Constructing Algorithms and Pseudocoding

This document was originally developed by Professor John P. Russo

Purpose:

- **#** Describe the method for constructing algorithms.
- **#** Describe an informal language for writing PSEUDOCODE.
- **#** Provide you with sample algorithms.

Constructing Algorithms

1) Understand the problem

A common mistake is to start writing an algorithm before the problem to be solved is fully understood. Among other things, understanding the problem involves knowing the answers to the following questions.

a) What is the input list, i.e. what information will be required by the algorithm?

b) What is the desired output or result?

c) What are the initial conditions?

2) Devise a Plan

The first step in devising a plan is to solve the problem yourself. Arm yourself with paper and pencil and try to determine precisely what steps are used when you solve the problem "by hand". Force yourself to perform the task slowly and methodically -- think about <<what>> you are doing. Do not use your own memory -- introduce variables when you need to remember information used by the algorithm.

Using the information gleaned from solving the problem by hand, devise an explicit step-by-step method of solving the problem. These steps can be written down in an informal outline form -- it is not necessary at this point to use pseudo-code.

3) Refine the Plan and Translate into Pseudo-Code

The plan devised in step 2) needs to be translated into pseudo-code. This refinement may be trivial or it may take several "passes".

4) Test the Design

Using appropriate test data, test the algorithm by going through it step by step. Think about ordinary sets of input data as well as those which are "degenerate". If you find errors correct them. If serious errors are found, it may be necessary to scrap your pseudo-code and go back to step 2).

An Informal Language for Writing Algorithms (Pseudocoding)

- An algorithm is a finite series of unambiguous instructions that when given a set of input values produces a desired result.

- A desirable property of an algorithm is that it be easy to translate into a programming language.

- Algorithms are written during the programming process, which involves the steps below:

  Problem analysis
  Algorithm construction
  Translation of algorithm into programming language
  Testing and debugging of the program

- The ability to write an algorithm is one of the most important skills to be learned by students taking this course!

- It is a well known fact of programming life that if the algorithm construction is not done well, a programmer will probably spend excessive amounts of time debugging, i.e. finding and fixing errors in the program.

- There are many ways of expressing algorithms. In this class we will adopt an informal algorithmic language which is powerful, concise, and easy to translate into the Turbo Pascal programming language. We will refer to this language as **pseudocode**. The textbook also discusses pseudocode, as well as flowcharts, which are another way of describing algorithms.

- Before specifying our language formally, we'll look at some examples and discuss a few concepts and definitions that will be used below.

  Example 1: An algorithm that asks for a number, then prints its square.

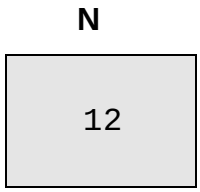
```
print "Enter a number"
read N
print "The square of " N " is " N * N
```

  Example 2: An algorithm that asks for a number, then tells whether it is odd or even.

```
print "Enter a number"
read Number
if Number mod 2 = 0 then
   print N "is even"
else
   print N "is odd"
end if
```

Variables

It is not possible to write useful algorithms without using variables. A variable can be thought of a named memory cell. Variables can be assigned values and later these values can be inspected or referenced. Note that a variable has both a name and a value. Often we will use a diagram such as the one below to convey this idea.



In this case, the name of the variable is "N" and the value of the variable is 12.

Arithmetic Operations

We'll assume that the following arithmetic operations are available:

```
+  -  *  /  div  mod inc dec
```

"div" is integer division. For example,

```
7 div 3 = 2
4 div 5 = 0
```

"mod" is used to find the remainder when one number is divided by another, i.e. **a mod b** is the remainder when a is divided by b.

For example 7 mod 3 = 1 and 4 mod 5 = 4.

"inc" and **"dec"** increment and decrement a variable. After

```
N = 1
inc(N)
```

the value of N will be 2.

Standard mathematical functions such as square root, log, and abs (absolute value) can be used.

Relations and Logical Operations

The following relations can be used

=	equal
<>	not equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal

More sophisticated checks can be made by using the logical operations, AND, OR and NOT. For example, expressions such as

```
A > 0 AND A < 10
```

are allowed.

Pseudo code Statements

At first, our language will have 6 types of statements. These are described below.

1) **Assignment statements** are used to give a values to a variables. Various notation are used for assigning a value to a variable X, including:

```
assign X the value 0
set X to 0
X := 0
X =  0
```

In examples, the last notation is used, but any of the above are acceptable.

Examples of assignment statements:

```
N = 0
N = N + 1
```

2) **Input statements** are used to get information from some external source, such as the users keyboard. Algorithms need a way of getting information to be used. "read X" or "input X are both commonly used to indicate the "reading" or "input" of the next item in the input list into the variable X. We'll use the latter notation.

Examples of input statements.

```
input X
input Number
read Number
```

3) **Output statements** are used to display information determined by an algorithm.

There are two sorts of items which we will need to display.

a) The value of a variable or expression.
b) A string of characters.

Several kinds of notation can be used to display the value of a variable X. These include

```
output X
display X
write X
print X
```

Examples 1:

```
print "Hello"
```

Example 2:

```
print "The value of X is " X
```

4) The **if-then** statement allows allow other statements to be executed conditionally, i.e. if a certain condition is met. The if-then statement is of the form

```
if <condition> then
        <statements to be executed if condition is true>
end if
```

The "end if" serves a delimiter, to mark the end of the statements to be executed if the condition is true.

Example 1:

```
if Sum > 0 then
        print "The sum is positive"
end if
```

Example 2:

```
if N < 0 then
        print "Please enter a positive number"
        input N
end if
```

5) The **if-then-else** statement is a more powerful version of the if-then statement. It is of the form:

```
if <condition> then
        <statements to be executed if condition is true>
else
        <statements to be executed if condition is false>
end if
```

Example:

```
if Y < 0 then
        print "Division by 0 is not allowed"
else
        Quotient = X/Y
        print "The quotient is " Quotient
end if
```

6) The **while statement** allows a group of statements to be repeated while a condition is true. It is of the form:

```
while <condition is true>
        <statements to be executed while condition is true>
end while
```

Example: Algorithm to add numbers entered by user.

```
print "Enter a series of number"
print "Enter a non-positive number to end."
Sum = 0
read N
while N > 0
        print "Adding " N "to " Sum
        Sum = Sum + N
        print "Enter next number "
        read N
end while
print "The sum of the numbers you entered is " Sum
```

7) The **repeat-until statement** allows a group of statements to be repeated until a condition is true. It is of the form

```
repeat
        <statements to be executed until condition is true>
until <condition is true>
```

It is important to note that the statements in the repeat loop body are always executed at least once.

Example:

Algorithm to read integers entered by user and find the sum. Input ends when a non-positive number is entered.

```
print "This algorithm adds numbers that you enter."
print "Enter 0 or negative number to end."
Sum = 0

repeat
        print "Enter next number"
        input N
        Sum = Sum + N
until N <= 0

print "The sum of the numbers you entered is " Sum
```

8) The **for statement** allows a group of statements to be repeated an exact number of times. It is of the form

```
for i = 1 to 10
        <statements to be executed>
end for
```

The loop variable "i" is automatically incremented by 1, every time the loop is executed.

Example:

Algorithm to read 10 integers entered by user and find their sum. Input ends after the 10th. number is entered.

```
print "This algorithm adds 10 numbers that you enter."
Sum = 0

for i = 1 to 10
        print "Enter a number"
        input N
        Sum = Sum + N
end for

print "The sum of the numbers you entered is " Sum
```

Tips On Loop Selection

The REPEAT LOOP is usually best if you are constructing a loop that will execute indefinite number of times, but at least once. (Repeat 1 or more times)

The WHILE LOOP is usually best if you are constructing a loop that will execute indefinite number of times, and possibly not at all. (Repeats 0 or more times)

The FOR LOOP is usually best if you are constructing a loop, and you know exactly how many times the loop has to execute.

More Algorithm Examples

1) An algorithm which finds the sum of a list of 10 numbers.

```
List_Size = 10
Sum = 0
while List_Size > 0
  input N
  Sum = Sum + N
  List_Size = List_Size - 1
end while
print "The sum is " Sum
```

2) An algorithm which asks the user to guess a secret character. If the first guess is incorrect, a clue is revealed and one more chance is given.

```
algorithm Guess_The_Secret_Character
        print  "I'm thinking of a secret character -- what is it? "
        read Guess
        if Guess = "H" or Guess = "h" then
                print "What a guesser!  You are correct!"
        else
                print "The last name of one of your instructors begins with this character."
                print "What is your second guess? "
                read Guess
                if Guess = "H" or Guess = "h" then
                        print "This time you are correct!"
                else
                        print "Sorry, you were wrong again.  The secret character is H"
                end if
        end if
end Guess_The_Secret_Character
```