

Algorithm, Pseudocode and Flowchart

Md. Aminul Islam Shazid

Outline

1 Introduction

2 Algorithms

3 Control Structures

4 Flowcharts

5 Pseudocode

6 Examples

7 Exercises

Introduction

Introduction

- Algorithms, flowcharts, and pseudocode are essential tools for problem-solving
- They provide a bridge between problem analysis and actual programming
- This lecture introduces their concepts, notations, and best practices

Algorithms

What is an Algorithm?

- A step-by-step procedure to solve a problem
- Unambiguous and finite sequence of instructions
- Example: A recipe for cooking is an algorithm in real life

Characteristics of a Good Algorithm

- Finiteness: must terminate after finite steps
- Definiteness: each step is clearly defined
- Input: specified set of inputs
- Output: specified set of outputs
- Effectiveness: steps can be performed with available resources

Examples of Simple Algorithms

- Finding the maximum of three numbers
- Calculating factorial of a number
- Linear search in an array

Example Algorithm: Finding the Area of a Triangle

- ① Input base, b and height, h
- ② Let area, $a = bh/2$
- ③ Output a

Control Structures

Control Flow and Structure of a Program

- Need to be familiar with control structure to be able to write algorithms
- Control flow or control structure can be divided into a few types:
 - **Sequence**: step by step execution of commands from top to bottom
 - **Selection** or **conditional execution**: executing a codeblock if certain conditions are met (if-else statement)
 - **Iteration** or **loop**: repeatedly executing a block of code or commands while a certain condition is true, stop the loop if the condition is no longer true
- Every program can be built using the three structures

Additionally, functions (collection of commands) with zero or more inputs can be defined.

Sequence

- Code or commands are executed step by step, or sequentially
- Example: Input a number, then calculate its square, then print the result

Example Algorithm: Showing the Square of a Number

- ① Input a number, a
- ② Let square, $s = a^2$
- ③ Output s

In the above, the commands are executed from top to bottom sequentially.

Selection or Conditional Execution

- **IF**: execute a block if condition is true
- **IF-ELSE**: choose between two alternatives
- **ELSE IF ladder**: multiple conditions
- Can have an **IF** statement inside another, this is called nested **IF** statements

Example Algorithm: Finding the Larger of Two Numbers

- ① Input two numbers, a and b
- ② If a is larger than b :
 - i Then, output a
 - ii Else, output b

Iteration or Loops

- **FOR loop:** repeatedly execute commands for a fixed number of times
- **WHILE loop:** repeatedly execute a block of code while a condition is true. Usually, the number of iteration required until the condition becomes false, is not known advanced
- **DO-WHILE loop:** run the commands at least once, then repeat if condition holds
- Can have a loop inside another loop, it is known as nested looping

For the purpose of this slide, only **WHILE** loop shall be used to keep things simple for now.

Example Algorithm: Outputting the First n Integers

- ① Input n
- ② Set $i = 1$
- ③ While $i <= n$:
 - ① Output i
 - ② $i = i + 1$

Break and Continue

- **BREAK**: exit a loop immediately without any further iteration. When inside nested loops, it exits out of the loop in which the **BREAK** statement is called
- **CONTINUE**: skip the rest of the current iteration, proceed to the next iteration

Functions

- Collection of commands that perform a specific task
- Groups together logic or commands that needs to be written across multiple places in a program
- Usually given a name
- Can call a function with its name followed by its parameters in brackets
- Can have zero or more inputs. These inputs are known as parameter or arguments
- Since a function is only defined once and subsequently called only using its name, this reduces code duplication leading to better readability and maintainability of code

Recursion

- Function calling itself to solve smaller subproblems
- Example: factorial, Fibonacci
- Must have a base case to terminate
- A base case is a condition which when true, the function stops calling itself and returns the final result
- The function must be able to reach its base case, otherwise it will turn into an infinite loop

Before Designing an Algorithm

Before writing an algorithm, think carefully about the following:

- **Inputs:** What data is required to solve the problem?
- **Outputs:** What results should be shown?
- **Variables:** What values need to be stored and updated during execution?
- **Processing steps:** What operations or calculations are required?
- **Formulas:** What mathematical or logical formulas are needed?
- **Decision making:** Are conditional checks (IF - ELSE) required?
- **Repetition:** Are loops required, and should the output be shown once or repeatedly?
- **Loop control:** What condition starts and stops each loop?
- **Recursion:** If recursion is used, what is the base case and how does the problem reduce?

Example Algorithm: Factorial of a Number

- ① Input an integer, n
- ② Set result = 1
- ③ While n is larger than 1, repeat the following:
 - i result = result \times n
 - ii $n = n - 1$
- ④ Output the result

Note: In step 3, “While” is a looping construct.

The statements under the “While” key-word are executed repeatedly as long as the condition (n is larger than 1) is true.

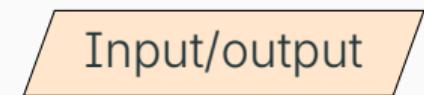
Flowcharts

Definition and Purpose

- Flowchart: graphical representation of an algorithm
- Uses standard symbols to show the flow of control
- Helps visualize program logic before coding

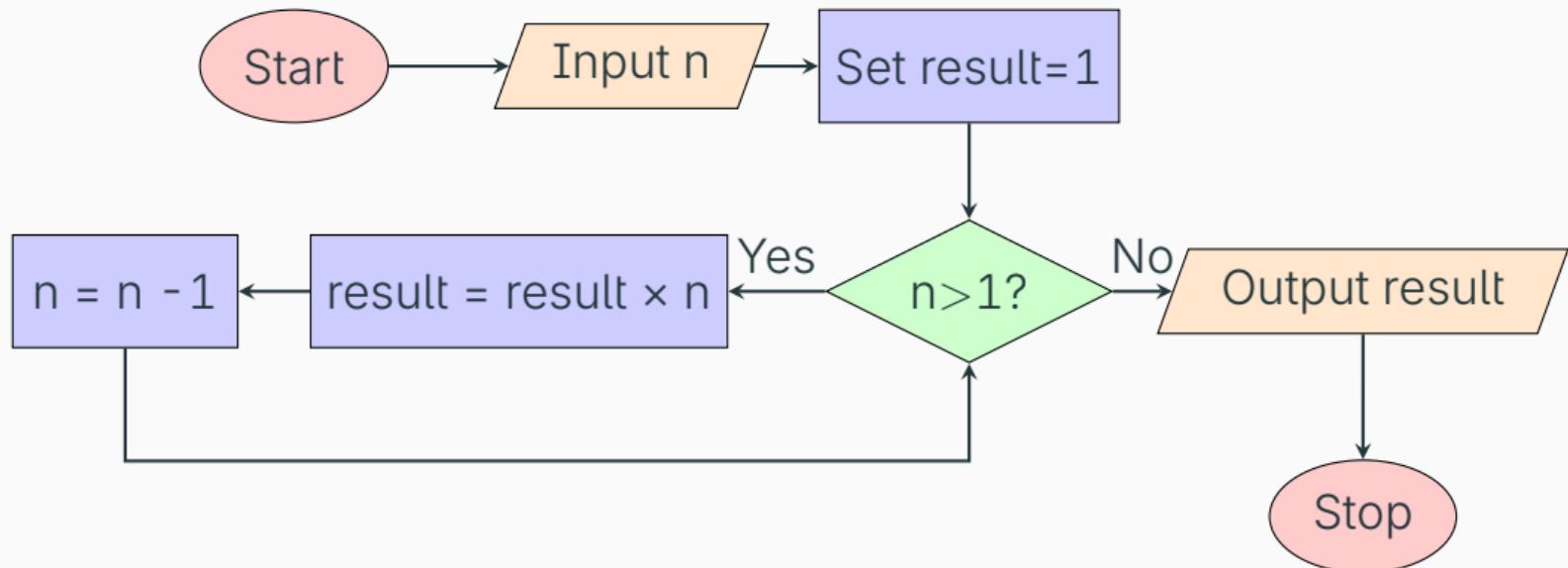
Flowchart Shapes

- **Start/Stop:** ellipse
- **Process:** rectangle
- **Decision:** diamond
- **Input/Output:** parallelogram
- **Sequence:** arrow



Sequence
→

Example Flowchart: Factorial of a Number



Pseudocode

Purpose of Pseudocode

- Represents algorithms in structured, human-readable code
- Independent of programming language, but may include programming key-words
- Easier to understand and refine before coding

Conventions

- Use natural language mixed with structured logic
- Variable names should be consistent and meaningful
- Keywords like Input, If, While, For, Output, Function
- Indentation to show block structure
- Colons indicate the beginning of a block
- Keywords like EndIf, EndWhile, EndFor, EndFunction to indicate the end of a code block
- Pseudocode should be language-independent

Example pseudocode: Area of a Triangle

```
Start
Input base, height
Set area = base * height / 2
Output area
End
```

Example pseudocode: Factorial of a Number

```
Start
Input n
Set result = 1
While n>1:
    result = result * n
    n = n - 1
EndWhile
Output result
End
```

Example Pseudocode: Function for Finding Factorial

The following defines a function named Factorial() with a single input n.

```
Function Factorial(n):
    While n>1:
        result = result * n
        n = n - 1
    EndWhile
    Return result
EndFunction
```

The Return keyword indicates which value to return to the caller of the function. It also marks the end of execution of a function.

Example Pseudocode: Factorial of a Number using Recursion

A recursive function (function that calls itself) named Factorial() is defined that takes a single input:

```
Function Factorial(n):
    If (n==0):
        Return 1
    Else:
        Return n * Factorial(n-1)
    EndIf
EndFunction
```

Note: “a==b” checks whether a is equal to b, returns True if they are equal, otherwise, returns False.

Best Practices

- Keep flowcharts clean and uncluttered
- Use consistent symbols and indentation
- Pseudocode should be language-independent
- Algorithms should be logically ordered and unambiguous

Common Pitfalls

- Overcomplicating flowcharts with too many details
- Ambiguous pseudocode (mixing multiple languages)
- Ignoring edge cases in algorithms
- Writing unstructured logic

Examples

Putting It All Together

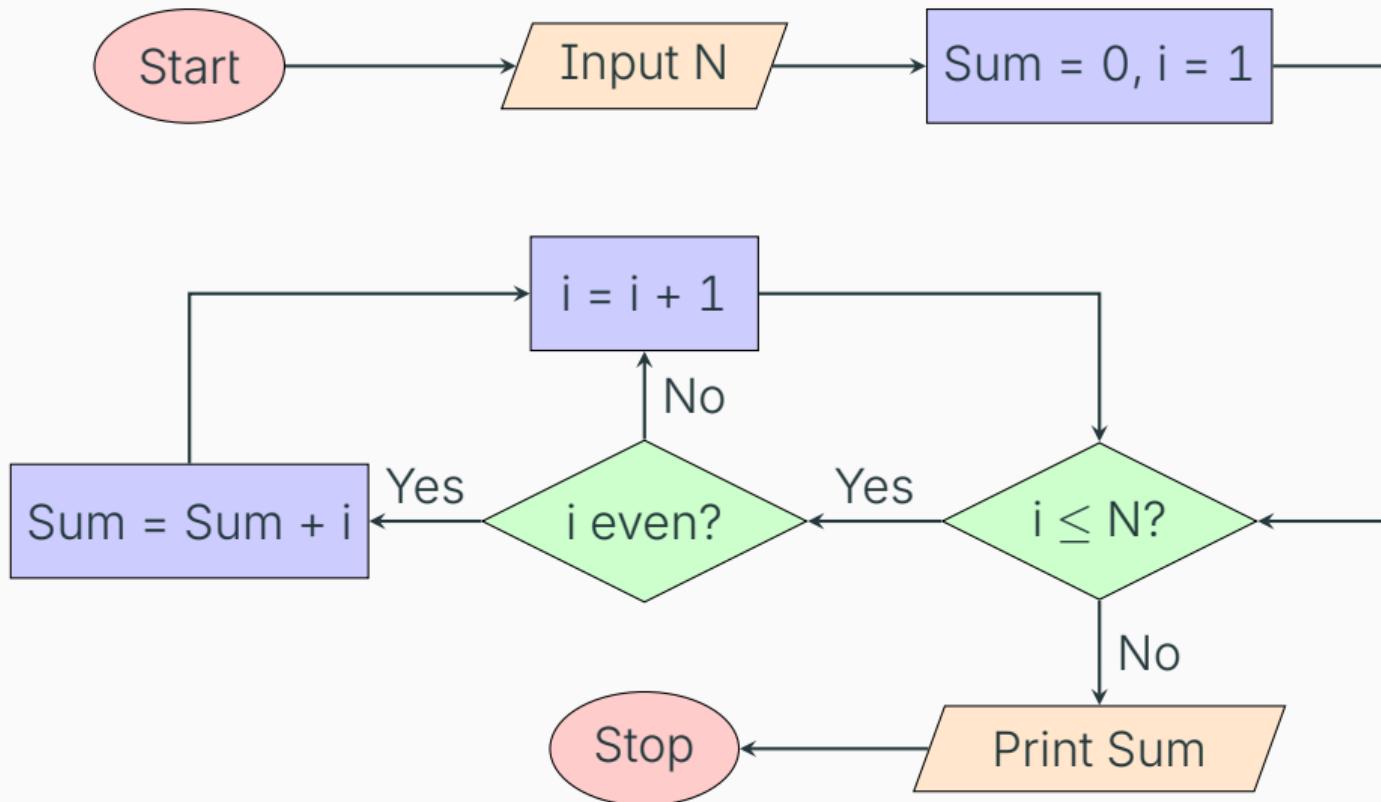
Example task

Compute the sum of all even numbers from 1 to N

Algorithm

- ① Read n
- ② Set sum = 0, i = 1
- ③ While $i \leq n$:
 - ④ If i is even:
 - ▶ add i to sum
 - ⑤ Add 1 to i
- ⑥ Print sum

Flowchart



Pseudocode

```
Start
Input n
sum = 0
i = 1
While (i <= n):
    If (i is even):
        sum = sum + i
    EndIf
    i = i + 1
EndWhile
Output sum
End
```

Example: Find Whether a Number is Even or Odd

```
Start
Input num
If (num mod 2 == 0):
    Output Even
Else
    Output Odd
EndIf
End
```

Note: In the above, $x \text{ mod } y$ returns the remainder when x is divided by y .

Example: Solution of Quadratic Equation

The equation is given as: $ax^2 + bx + c = 0$

Start

Input a, b, c

$x_1 = (-b + \sqrt{b^2 - 4ac}) / 2a$

$x_2 = (-b - \sqrt{b^2 - 4ac}) / 2a$

Output x1, x2

End

Note: In the above, `sqrt(p)` returns the square root of p.

Example: Quadratic Equation: Handling Edge-Case

Sometimes a quadratic equation may not have real-valued solutions.

Start

Input a, b, c

If $(b^2 - 4ac) < 0$:

 Output: No real-valued soltions

Else:

$x_1 = (-b + \sqrt{b^2 - 4ac}) / 2a$

$x_2 = (-b - \sqrt{b^2 - 4ac}) / 2a$

 Output: x_1, x_2

EndIf

End

Example: Quadratic Equation: Handling Edge-Case (cont.)

Sometimes, there might be only one real-valued solution.

Start

Input a, b, c

If ($b^2 - 4ac < 0$):

 Output: No real-valued solutions

ElseIf ($b^2 - 4ac == 0$):

$x = -b / 2a$

 Output: x

Else:

$x_1 = (-b + \sqrt{b^2 - 4ac}) / 2a$

$x_2 = (-b - \sqrt{b^2 - 4ac}) / 2a$

 Output: x_1, x_2

EndIf

End

Exercises

Exercises

- ① Design an algorithm and flow chart to find the largest of the three numbers
- ② Develop pseudocode for computing the sum of the digits of a given integer
- ③ Write an algorithm and pseudocode to check whether a number is prime
- ④ Write a pseudocode for the Euclidean algorithm of finding GCD of two integers
- ⑤ Write a pseudocode for finding the LCM of two integers

Questions?
