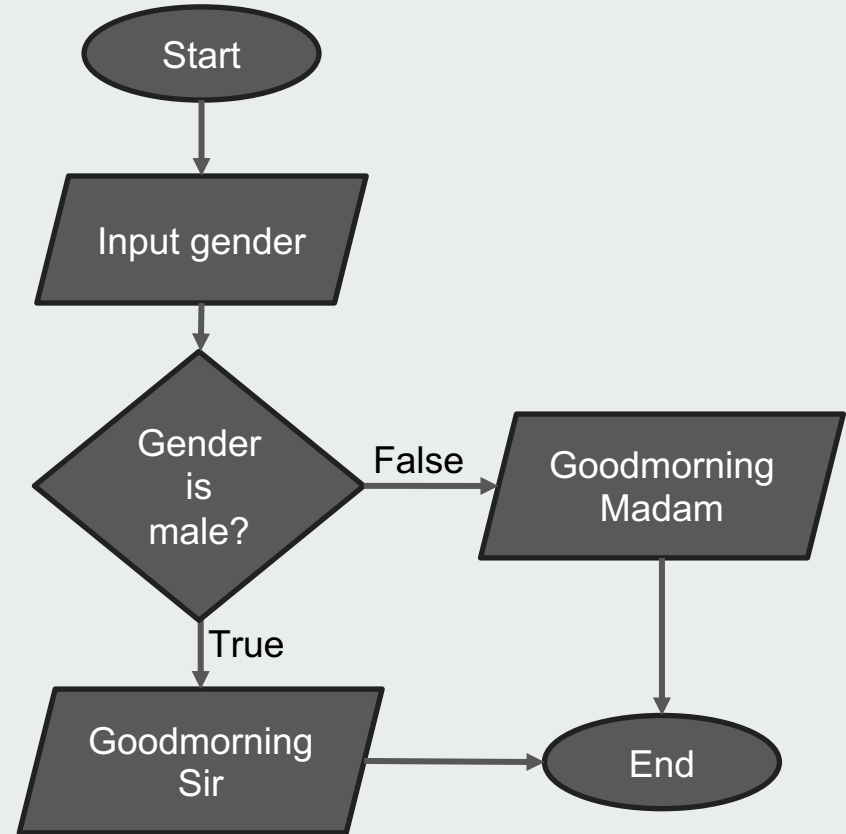
1. Start
2. Input distance and miles_gallon
3. $\text{total_gas} = \text{distance} / \text{miles_gallon}$
4. Input cost_gallon
5. $\text{total_cost} = \text{total_gas} \times \text{cost_gallon}$
6. Output total_gas, total_cost
7. End



Flowchart Examples – Decisions

Write the pseudocode and draw the Flow Chart that asks a user whether he/she identifies as male or female and greet them accordingly with Goodmorning Sir or Goodmorning Madam.

1. Start
2. Input gender
3. If gender is male
 - 3.1 Output “Goodmorning Sir”
 - else
 - 3.2 Output “Goodmorning Madam”
4. End

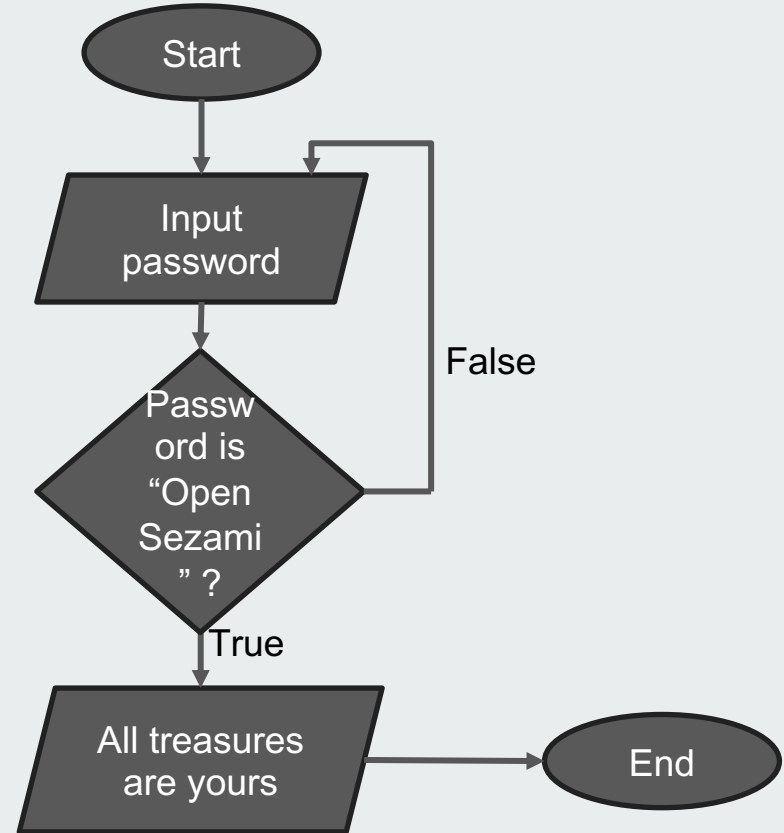


Flowchart Examples – Repetitions / Iterations (1)

Write the pseudocode and draw the Flow Chart that asks a user the password. As long as the wrong password is provided, inform the user that the password is wrong and ask the user for the password again. The password is "OpenSezami". If successful, inform the user that all the treasures are his/hers!

1. Start
2. Input password
3. If password is "OpenSezami"
 - 3.1 Output "All treasures are yours!"
- else
 - 3.2 Output "Wrong Password"
 - 3.3 GoTo Line 2
4. End

What happens if the right password is never entered?



Flowchart Examples – Repetitions / Iterations (2)

Draw the Flow Chart!

Write the pseudocode and draw the Flow Chart that ask the user for the number of students in the class and then for each student, ask for his/her mark. Finally, calculate and display the average mark of the class.

1. Start
2. Input num_students
3. total_marks=0
4. If num_students not 0
 - 4.1. input mark
 - 4.2. total_marks=total_marks+mark
 - 4.3. num_students=num_students - 1
 - 4.4. GoTo Line 4
5. average= total_marks/num_students
6. Output average
7. End

Acknowledgements

- Slides adapted from:
- & Dr P.Appavoo's slides

