ENG 346
Data Structures and Algorithms for Artificial Intelligence
Writing Pseudocodes

Dr. Kürşat İnce

kince@gtu.edu.tr

What is Pseudocode?

- High-level description of an algorithm or program that is designed for human reading.
- It is not tied to any specific programming language, enabling programmers to focus on the logic and flow of the algorithm rather than syntax.

Structure:

- **Keywords**: Used to denote actions (e.g., if, loop, return)
- **Variables**: Represent data elements (e.g., Max, Total)
- **Control Structures**: Indicate how the algorithm flows (e.g., conditions, loops)

Why We Use Pseudocode?

- **Focus on Logic**: Outlines the logic and structure of an algorithm, which helps in visualizing the algorithm's flow.
- **Simplifies Problem-Solving**: From a complex problem into smaller, manageable parts, making it easier to devise a solution.
- **Improves Communication**: Effective communication tool between team members.
- **Facilitates Design and Debugging**: Writing pseudocode allows programmers to think through the algorithm, identify errors or inefficiencies, and make adjustments before implementing the code in a specific programming language.

Characteristics of Good Pseudocode

- Readability and Simplicity:
  - * Clear Expression, * Avoid Complexity, * Consistent Formatting
- Language-Agnostic:
  - * General Syntax, * Common Terminology
- Use of Logical Structures:
  - * Control Flow and Iterations: IF-THEN-ELSE, FOR, WHILE * Statements
- Clarity of Variables and Data Types
  - * Descriptive Naming, * Data Types where necessary.
- Minimalistic Approach: Highlight main logic and essential steps only.
- Use of Comments: In-line Explanations
- Consistency:
  - * Standardized Terminology, * Uniform Style

Basic Constructs in Pseudocode

- Statements: The simplest structure in pseudocode, where instructions are executed one after the other in a linear fashion.
- Conditional Statements: Allows for decision-making in pseudocode, enabling the algorithm to follow different paths based on conditions.
  - IF-THEN-ELSE: Executes one block if the condition is true and another block if it is false.
  - SWITCH/CASE: Allows selection from multiple options based on variable values.
- Loops: Constructs that allow for the execution of a block of code multiple times, either a set number of times or while a certain condition is met.
  - FOR Loop: Repeats a block of code a specific number of times.
  - WHILE Loop: Continues executing as long as a specified condition is true.
  - REPEAT-UNTIL Loop: Executes a block of code repeatedly until a certain condition is met, ensuring the loop runs at least once.

Basic Constructs in Pseudocode

- Variables and Data Types: Variables are used to store data for manipulation. The data types represent the kind of data that can be stored.
  - Integer: Whole numbers (e.g., count, total).
  - Float: Decimal numbers (e.g., average, score).
  - String: Text or character sequences (e.g., name, message).
  - Array/List: A collection of items (e.g., list of scores, names).
  - Other data structure we will learn in this course
- Input and Output: Constructs for receiving data from the user or an external source, and for providing results or feedback.
  - Input Statement: For reading data (e.g., READ input).
  - Output Statement: For displaying results (e.g., PRINT result).
- Input and Output for functions:
  - Input variables: Inputs to the function
  - Function output: RETURN variables

Tips for Writing Effective Pseudocode

- **Use Clear and Descriptive Names**: * Meaningful Variables, * Avoid Abbreviations unless commonly understood
- **Maintain Simplicity and Clarity**: * Avoid overly complex expressions, * Limit Nesting
- **Consistent Formatting**: * Use consistent Indentation and Spacing, * Stick to a consistent style for keywords
- **Be Language-Agnostic**: * Avoid Language-Specific Syntax, * Use Standard Constructs (e.g., IF-THEN, FOR, WHILE)
- **Include Comments Sparingly**: * Use comments to explain complex logic or to clarify crucial points, * Highlight Important Sections and provide context for different parts
- **Break Down Complex Problems**: * Use Modular Approach, i.e. if the algorithm is complex, break it down into subroutines or functions.
- **Be Concise**: * Keep the focus on the logical flow and the solution, * Limit Unnecessary Details

Example Problem

- **Objective**: Write an algorithm that finds the minimum and maximum number from a given list of integers.
- **Input**: A list of integers. The list may contain positive and negative numbers and may be empty.
- **Output**: A tuple of minimum and maximum integer in the list. If the list is empty, the output should be None.
- **Constraints**: The algorithm should handle edge cases such as empty list.

MinMax() Function

```
FUNCTION MinMax(aList):
Input: aList: List of integers
Output: Tuple of min and max of the list
# Check for edge case
   IF aList is empty THEN
      RETURN None  # Found edge case

# Initialize variables
   Min = Max = aList[0]        # Initialize Min and Max to the first element

# Search for min and max in the list
   FOR each Number in aList:
      IF Number < Min THEN
         Min = Number   # Update Min if a smaller number is found
      IF Number > Max THEN
         Max = Number    # Update Max if a larger number is found

   RETURN (Min, Max)     # Return the tuple containing Min and Max
```

Example Problem – 2

- **Objective**: Write an algorithm that takes a sequence of integer values and *determines* if there is a distinct pair of numbers in the sequence whose product is odd.
- **Input**: A list of integers.
- **Output**: True or False
- **Constraints**: The algorithm should handle edge cases such as empty list.

has_odd_product_pair()

```
FUNCTION has_odd_product_pair(aList):
Input: aList: List of integers
Output: True or False

# Initialize a count for odd numbers
    odd_count = 0

# Iterate through the sequence to count odd numbers
    FOR each number IN aList DO:
        IF number MOD 2 != 0 THEN  // Check if the number is odd
            odd_count = odd_count + 1  // Increment the odd counter

# Check for at least two odd numbers
    IF odd_count >= 2 THEN
        RETURN True  # An odd product pair exists

    RETURN False # No distinct pair can produce an odd product
```

is_palindrome() Function

```
FUNCTION is_palindrome(input_string):
Input: input_string: String to check for palindrome
Output: True or False
# Normalize the string: remove spaces and convert to lower case
    normalized_string = REPLACE(input_string, " ", "")  # Remove spaces
    normalized_string = LOWERCASE(normalized_string)      # Convert to lowercase
# Initialize start and end indices
    start = 0
    end = LENGTH(normalized_string) - 1
# Check characters from both ends of the string
    WHILE start < end DO:
        IF normalized_string[start] != normalized_string[end] THEN
            RETURN False   # Not a palindrome if characters do not match
        END IF
        start = start + 1    # Increment start index
        end = end - 1        # Decrement end index
    RETURN True              # The string is a palindrome if all characters match
```