PROBATTLE PSEUDOCODE STANDARD

Pseudocode is a kind of structured english for describing algorithms. It allows the designer to focus on the logic of the algorithm without being distracted by details of language syntax. At the same time, the pseudocode needs to be complete. It describe the entire logic of the algorithm so that implementation becomes a rote mechanical task of translating line by line into source code.

In general the vocabulary used in the pseudocode should be the vocabulary of the problem domain, not of the implementation domain. The pseudocode is a narrative for someone who knows the requirements (problem domain) and is trying to learn how the solution is organized.

For example:

Extract the next word from the line *(good)*
set word to get next token *(poor)*

Append the file extension to the name *(good)*
name = name + extension *(poor)*

FOR all the characters in the name *(good)*
FOR character = first to last *(ok)*

Note that the logic must be decomposed to the level of a single loop or decision. Thus "Search the list and find the customer with highest balance" is too vague because it takes a loop AND a nested decision to implement it.

Each individual designer may have their own personal style of pseudocode. Pseudocode is not a rigorous notation, since it is read by other people, not by the computer. There is no universal "standard" for the industry, but for competition purposes it is helpful if we all follow a similar style. The format below is recommended for expressing your solutions during the competition.

It has been proven that three basic constructs for flow of control are sufficient to implement any "proper" algorithm.

SEQUENCE is a linear progression where one task is performed sequentially after another.

WHILE is a loop (repetition) with a simple conditional test at its beginning.

IF-THEN-ELSE is a decision (selection) in which a choice is made between two alternative courses of action.

Although these constructs are sufficient, it is often useful to include three more constructs:

REPEAT-UNTIL is a loop with a simple conditional test at the bottom.

CASE is a multiway branch (decision) based on the value of an expression.

FOR is a "counting" loop.

SEQUENCE

Sequential control is indicated by writing one action after another, each action on a line by itself, and all actions aligned with the same indent. The actions are performed in the sequence (top to bottom) that they are written.

Example

```
READ height of rectangle
READ width of rectangle
COMPUTE area as height times width
```

Common Action Keywords

Several keywords are often used to indicate common input, output, and processing operations.

Input: READ, OBTAIN, GET
Output: PRINT, DISPLAY, SHOW
Compute: COMPUTE, CALCULATE, DETERMINE
Initialize: SET, INIT
Add one: INCREMENT, BUMP

IF-THEN-ELSE

Binary choice on a given Boolean condition is indicated by the use of four keywords: IF, THEN, ELSE, and ENDIF. The general form is:

```
IF condition THEN
        sequence 1
ELSE
        sequence 2
ENDIF
```

The ELSE keyword and "sequence 2" are optional. If the condition is true, sequence 1 is performed, otherwise sequence 2 is performed.

WHILE

The WHILE construct is used to specify a loop with a test at the top. The beginning and ending of the loop are indicated by two keywords WHILE and ENDWHILE. The general form is:

```
WHILE condition
        Sequence
ENDWHILE
```

The loop is entered only if the condition is true. The "sequence" is performed for each iteration. At the conclusion of each iteration, the condition is evaluated and the loop continues as long as the condition is true.

CASE

A CASE construct indicates a multiway branch based on conditions that are mutually exclusive. Four keywords, CASE, OF, OTHERS, and ENDCASE, and conditions are used to indicate the various alternatives. The general form is:

```
CASE expression OF
        condition 1 : sequence 1
        condition 2 : sequence 2
        ...
        condition n : sequence n
        DEFAULT:
        default sequence
ENDCASE
```

The DEFAULT clause with its default sequence is optional. Conditions are normally numbers or characters indicating the value of "expression", but they can be English statements or some other notation that specifies the condition under which the given sequence is to be performed. A certain sequence may be associated with more than one condition.

REPEAT-UNTIL

This loop is similar to the WHILE loop except that the test is performed at the bottom of the loop instead of at the top. Two keywords, REPEAT and UNTIL are used. The general form is:

```
REPEAT
        sequence
UNTIL condition
```

The "sequence" in this type of loop is always performed at least once, because the test is performed after the sequence is executed. At the conclusion of each iteration, the condition is evaluated, and the loop repeats if the condition is false. The loop terminates when the condition becomes true.

FOR

This loop is a specialized construct for iterating a specific number of times, often called a "counting" loop. Two keywords, FOR and ENDFOR are used. The general form is:

```
FOR iteration bounds
        sequence
ENDFOR
```

In cases where the loop constraints can be obviously inferred it is best to describe the loop using problem domain vocabulary.

Example

```
FOR each month of the year (good)
FOR month = 1 to 12 (ok)

FOR each employee in the list (good)
FOR empno = 1 to listsize (ok)
```

NESTED CONSTRUCTS OR PARENTHESIS {}

The constructs can be embedded within each other, and this is made clear by use of indenting. Nested constructs should be clearly indented from their surrounding constructs. Alternatively, parenthesis can be used to clarify nesting constructs.

Example

```
SET total to zero
REPEAT
        READ Temperature
        IF Temperature > Freezing THEN
                INCREMENT total
        END IF
UNTIL Temperature < zero
Print total
```

In the above example, the IF construct is nested within the REPEAT construct, and therefore is indented.

INVOKING SUBPROCEDURES

Use the CALL keyword. For example:

```
CALL AvgAge with StudentAges
CALL Swap with CurrentItem and TargetItem
CALL Account.debit with CheckAmount
CALL getBalance RETURNING aBalance
CALL SquareRoot with orbitHeight RETURNING nominalOrbit
```

FUNCTION DEFINITION

Functions can be defined using the FUNCTION keyword using the following syntax:

```
FUNCTION functionName [output arg] (input argument 1, input argument 2,.., input argument n)
        Statements
END
```

Some sample pseudocode examples:

```
CASE  grade  OF
        A       : points = 4
        B       : points = 3
        C       : points = 2
        D       : points = 1
        F       : points = 0
ENDCASE
```

```
CASE  Title  OF
                Mr      : Print "Mister"
                Mrs     : Print "Missus"
                Miss    : Print "Miss"
                Ms      : Print "Mizz"
                Dr      : Print "Doctor"
ENDCASE
```

```
WHILE employee.type NOT EQUAL manager AND personCount < numEmployees
        INCREMENT personCount
        CALL employeeList.getPerson with personCount RETURNING employee
ENDWHILE
```

```
WHILE Population < Limit
        Compute Population as Population + Births - Deaths
ENDWHILE
```

```
IF HoursWorked > NormalMax THEN
        Display overtime message
ELSE
        Display regular time message
ENDIF
```