Pseudo-Code Standards

Mubashir N. Mir – www.mubashirnabi.com

PART I

How to read the symbols

<> : The angler brackets mean that user must provide the value of whatever is written within these brackets.
[] : The square brackets mean the this is an optional feature. If its not needed, it may be left out.
... : The three dots (ellipses) mean that items similar to previous can come here.
| : A pipe sign shows two or more options (on each side of the sign) either of which can be used.
IF : Anything written in capitol letters is a command and must be written exactly the way it is shown.
' : A single quote is used to show comments about that line of code.

Pseudo-Code Commands

1. Data types

TYPE	PURPOSE
Integer	Whole numbers, negative or positive. (123, -123, 0, 23434, -8 etc.)
Real	Floating point numbers, negative or positive. (-143, 0, 45.67, -12.8987 etc.)
Boolean	Logical. Only TRUE or FALSE.
String	Any character. Can't be used in arithmetic expressions. ("Ali Khan", "14-D/II Gulberg", "-123", "Hello!?!", "2*7/(3-2)" etc.)
Currency	Any negative or positive number with or without decimal. (12343.45, -23, 0, 34333 etc.)
Date	Date in any format. (12/12/2009, 01-03-2007 etc.)

2. Declaring variables

Syntax: DIM <data type> <variable name>
OR
<variable name> : <data type>
OR
<data type> : <variable name>

Ex: DIM integer Roll
StudentName : String
boolean : IsWorking

Note: A variable name (also called an Identifier) must start with either an alphabet or an underscore followed by an alphabet. It must not contain and special characters such as ^, & @ etc. Variable names should be meaningful, such as 'Student_Name' and not 'x' or 'y' etc. Also, try and keep them short so that they are easier to remember.

3. Using variables

a) Hard coding values into variables

```
Ex:     a = 10
        Name= "Ali Khan"
        Price = 345.98
        Average = 45.77
        Total = Price * Quantity        'Total gets the result of Price * Quantity
```

b) Getting values from the user (the Input command)

Syntax: Input <variable>

```
Ex:     Input i
        Input Name
        Input PartNo, Price, Quantity       'Three values will be asked from the user and put into
                                             these variables respectively
```

4. Sending output to the screen

```
Syntax: PRINT <variable>
        PRINT "any string"
        PRINT <variable | constant> <operator> <variable | constant>
```

```
Ex:     PRINT i
        PRINT "This is a message"
        PRINT i + p                         'The result of i+p will be printed
        PRINT "The result is", i / 2        'Two items will be printed, the message in quotes and
                                             the result of i/2
        PRINT                               ' This PRINT command without anything to print will
                                             actually print an empty line
```

Note: In some algorithms OUTPUT command is used instead of PRINT. They are both the same as far as pseudo-code is concern. Just remember not to mix the two. Use only one of them throughout your code.

5. Decisions (branching)

a) Single-line 'IF' statement

Syntax: IF <condition> THEN <statement> [ELSE <statement>]

Note: Only one statement is allowed in both the true and false part. The ELSE part is optional and can be left is nothing is desired in case the condition is found to be false. Note that there is no END IF in single-line IF condition.

```
Ex:     IF a < 10 THEN PRINT "a is less than 10" ELSE PRINT "a is not less than 10"
```

Note: Only one path will be taken depending on the evaluation of the condition. The other one will be ignored.

```
Ex:     IF IsWorking = TRUE THEN PRINT "Something is working"       'The ELSE part is optional. Nothing will
                                                                     be done if the condition in this example
```

is False

b) Block 'IF' structure

```
Syntax: IF <condition> THEN
            <statements>
        [ELSE                                   'This part is optional
            <statements>]
        ENDIF
```

Note: A block 'IF' can be used when more than one statement is required in true and/or false sections. All block 'IF' structures must be terminated with an ENDIF keyword to establish a clear end of the structure.

```
Ex:     IF a < 10 THEN
            PRINT "a is less than 10"
        ELSE
            PRINT "a is not less than 10"
        ENDIF
```

Block 'IF' can be nested inside another block 'IF' structure.

```
Ex:     IF Tea = TRUE THEN
            IF Cookies = TRUE THEN
                PRINT "Tea and cookies will be served"
            ELSE
                PRINT "Only tea will be served"
            ENDIF
        ELSE
            PRINT "No tea can be served"
        ENDIF
```

Note: Indentation is used to visually separate the IF condition with the nested IF conditions. Otherwise, reading the nested structures becomes very difficult.

c) Block 'IF' with 'ELSEIF' clause

```
Syntax: IF <condition> THEN
            <statements>
        [ELSEIF <condition> THEN
            <statements>
            ...
        ELSE
            <statements>]
        ENDIF
```

```
Ex:             IF Tea = TRUE THEN
                    PRINT "Tea will be served"
                ELSEIF Coffee = TRUE THEN
                    PRINT "Coffee will be served"
                ELSE
                    PRINT "Tea and coffee are not available"
                ENDIF
```

6. Iteration (Loops)

a) Counter loop: FOR..NEXT

```
Syntax:   FOR <variable> = <start value> TO <end value> [STEP <step value>]
              <statements>
          NEXT <variable>

Ex:       FOR i = 1 TO 10
              PRINT "This is line number", i
          NEXT i
```

The STEP clause can be used to change the default increment value of 1.

```
Ex:       FOR i = 1 TO 50 STEP 2
              PRINT i
          NEXT i
```

Usually, the start value is less than or equal to the end value of the counter variable. If the start value is greater than the end value, then the STEP clause must be used and the step value must be a negative number. It is also possible to use a variable instead of constants for the start and end values.

```
Ex:       p = 1
          FOR i = 10 TO p STEP -1
              PRINT i
          NEXT i
```

b) Conditional loop: DO WHILE..END DO

```
Syntax:   DO WHILE <condition>
              <statements>
          END DO

Ex:       a = 0
          DO WHILE a < 10
              PRINT a
              a = a + 1
          END DO
```

Make sure that you increment then counter yourself, otherwise this loop will become an endless loop.

Note: This loop also appears in some algorithms in these forms, however, their working is the same:

```
Form 1:   While <condition> Do
              <statements>
          End While

Form 2:   While <condition>
              <statements>
          Loop
```

A bottom-testing version of this loop can also be used:

Syntax:
```
DO
        <statements>
WHILE <condition>
```

Ex:
```
a = 0
DO
        PRINT a
        a = a + 1
WHILE a < 10
```

Unlike the top-testing version, the bottom-testing loop will always run once even if the testing condition is False the first time around.

c) Conditional loop: REPEAT ... UNTIL

Syntax:
```
REPEAT
        <statements>
UNTIL <condition>
```

Ex:
```
a = 1
DO UNTIL a > 10
        PRINT a
END DO
```

Unlike the DO WHILE loop, the REPEAT ... UNTIL loops runs 'until' the condition becomes True.

7. Procedures

Syntax:
```
Procedure <procedure name>(parameter list)
        <statements>
End Procedure
```

OR

```
Procedure <procedure name>
        <statements>
End Procedure
```

Ex:
```
Procedure Sum(x:integer, y:integer)
        Print "Sum of two number is ", x+y
End Procedure

Procedure HelloWorld
        Print "Hello to the World"
End Procedure
```

To call a procedure: <procedure name> OR <procedure name>(parameter list)

Ex: Sum OR Sum(a, 10, c)

Note: Procedures do not run by themselves. It must be called from the main program or another procedure to run. It is also possible to write procedures that do not take any parameters (information sent to the procedure). In case one or more parameters are sent to the procedure, remember the variables that are used to receive the data are local in scope to the procedure. This means that if the calling program send a variable with the name 'a' and the procedure also has a parameter variable named 'a' to receive the data, the two variables are different and change in one will not reflect in the other.

8. Functions

Syntax:
```
Function <function name>(parameter list)
        <statements>
End Function
```

OR

```
Function <function name>
        <statement>
End Function
```

Ex:
```
Function Sum(x:integer, y:integer)
        Return x+y
End Function

Function AMPM
        If hours > 1 and hours < 12 then
                Return "AM"
        Elseif hours > 12 and hours < 24 then
                Return "PM"
        Endif
End Function
```

To call a function: <variable> = <function name>(parameter list)

Ex:
```
p = Sum(a,b)
p = a + Sum(d,f)
If sum(a,b) > 10 then Print "sum is greater"
```

Note: The only difference between a procedure and a function is that a function must return a value to the calling routine.

END OF PART I