Pseudocode for Algorithm Specification

What is an Algorithm ?

An algorithm is a sequence of instructions that can be performed by a machine. An algorithm also has clearly defined inputs, and clearly defined outputs.

What is Pseudocode ?

Pseudocode is a notation system for writing algorithms. The pseudocode notation specifies operations that a machine can perform in as human-friendly (e.g., easy to read) way as possible, while avoiding ambiguity.

What notation will we use ?

We will write our algorithms as a sequence of pseudocode statements. Let us start with our notation to represent data objects.

- A **variable** is a name for one or more contiguous memory locations. A variable name must begin with a letter, but make contain digits or an underscore. Names are chosen and capitalization may used to visually help convey the intention of the algorithm writer, and to clarify the type of information stored.

  Examples: `i`, `n`, `Employee_Pay_Rate`

  In the examples above, you don't get too many clues about what `i` or `n` might represent. Any idea what the identifier `Employee_Pay_Rate` might represent ? < LOL >.

- A **scalar variable** is a variable containing just one data item.

- An **array variable** is a name for many data items of the same kind. For example, we might have an array of integers that represent Tiddlywink scores in a campus tournament as illustrated below:

26	11	31	17	43	22	15	18
0	1	2	3	4	5	6	7

  Suppose the name of the array is `A`. We specify a particular element of an array (say the data in position three) with the notation: `A[3]`. In keeping with computer science convention, we start numbering the positions in the array with zero. In the array pictured above `A[0]` is the number 26 and `A[3]` is the number 17.

  The operation specified by the square brackets `[` and `]` is called **array indexing**. The object between the square brackets may also be another variable. For example, the following two statements:

```
i = 5
b = A[i]
```

  result in the variable `b` holding the value 22.

- Comments begin with `//` and go to the end of the line. For example:

```
//  The quick red fox jumped over the lazy brown dog.
```

Comment statements are not executed by the computer. They are only reminders to ourselves.

We now consider simple statements:

- The equals sign `=` is used to assign a value to a variable. For example: `i = 1`. Such a statement is usually called an **assignment statement**. You should think of assignment as the movement of data into the storage location named by the variable. Unlike mathematical notation, the statement is not symmetric. I.e., we never write `1 = i`. Values stored in a variable may be replaced at any time. For example:

  ```
  i = 1
  i = 2
  ```

  results in the value `2` stored in the variable `i`.

- Mathematical expressions are allowed on the right of an assignment. For example:

  ```
  z = x * y + 1
  ```

- We may read a value from an input device. For example, the statement:

  ```
  read x
  ```

  will get a data value from the input device and store it in the variable x.

- We may write a value to an output device. For example, the statement:

  ```
  write x
  ```

  will copy the value of `x` to the output device. The value stored in the memory location named by `x` is not changed.

We now consider something called **control constructs**. To express an algorithm, we must be able to:

- group statements into a block,
- conditionally perform (or not perform) a statement or a block of statements depending on a logical expression.
- and repeat a statement or block of statements.

We now introduce the details of each of these three control constructs.

- **Statement Blocks**

  We will group a set of statements using the curly brace characters "**{**" and "**}**". For example:

```
{
   read celsius_temperature
   fahrenheit = 5.0/9.0 * celsius_temperature + 32
}
```

- **Conditional Execution**

  We can conditionally execute a statement (or a block of statements) using the **if ... else ...** control construct. For example:

```
if ( x < 0 ) {
   x = 0 - x
}
```

  The example above has the effect of taking the absolute value of `x`. I.e., if `x` is initially -5, subtracting from zero gives us positive 5.

  The `else` clause is optional and will be executed whenever the logical condition is false.

- **Repeating a Statement (or a Block of Statements)**

  We can conditionally repeat a statement (or a block of statements) using the `while` construct. For example:

```
t = 1
while ( t <= 5 ) {
   write t
   t = t + 1
}
```

  The example above would output the numbers: 1 2 3 4 5 Notice we use indenting to convey the intended meaning, i.e., that the `write` statement and the statement incrementing `t` is under the control of the `while` construct. A repeated set of statements (including the control construct) is called a **loop**.

Example Algorithm

Input: An integer `n`, and an array `A` containing `n` integers.

Output: The largest integer in the array.

Method:

```
t = A[0]              // A[0] is our first candidate for the largest.
i = 1                 // i is our variable index.
while ( i < n ) {     // Start a loop.
   if ( t < A[i] ) {
      t = A[i]
   }
   i = i + 1
}                     // end while loop
write t               // Output the answer
```

In class we will trace this algorithm with `n = 5` and with the following array:

22	37	11	17	30
0	1	2	3	4