Class Note: Arrays (Session 8)

Overview

Arrays are powerful data structures that allow you to store and manage multiple values of the same data type under a single variable name, ideal for handling lists like inventory quantities or exam marks. This session explores arrays, their declaration, initialization, and operations in pseudocode, with flowchart visualizations to reinforce understanding. By the end of this class, you will be able to declare and manipulate arrays, write pseudocode for common array tasks, and create flowcharts for array-based algorithms, applying structured programming principles like modularity and control structures.

Learning Objectives

- Understand the purpose and structure of arrays.
- Declare and initialize arrays in pseudocode.
- Access and modify array elements using indices.
- Process arrays using loops for operations like summation, searching, or counting.
- Design flowcharts and pseudocode for array-based solutions to practical problems.

Key Concepts

1. **What Are Arrays?**
    - An array is a collection of elements of the same data type, stored under one variable name.
    - Elements are accessed via an **index**, starting at 0 (e.g., `prices[0]` is the first element).
    - Example: An array `marks` with `[70, 85, 90]` has `marks[0] = 70`, `marks[1] = 85`, `marks[2] = 90`.
    - Arrays simplify managing lists compared to separate variables.
2. **Array Declaration and Initialization**
    - **Declaration**: Defines the array's name, size, and data type.
        - Syntax: `DECLARE arrayName[size] OF dataType`

    - Example: `DECLARE prices[6] OF FLOAT` creates an array for 6 floating-point numbers.
  - **Initialization**: Assigns values to elements.
    - Individual: `SET arrayName[index] = value` (e.g., `SET prices[0] = 10.5`).
    - List: `SET arrayName = [value1, value2, ...]` (e.g., `SET prices = [10.5, 20.0, 15.75]`).
  - **Bounds**: Indices range from 0 to size-1 to avoid errors.
3. **Accessing and Modifying Elements**
  - Read an element: `value = arrayName[index]` (e.g., `value = marks[1]`).
  - Update an element: `SET arrayName[index] = newValue` (e.g., `SET marks[1] = 88`).
  - **Bounds Checking**: Accessing invalid indices (e.g., `marks[10]` in a size-5 array) causes errors.
4. **Array Operations with Loops**
  - Loops (for, while) are used to:
    - **Input/Output**: Read or display array elements.
    - **Summation**: Calculate the total of elements.
    - **Searching**: Find an element or its index.
    - **Counting**: Count elements meeting a condition (e.g., values above threshold).
  - Example: A for loop can iterate over indices to process each element.
5. **Structured Programming with Arrays**
  - Use **procedures** to modularize array operations (e.g., a procedure to find the maximum value).
  - Combine arrays with **control structures** (sequence, selection, iteration) for efficient solutions.
  - Apply **top-down design** to break complex problems into smaller tasks (e.g., input, process, output).
6. **Flowchart Representation**
  - **Rectangles**: Array operations (e.g., assign value, sum elements).
  - **Diamonds**: Decisions (e.g., check if element matches a value).
  - **Arrows with Loops**: Iteration over array indices.
  - **Parallelograms**: Input/output of array elements.

Examples

1. **Store and Display Array Elements**

- **Problem**: Input 4 book prices into an array and display them in order.
- **Pseudocode**:

```
DECLARE prices[4] OF FLOAT
START
OUTPUT "Enter 4 prices:"
FOR i = 0 TO 3
    INPUT prices[i]
END FOR
OUTPUT "Book prices:"
FOR i = 0 TO 3
    OUTPUT prices[i]
END FOR
END
```

- **Flowchart** (text-based):

```
[Start] → [DECLARE prices[4]] → [Output "Enter 4 prices:"] → [i = 0] → [i <= 3?] → Yes →
                                                    ↓ No
                                              [Output "Book prices:"] → [i = 0] → [i <= 3?] → Y
                                                                                    ↓ No
                                                                                [End]
```

- **Explanation**: Uses two for loops: one to input prices and another to display them, demonstrating array storage and retrieval.

2. **Calculate Minimum Value with Procedure**
   - **Problem**: Find the minimum value in an array of 5 distances using a procedure.
   - **Pseudocode**:

```
PROCEDURE FindMin(distances[5])
    SET minValue = distances[0]
    FOR i = 1 TO 4
        IF distances[i] < minValue THEN
            minValue = distances[i]
        END IF
    END FOR
    RETURN minValue
END PROCEDURE

DECLARE distances[5] OF FLOAT
START
OUTPUT "Enter 5 distances:"
FOR i = 0 TO 4
    INPUT distances[i]
END FOR
SET minDistance = FindMin(distances)
OUTPUT "Minimum distance: ", minDistance
END
```

- **Flowchart** (main, text-based):

```
[Start] → [DECLARE distances[5]] → [Output "Enter 5 distances:"] → [i = 0] → [i <= 4?] → Y
                                                                      ↓ No
                                                                [Call FindMin] → [Output "Minimum distance: ",
```

  - **FindMin Flowchart**:

```
[Start FindMin] → [minValue = distances[0], i = 1] → [i <= 4?] → Yes → [distances[i]
                                                                      ↓ No
                                                                [Return minValue] → [End FindMin]
```

- **Explanation**: The procedure modularizes finding the minimum, comparing each element to the current minimum.

3. **Linear Search for Element**
   - **Problem**: Search an array of 6 employee IDs for a user-input ID and output whether it's found, using a procedure.
   - **Pseudocode**:

```
PROCEDURE SearchID(ids[6], target)
    FOR i = 0 TO 5
        IF ids[i] == target THEN
            RETURN "Found"
        END IF
    END FOR
    RETURN "Not found"
END PROCEDURE

DECLARE ids[6] OF INTEGER
START
OUTPUT "Enter 6 employee IDs:"
FOR i = 0 TO 5
    INPUT ids[i]
END FOR
INPUT target
SET result = SearchID(ids, target)
OUTPUT result
END
```

- **Flowchart** (main, text-based):

```
[Start] → [DECLARE ids[6]] → [Output "Enter 6 employee IDs:"] → [i = 0] → [i <= 5?] → Yes
                                                                    ↓ No
                                                  [Input target] → [Call SearchID] → [Output result]
```

  - **SearchID Flowchart**:

```
[Start SearchID] → [i = 0] → [i <= 5?] → Yes → [ids[i] == target?] → Yes → [Return "F
                                    ↓ No                                ↓ No
                                [i = i + 1] → [Back to i <= 5?]    [i = i + 1] → [Back
                                    ↓ No
                                [Return "Not found"] → [End SearchID]
```

- **Explanation**: The procedure returns "Found" if the target ID matches an element, otherwise "Not found", using a linear search.

4. **Count Elements Below Average**
   - **Problem**: For an array of 7 rainfall amounts, count how many days had below-average rainfall, using procedures for average and counting.

- **Pseudocode**:

```
PROCEDURE CalculateAverage(rainfall[7])
    SET sum = 0
    FOR i = 0 TO 6
        sum = sum + rainfall[i]
    END FOR
    RETURN sum / 7
END PROCEDURE

PROCEDURE CountBelowAverage(rainfall[7], avg)
    SET count = 0
    FOR i = 0 TO 6
        IF rainfall[i] < avg THEN
            count = count + 1
        END IF
    END FOR
    RETURN count
END PROCEDURE

DECLARE rainfall[7] OF FLOAT
START
OUTPUT "Enter 7 daily rainfall amounts:"
FOR i = 0 TO 6
    INPUT rainfall[i]
END FOR
SET avg = CalculateAverage(rainfall)
SET belowCount = CountBelowAverage(rainfall, avg)
OUTPUT "Average rainfall: ", avg
OUTPUT "Days below average: ", belowCount
END
```

- **Flowchart** (main, text-based):

```
[Start] → [DECLARE rainfall[7]] → [Output "Enter 7 daily rainfall amounts:"] → [i = 0] →
                                                                    ↓ No
                                                          [Call CalculateAverage] → [Call CountBelowAv
```

  - **CalculateAverage Flowchart**:

```
[Start CalculateAverage] → [sum = 0, i = 0] → [i <= 6?] → Yes → [sum = sum + rainfall
                                                  ↓ No
                                          [Return sum / 7] → [End CalculateAverage]
```

- **CountBelowAverage Flowchart**:

```
[Start CountBelowAverage] → [count = 0, i = 0] → [i <= 6?] → Yes → [rainfall[i] < avg
                                                    ↓ No                              ↓
                                            [Return count] → [End CountBelowAverage]
```

- **Explanation**: Uses two procedures for modularity: one to compute the average, another to count elements below it.

Classwork Activities

1. **Pseudocode Writing**:
   - **Task**: Write pseudocode to store 5 temperatures in an array and output the highest temperature using a procedure.
   - **Expected Pseudocode**:

```
PROCEDURE FindMax(temps[5])
    SET maxTemp = temps[0]
    FOR i = 1 TO 4
        IF temps[i] > maxTemp THEN
            maxTemp = temps[i]
        END IF
    END FOR
    RETURN maxTemp
END PROCEDURE

DECLARE temps[5] OF FLOAT
START
OUTPUT "Enter 5 temperatures:"
FOR i = 0 TO 4
    INPUT temps[i]
END FOR
SET highest = FindMax(temps)
OUTPUT "Highest temperature: ", highest
END
```

2. **Flowchart Design**:
   - **Task**: Create a flowchart for a program that inputs 4 product quantities into an array and outputs the total quantity.
   - **Expected Flowchart** (text-based):

```
[Start] → [DECLARE quantities[4]] → [total = 0, i = 0] → [i <= 3?] → Yes → [Input quantit
                                                          ↓ No
                                                [Output "Total quantity: ", total] → [End]
```

3. **Error Correction**:
   - **Task**: Fix errors in this pseudocode:

```
DECLARE values[4] OF INTEGER
SET values = [10, 20, 30, 40]
FOR i = 0 TO 4
    OUTPUT values[i]
END
```

   - **Issues**:

    - Loop bound exceeds array size (`i = 0 TO 3` needed for size 4).
    - Missing `END FOR`.
    - No `START/END`.
- **Corrected Pseudocode**:

```
DECLARE values[4] OF INTEGER
START
SET values = [10, 20, 30, 40]
FOR i = 0 TO 3
    OUTPUT values[i]
END FOR
END
```

4. **Group Activity: Real-World Scenario**:
    - **Task**: In pairs, design pseudocode and a flowchart for a program that:
        - Stores 5 customer ratings (1–5) in an array.
        - Uses a procedure to count ratings of 5.
        - Outputs the count and the average rating.
    - **Example Pseudocode**:

```
PROCEDURE CountFives(ratings[5])
    SET count = 0
    FOR i = 0 TO 4
        IF ratings[i] == 5 THEN
            count = count + 1
        END IF
    END FOR
    RETURN count
END PROCEDURE

PROCEDURE CalculateAverage(ratings[5])
    SET sum = 0
    FOR i = 0 TO 4
        sum = sum + ratings[i]
    END FOR
    RETURN sum / 5
END PROCEDURE

DECLARE ratings[5] OF INTEGER
START
OUTPUT "Enter 5 ratings (1-5):"
FOR i = 0 TO 4
    INPUT ratings[i]
END FOR
SET fiveCount = CountFives(ratings)
SET avg = CalculateAverage(ratings)
OUTPUT "Number of 5-star ratings: ", fiveCount
OUTPUT "Average rating: ", avg
END
```

- **Discussion**: Share solutions and discuss how arrays improve efficiency over individual variables.

Objective Questions (Multiple Choice)

Test your understanding of arrays with the following questions:

1. **What is the purpose of an array?**

- A. To store a single value
- B. To store multiple values of the same type
- C. To create a loop
- D. To define a procedure
- **Answer**: B
- **Explanation**: Arrays store multiple elements of the same data type under one name, unlike a single variable (A). Loops (C) and procedures (D) are unrelated.

2. **What is the valid index range for an array of size 5?**
   - A. 1 to 5
   - B. 0 to 4
   - C. 0 to 5
   - D. 1 to 4
   - **Answer**: B
   - **Explanation**: For a size-5 array, indices range from 0 to 4 (size-1). Other options include invalid or incomplete ranges.
3. **How do you access the third element of an array named `data`?**
   - A. data[3]
   - B. data[2]
   - C. data[1]
   - D. data[0]
   - **Answer**: B
   - **Explanation**: The third element is at index 2 (since indices start at 0), so `data[2]`.
4. **What does this pseudocode output?**

```
DECLARE numbers[3] OF INTEGER
SET numbers = [5, 10, 15]
OUTPUT numbers[0]
```

   - A. 5
   - B. 10
   - C. 15
   - D. All elements
   - **Answer**: A
   - **Explanation**: `numbers[0]` accesses the first element, which is 5. Only one element is output.
5. **What is wrong with this pseudocode?**

```
DECLARE scores[3] OF INTEGER
FOR i = 1 TO 3
    INPUT scores[i]
END FOR
```

- A. Missing OUTPUT
- B. Incorrect loop bounds
- C. Invalid array size
- D. No procedure
- **Answer**: B
- **Explanation**: For a size-3 array, indices are 0 to 2, so the loop should be `i = 0 TO 2`. Other options are not required or incorrect.

6. **What does this procedure do?**

```
PROCEDURE CountPositive(values[4])
    SET count = 0
    FOR i = 0 TO 3
        IF values[i] > 0 THEN
            count = count + 1
        END IF
    END FOR
    RETURN count
END PROCEDURE
```

- A. Sums positive values
- B. Counts positive values
- C. Finds the maximum
- D. Returns all values
- **Answer**: B
- **Explanation**: The procedure counts elements > 0. It doesn't sum (A), find the maximum (C), or return all values (D).

7. **In a flowchart, how is an array loop typically shown?**
    - A. Diamond for each element
    - B. Rectangle with looping arrows
    - C. Oval for indices
    - D. Parallelogram for decisions

- **Answer**: B
- **Explanation**: Loops use rectangles for actions and arrows to loop back to a decision (e.g., `i <= size?`). Diamonds are for decisions (A), ovals for start/end (C), and parallelograms for input/output (D).

8. **What is the output if `values = [10, 20, 30, 40, 50]`?**

```
DECLARE values[5] OF INTEGER
SET values = [10, 20, 30, 40, 50]
SET sum = 0
FOR i = 0 TO 4
    sum = sum + values[i]
END FOR
OUTPUT sum
```

   - A. 150
   - B. 50
   - C. 10
   - D. 5
   - **Answer**: A
   - **Explanation**: The loop sums `10 + 20 + 30 + 40 + 50 = 150`. Other options miscalculate.

9. **Why use procedures with arrays?**
   - A. To change array size
   - B. To make code modular and reusable
   - C. To avoid loops
   - D. To store different data types
   - **Answer**: B
   - **Explanation**: Procedures enhance modularity and reusability for array operations. Array size is fixed (A), loops are often needed (C), and arrays store one data type (D).

10. **What does this pseudocode do if `items = [1, 3, 5, 7]`?**

```
DECLARE items[4] OF INTEGER
SET items = [1, 3, 5, 7]
SET target = 5
FOR i = 0 TO 3
    IF items[i] == target THEN
        OUTPUT "Found"
    END IF
END FOR
```

- A. Outputs all elements
- B. Outputs "Found"
- C. Outputs 5
- D. Outputs nothing
- **Answer**: B
- **Explanation**: The loop finds `5` at index 2 and outputs "Found". It doesn't output all elements (A), the value 5 (C), or nothing (D).

Homework

1. **Pseudocode Practice**:
   - Write pseudocode to store 6 test scores in an array and output the lowest score using a procedure.
2. **Flowchart Creation**:
   - Design a flowchart for a program that inputs 5 sales amounts into an array and outputs their average.
3. **Real-World Application**:
   - Describe a scenario where arrays are used (e.g., managing student attendance). Write a short paragraph and pseudocode with a procedure to count specific values (e.g., present days).
4. **Debugging Challenge**:
   - Fix this pseudocode:

```
DECLARE marks[4] OF INTEGER
SET marks = [50, 60, 70, 80]
FOR i = 0 TO 4
    OUTPUT marks[i]
END FOR
```

    - Issues: Loop bound exceeds array size (`i = 0 TO 3` needed), no `START/END`.
    - Provide the corrected version.

Additional Notes

- **Teaching Tips**:
    - **Duration**: 2–3 class periods (3–4.5 hours).
        - **Period 1**: Array basics, Example 1, Activity 1.
        - **Period 2**: Array operations, Examples 2 and 3, Activities 2 and 3.
        - **Period 3 (optional)**: Example 4, Activity 4, review, quiz.
    - **Visual Aids**: Use diagrams to show array indices (e.g., boxes labeled 0 to n-1).
    - **Engagement**: Relate arrays to real-world lists (e.g., prices, ratings).
    - **Differentiation**:
        - Beginners: Focus on input/output and simple operations.
        - Advanced: Preview sorting algorithms or 2D arrays.
- **Resources**:
    - Flowchart tools: Draw.io, Lucidchart, or paper templates.
    - Text editor for pseudocode: Notepad++, Visual Studio Code.
- **Assessment**:
    - Use objective questions for quizzes or homework.
    - Grade classwork for correct array usage, loop bounds, and flowchart logic.
- **Extension**:
    - Implement a pseudocode example in Python:

```
def find_min(distances):
    min_val = distances[0]
    for i in range(1, len(distances)):
        if distances[i] < min_val:
            min_val = distances[i]
    return min_val

distances = []
for i in range(5):
    distances.append(float(input("Enter distance: ")))
min_dist = find_min(distances)
print("Minimum distance:", min_dist)
```

- **Connections**:
  - Builds on Sessions 6 (loops) and 7 (structured programming) for array processing.
  - Prepares for Session 9 (Subroutines) by emphasizing data structures and modular design.