Pseudocode Keywords and Terms Reference

Table of Contents

1. Basic Structure
2. Data Types and Variables
3. Input/Output Operations
4. Control Structures
5. Loop Constructs
6. Functions and Procedures
7. Array Operations
8. String Operations
9. Mathematical Operations
10. Logical Operations
11. Comparison Operations
12. Data Structure Operations
13. Common Conventions

Basic Structure

Program Structure

```
ALGORITHM algorithm_name
BEGIN
    // Algorithm steps
END
```

Comments

```
// Single line comment
/* Multi-line comment */
```

Variable Declaration

```
DECLARE variable_name AS data_type
DECLARE variable_name AS data_type ← initial_value
```

Data Types and Variables

Basic Data Types

- `INTEGER` - Whole numbers
- `REAL` / `FLOAT` - Decimal numbers
- `STRING` / `TEXT` - Text data
- `BOOLEAN` - True/False values
- `CHARACTER` / `CHAR` - Single character

Declaration Examples

```
DECLARE age AS INTEGER
DECLARE name AS STRING ← "John"
DECLARE price AS REAL ← 19.99
DECLARE isValid AS BOOLEAN ← TRUE
DECLARE grade AS CHARACTER ← 'A'
```

Constants

```
CONSTANT PI ← 3.14159
CONSTANT MAX_SIZE ← 100
```

Input/Output Operations

Input Operations

```
INPUT variable_name
READ variable_name
GET variable_name
PROMPT "Enter value: " AND READ variable_name
```

Output Operations

```
OUTPUT variable_name
PRINT variable_name
DISPLAY variable_name
WRITE variable_name
SHOW variable_name
```

Formatted Output

```
PRINT "Hello, " + name
OUTPUT "Age: ", age
DISPLAY "Result: " + result
```

Control Structures

Conditional Statements

If Statement

```
IF condition THEN
    // statements
END IF

IF condition THEN
    // statements
ELSE
    // statements
END IF

IF condition1 THEN
    // statements
ELSE IF condition2 THEN
    // statements
ELSE
    // statements
END IF
```

Case/Switch Statement

```
SELECT CASE variable
    CASE value1:
        // statements
    CASE value2:
        // statements
    DEFAULT:
        // statements
END SELECT

SWITCH variable
    CASE value1:
        // statements
        BREAK
    CASE value2:
        // statements
        BREAK
    DEFAULT:
        // statements
END SWITCH
```

Loop Constructs

For Loop

```
FOR counter FROM start TO end DO
    // statements
END FOR

FOR counter FROM start TO end STEP increment DO
    // statements
END FOR

FOR counter ← start TO end DO
    // statements
END FOR
```

While Loop

```
WHILE condition DO
    // statements
END WHILE

WHILE condition
    // statements
END WHILE
```

Do-While Loop

```
DO
    // statements
WHILE condition

REPEAT
    // statements
UNTIL condition
```

For-Each Loop

```
FOR EACH element IN collection DO
    // statements
END FOR

FOR element IN collection DO
    // statements
END FOR
```

Loop Control

```
BREAK       // Exit loop
CONTINUE    // Skip to next iteration
EXIT        // Exit loop
```

Functions and Procedures

Function Definition

```
FUNCTION function_name(parameter1, parameter2) RETURNS data_type
    // statements
    RETURN value
END FUNCTION

FUNCTION function_name(parameter1 AS type, parameter2 AS type) → return_type
    // statements
    RETURN value
END FUNCTION
```

Procedure Definition

```
PROCEDURE procedure_name(parameter1, parameter2)
    // statements
END PROCEDURE

SUBROUTINE subroutine_name(parameters)
    // statements
END SUBROUTINE
```

Function/Procedure Call

```
CALL procedure_name(arguments)
result ← function_name(arguments)
SET result TO function_name(arguments)
```

Array Operations

Array Declaration

```
DECLARE array_name AS ARRAY[size] OF data_type
DECLARE array_name AS ARRAY[1..10] OF INTEGER
DECLARE matrix AS ARRAY[1..3, 1..3] OF REAL
```

Array Access

```
array_name[index] ← value
value ← array_name[index]
SET array_name[index] TO value
```

Array Operations

```
LENGTH(array_name)          // Get array length
SIZE(array_name)            // Get array size
SORT(array_name)            // Sort array
REVERSE(array_name)         // Reverse array
```

String Operations

String Functions

```
LENGTH(string)              // Get string length
SUBSTRING(string, start, end)   // Extract substring
CONCATENATE(string1, string2)   // Join strings
UPPER(string)               // Convert to uppercase
LOWER(string)               // Convert to lowercase
TRIM(string)                // Remove whitespace
FIND(string, substring)     // Find position of substring
REPLACE(string, old, new)   // Replace text
```

String Comparisons

```
string1 = string2           // Equal
string1 ≠ string2           // Not equal
string1 < string2           // Lexicographically less
string1 > string2           // Lexicographically greater
```

Mathematical Operations

Arithmetic Operators

```
+       // Addition
-       // Subtraction
*       // Multiplication
/       // Division
MOD     // Modulus (remainder)
DIV     // Integer division
^       // Exponentiation
**      // Alternative exponentiation
```

Mathematical Functions

```
ABS(x)          // Absolute value
SQRT(x)         // Square root
POWER(x, y)     // x raised to power y
MAX(x, y)       // Maximum value
MIN(x, y)       // Minimum value
ROUND(x)        // Round to nearest integer
FLOOR(x)        // Round down
CEILING(x)      // Round up
RANDOM()        // Random number 0-1
```

Logical Operations

Logical Operators

```
AND     // Logical AND
OR      // Logical OR
NOT     // Logical NOT
XOR     // Exclusive OR
```

Boolean Values

```
TRUE    // Boolean true
FALSE   // Boolean false
```

Logical Expressions

```
condition1 AND condition2
condition1 OR condition2
NOT condition
(condition1 AND condition2) OR condition3
```

Comparison Operations

Comparison Operators

```
=       // Equal to
≠       // Not equal to
<>      // Not equal to (alternative)
<       // Less than
>       // Greater than
≤       // Less than or equal to
>=      // Greater than or equal to (alternative)
≥       // Greater than or equal to
<=      // Less than or equal to (alternative)
```

Comparison Examples

```
IF age >= 18 THEN
    // statements
END IF

WHILE count < 10 DO
    // statements
END WHILE
```

Data Structure Operations

Stack Operations

```
PUSH(stack, element)        // Add to top
POP(stack)                  // Remove from top
PEEK(stack)                 // View top element
ISEMPTY(stack)              // Check if empty
```

Queue Operations

```
ENQUEUE(queue, element)     // Add to rear
DEQUEUE(queue)              // Remove from front
FRONT(queue)                // View front element
ISEMPTY(queue)              // Check if empty
```

List Operations

```
INSERT(list, index, element)    // Insert at position
DELETE(list, index)             // Delete at position
APPEND(list, element)           // Add to end
PREPEND(list, element)          // Add to beginning
SEARCH(list, element)           // Find element
```

Common Conventions

Assignment Operations

```
variable ← value            // Assignment
SET variable TO value       // Alternative assignment
variable := value           // Alternative assignment
LET variable = value        // Alternative assignment
```

Increment/Decrement

```
INCREMENT variable          // Add 1
DECREMENT variable          // Subtract 1
variable ← variable + 1     // Manual increment
variable ← variable - 1     // Manual decrement
```

Null/Empty Checks

```
IF variable IS NULL THEN
IF variable IS NOT NULL THEN
IF variable IS EMPTY THEN
IF variable IS NOT EMPTY THEN
```

Exception Handling

```
TRY
    // statements
CATCH exception_type
    // error handling
FINALLY
    // cleanup
END TRY
```

Memory Management

```
ALLOCATE memory_block
DEALLOCATE memory_block
NEW object_type
DELETE object
```

Algorithm Patterns

Sequential Processing

```
BEGIN
    Step 1
    Step 2
    Step 3
END
```

Selection Pattern

```
IF condition THEN
    Process A
ELSE
    Process B
END IF
```

Iteration Pattern

```
WHILE condition DO
    Process
    Update condition
END WHILE
```

Recursion Pattern

```
FUNCTION recursive_function(parameters)
    IF base_case THEN
        RETURN base_value
    ELSE
        RETURN recursive_function(modified_parameters)
    END IF
END FUNCTION
```

Style Guidelines

Naming Conventions

- Use descriptive names: `student_count` instead of `sc`
- Use snake_case or camelCase consistently
- Constants in UPPER_CASE: `MAX_STUDENTS`
- Functions/procedures start with verb: `calculate_average`

Indentation

- Use consistent indentation (2-4 spaces)
- Align nested structures properly
- Use blank lines to separate logical sections

Comments

- Explain complex logic
- Describe purpose of functions
- Document assumptions and constraints

Quick Reference Table

Operation	Keywords	Example
Assignment	`←`, `SET TO`, `:=`	`x ← 5`
Input	`INPUT`, `READ`, `GET`	`INPUT name`
Output	`OUTPUT`, `PRINT`, `DISPLAY`	`PRINT result`
Condition	`IF...THEN...ELSE`	`IF x > 0 THEN`
Loop	`FOR`, `WHILE`, `REPEAT`	`FOR i FROM 1 TO 10`
Function	`FUNCTION...RETURNS`	`FUNCTION add(a,b) RETURNS INTEGER`
Array	`ARRAY[size] OF type`	`DECLARE arr AS ARRAY[10] OF INTEGER`

Advanced Pseudocode Constructs

Object-Oriented Concepts

```
CLASS class_name
    PROPERTIES
        property1 AS data_type
        property2 AS data_type

    METHODS
        CONSTRUCTOR(parameters)
            // initialization
        END CONSTRUCTOR

        METHOD method_name(parameters) RETURNS return_type
            // method body
        END METHOD

        DESTRUCTOR
            // cleanup
        END DESTRUCTOR
END CLASS

// Object creation and usage
DECLARE object AS class_name
object ← NEW class_name(parameters)
object.method_name(arguments)
value ← object.property1
```

Inheritance

```
CLASS child_class INHERITS parent_class
    // additional properties and methods

    OVERRIDE METHOD method_name(parameters)
        // overridden implementation
    END METHOD
END CLASS
```

File Operations

```
OPEN file_name FOR READ/write/append AS file_handle
READ line FROM file_handle
WRITE data TO file_handle
CLOSE file_handle

// File processing pattern
OPEN "data.txt" FOR READ AS input_file
WHILE NOT EOF(input_file) DO
    READ_line ← READ input_file
    PROCESS read_line
END WHILE
CLOSE input_file
```

Error Handling

```
TRY
    // risky operations
    OPEN file FOR READ
    result ← DIVIDE(a, b)
CATCH FileNotFound
    PRINT "File not found error"
CATCH DivisionByZero
    PRINT "Cannot divide by zero"
CATCH GeneralError
    PRINT "An error occurred"
FINALLY
    // cleanup code
    CLOSE file
END TRY
```

Parallel/Concurrent Operations

```
PARALLEL BEGIN
    THREAD 1:
        // operations for thread 1
    THREAD 2:
        // operations for thread 2
END PARALLEL

SPAWN task_name(parameters)
WAIT FOR task_name TO COMPLETE
```

Data Validation Patterns

```
FUNCTION validate_input(input) RETURNS BOOLEAN
    IF input IS NULL OR input IS EMPTY THEN
        RETURN FALSE
    END IF

    IF NOT is_numeric(input) THEN
        RETURN FALSE
    END IF

    IF input < MIN_VALUE OR input > MAX_VALUE THEN
        RETURN FALSE
    END IF

    RETURN TRUE
END FUNCTION
```

Search Algorithms Pattern

```
// Linear Search
FUNCTION linear_search(array, target) RETURNS INTEGER
    FOR i FROM 0 TO LENGTH(array) - 1 DO
        IF array[i] = target THEN
            RETURN i
        END IF
    END FOR
    RETURN -1
END FUNCTION

// Binary Search
FUNCTION binary_search(sorted_array, target) RETURNS INTEGER
    left ← 0
    right ← LENGTH(sorted_array) - 1

    WHILE left ≤ right DO
        mid ← (left + right) DIV 2
        IF sorted_array[mid] = target THEN
            RETURN mid
        ELSE IF sorted_array[mid] < target THEN
            left ← mid + 1
        ELSE
            right ← mid - 1
        END IF
    END WHILE
    RETURN -1
END FUNCTION
```

Sorting Algorithms Pattern

```
// Bubble Sort
PROCEDURE bubble_sort(array)
    n ← LENGTH(array)
    FOR i FROM 0 TO n-2 DO
        FOR j FROM 0 TO n-2-i DO
            IF array[j] > array[j+1] THEN
                SWAP array[j] AND array[j+1]
            END IF
        END FOR
    END FOR
END PROCEDURE
```

Graph Algorithms Pattern

```
// Graph representation
DECLARE graph AS ARRAY[vertices] OF LIST

// Depth-First Search
PROCEDURE dfs(graph, start_vertex, visited)
    visited[start_vertex] ← TRUE
    PRINT start_vertex

    FOR EACH neighbor IN graph[start_vertex] DO
        IF NOT visited[neighbor] THEN
            CALL dfs(graph, neighbor, visited)
        END IF
    END FOR
END PROCEDURE
```

Database Operations

```
CONNECT TO database_name
EXECUTE QUERY "SELECT * FROM table_name WHERE condition"
WHILE MORE_RECORDS DO
    record ← FETCH_NEXT_RECORD()
    PROCESS record
END WHILE
CLOSE CONNECTION
```

Alternative Syntax Variations

Assignment Variations

```
x ← 5           // Arrow assignment
x := 5          // Pascal-style
SET x TO 5      // Explicit assignment
LET x = 5       // BASIC-style
x = 5           // Mathematical equality
```

Loop Variations

```
// For loop variations
FOR i = 1 TO 10
FOR i FROM 1 TO 10
FOR i IN 1..10
FOR i ← 1 TO 10 STEP 1

// While loop variations
WHILE condition DO
WHILE condition
WHILE (condition)

// Repeat variations
REPEAT...UNTIL condition
DO...WHILE condition
```

Conditional Variations

```
// If statement variations
IF condition THEN...END IF
IF (condition) THEN...ENDIF
IF condition:...END

// Case variations
CASE variable OF
    value1: statements
    value2: statements
    OTHERWISE: statements
END CASE

SWITCH variable
    WHEN value1: statements
    WHEN value2: statements
    ELSE: statements
END SWITCH
```

Common Algorithm Notation

Big O Notation References

- `O(1)` - Constant time
- `O(log n)` - Logarithmic time

- `O(n)` - Linear time
- `O(n log n)` - Linearithmic time
- `O(n²)` - Quadratic time
- `O(2ⁿ)` - Exponential time

Mathematical Notation

```
∑ (summation)
∏ (product)
⌊x⌋ (floor function)
⌈x⌉ (ceiling function)
|x| (absolute value)
√x (square root)
```

Set Operations

```
∈ (element of)
∉ (not element of)
∪ (union)
∩ (intersection)
⊆ (subset)
∅ (empty set)
```

Best Practices

Structure Guidelines

1. **Use clear, descriptive names** for variables and functions
2. **Maintain consistent indentation** throughout the algorithm
3. **Group related operations** together
4. **Use comments** to explain complex logic
5. **Keep functions/procedures focused** on a single task

Readability Tips

1. **Start with the main algorithm** outline
2. **Break complex problems** into smaller procedures
3. **Use meaningful variable names** instead of single letters

4. **Include input/output specifications**
5. **Document assumptions and constraints**

Common Mistakes to Avoid

1. **Inconsistent notation** within the same algorithm
2. **Overly complex nested structures** without clear organization
3. **Missing initialization** of variables
4. **Infinite loops** without proper termination conditions
5. **Unclear function parameters** and return values

This reference covers the most commonly used pseudocode keywords and conventions. Different textbooks and organizations may use slight variations in syntax, but the core concepts remain consistent across most pseudocode standards.