

CST8110: Introduction to Computing

Lab 3: Common Computing Algorithms

Overview

There are several common computation tasks — ones that you will encounter hundreds (perhaps thousands) of times in your computing career.

- Given a sequence of percentages (don't know how many), find the number with the *lowest value*.
- Given a sequence of numbers (don't know how many), find the number with the *highest value*.
- Given a sequence of numbers (don't know how many), find the *average* of those numbers.
- Given two numbers, stored in variables, *exchange* the values stored in those two variables.

I have written a problem statement (assumptions), an algorithm, and a test plan for the first of these. You and your partner will write an algorithm and test plan for the other three.

Problem Statement (Assumptions)

Read the initial problem description. Can you identify requirements that might be needed to solve the problem, but requirements that are not explicitly stated.

Algorithm

Your algorithm is a set of step-wise instructions. You will use a few basic instructions to achieve a problem solution, including:

GET	Gets input values from the user. Stores those input values in some variable.
PUT	Displays heading or prompt (surrounded by double quotes). Displays contents of some variable.
←	Assignment: Take the value on the right side of the arrow and assign it to the variable on the left side of the arrow. Note: Word will automatically generate this single-character symbol when you type the 3-character sequence <--
IF-ENDIF	This is a <i>selection</i> construct. The IF tests the state of some variable. All instructions between the IF and ENDIF are performed if the condition is true.
IF-ELSE-ENDIF	This is another <i>selection</i> construct. The IF tests the state of some variable. All instructions between the IF and ELSE are performed if the condition is true. All instructions between the ELSE and ENDIF are performed if the condition is false.
WHILE-ENDWHILE	This is a repetition (looping) construct. The WHILE tests the state of some variable. All instructions between the WHILE and ENDWHILE are performed if the condition is true. At the end of one iteration of the loop, the WHILE is tested again, and may repeat the instructions.

	(Note: This suggests that something inside the loop must be changing the variable being tested.)
Arithmetic Operators: + - * /	The four traditional arithmetic operators are used to: <i>add</i> , subtract, <i>multiply</i> and <i>divide</i> . Multiply is performed using the asterisk *. Divide is performed using the slash /
Relational Operators: < <= > >= == !=	When defining the test conditions in your IF instructions and your WHILE instructions, you will commonly use these symbols. For example: <i>IF Age >= 16</i>
Boolean Operators: AND OR	When you have multiple parts to a test, you can combine them with the AND OR operators. For example: <i>IF Number < 0 OR Number > 100</i>

Test Plan (First Draft)

You will need to run through your algorithm with sample data several times to check for errors. Each sample run should test one or more of the following issues:

- Out-of-order statements*: Does something happen during a test run that you were not expecting or violated an assumption?
- Incorrect Steps*: Did some individual action do the wrong thing?
- Missing Information*: Did you forget to get information from the user? Are you making an assumption that may not be valid?
- Boundary Conditions*: Are input values supposed to be within a certain range? Did you check for that? How does a repeating (looping) section know when to stop?

Find Lowest Value: Details

This section shows how the evolution of an algorithm might unfold as you seek the proper solution. The first draft of my algorithm is in error. As you work through the material, see if you can see the error (before I point it out later in this document).

There are three phases to the algorithm development:

- Problem Statement (Assumptions)
- Algorithm
- Test Plan

Find Lowest Value: First Draft

Problem Statement (Assumptions): First Draft

- Unknown number of numbers being input.
- Must use a method that allows program user to signal that input has finished.
- Numbers should all be between 0 and 100 (inclusive), but the user might enter something out of range. Need to warn user of input error.
- No character input, only numbers.
- Only need to retain the lowest value.

Algorithm (First Draft)

```

PUT "Enter a positive number (percent) or -1 to end"
GET Number // number to be considered
WHILE Number != -1 // watch for "ending" flag value
    IF Number < CurrentLowestValue THEN
        CurrentLowestValue ← Number // reset lowest
    ENDIF
ENDWHILE
PUT "Lowest:" CurrentLowestValue
    
```

Test Plan (First Draft)

Run through algorithm with sample data several times to check for errors. Each sample run should test one or more of the following issues:

User Input	Result
Enter -1 to test loop termination.	Success: Loop quits if -1 is the first <i>Number</i> entered.
Enter 89 76 57 93 -1	Fails: I never ask for input after the first <i>Number</i> is input. Fix: Add a prompt and <i>GET Number</i> instruction inside the loop.

Find Lowest Value: 1st Refinement

Algorithm (1st Refinement)

```

PUT "Enter a positive number (percent) or -1 to end:"
GET Number // number to be considered
WHILE Number != -1 // watch for "ending" flag value
    IF Number < CurrentLowestValue THEN
        CurrentLowestValue ← Number // reset lowest
    ENDIF
    
```

```

    PUT "Enter another positive number (percent) or -1 to end:"
    GET Number // number to be considered
    
```

```

ENDWHILE
PUT "Lowest:" CurrentLowestValue
    
```

Test Plan (1st Refinement)

User Input	Result
Enter -1 to test loop termination.	Success: Loop quits if -1 is the first <i>Number</i> entered.
Enter 89 76 57 93 -1	Fails: Outputs some completely unexpected value (one that was not input). Problem: When a user enters a reasonable <i>Number</i> (such as 89), and compares it to <i>CurrentLowestValue</i> , the test is essentially meaningless. The variable <i>CurrentLowestValue</i> has no initial defined value in it. Imagine that <i>CurrentLowestValue</i> started out with a low random garbage value (for example, 13). When the <i>IF</i> test is performed, the 13 will already be lower than the 89, so the value 89 will be discarded, and over-written by the next <i>Number</i> that is input. Fix: Give <i>CurrentLowestValue</i> an initial value that is larger than the largest possible legitimate value.

Find Lowest Value: 2nd Refinement

Algorithm (2nd Refinement)

```

CurrentLowestValue ← 999 // higher than highest possible
    
```

```

PUT "Enter a positive number (percent) or -1 to end:"
GET Number // number to be considered
WHILE Number != -1 // watch for "ending" flag value
    IF Number < CurrentLowestValue THEN
        CurrentLowestValue ← Number // reset lowest
    ENDIF
    PUT "Enter another positive number (percent) or -1 to end:"
    GET Number // number to be considered
ENDWHILE
PUT "Lowest:" CurrentLowestValue
    
```

Test Plan (2nd Refinement)

Enter -1 to test loop termination.	Success: Loop quits if -1 is the first <i>Number</i> entered. Fails: Outputs 999 as the lowest number. Fix: Test the value in <i>CurrentLowestValue</i> to see if it is still the initial value.
Enter 89 76 57 93 -1	Success: Outputs 57.

Find Lowest Value: 3rd Refinement

Algorithm (3rd Refinement)

```

CurrentLowestValue ← 999 // higher than highest possible
PUT "Enter a positive number (percent) or -1 to end:"
GET Number // number to be considered
WHILE Number != -1 // watch for "ending" flag value
    IF Number < CurrentLowestValue THEN
        CurrentLowestValue ← Number // reset lowest
    ENDIF
    PUT "Enter another positive number (percent) or -1 to end:"
    GET Number // number to be considered
ENDWHILE
    
```

```

IF CurrentLowestValue == 999 THEN // indicates no valid entries
    PUT "No valid entries"
ELSE
    
```

```

    PUT "Lowest:" CurrentLowestValue
    
```

```

ENDIF
    
```

Test Plan (3rd Refinement)

User Input	Result
Enter -1 to test loop termination.	Success: Loop quits if -1 is the first <i>Number</i> entered. "No valid entries" prompt output correctly.
Enter 89 76 57 93 -1	Success: Outputs 57.
Enter 300 89 34 -1	Fails: The value 300 is accepted, but it is out of range. Fix: Add a range test.
Enter 89 34 -55 -1	Fails: The number -55 is accepted, but it is out of range. Fix: Add a range test.

Find Lowest Value: 4th Refinement

Algorithm (4th Refinement)

```
CurrentLowestValue ← 999 // higher than highest possible
PUT "Enter a positive number (percent) or -1 to end:"
GET Number // number to be considered
WHILE Number != -1 // watch for "ending" flag value
```

```
IF Number > 100 OR Number < 0 THEN
    PUT "Number out of range."
```

```
ELSE IF Number < CurrentLowestValue THEN
    CurrentLowestValue ← Number // reset lowest
ENDIF
PUT "Enter another positive number (percent) or -1 to end:"
GET Number // number to be considered
ENDWHILE
IF CurrentLowestValue == 999 THEN // indicates no valid entries
    PUT "No valid entries"
ELSE
    PUT "Lowest:" CurrentLowestValue
ENDIF
```

Test Plan (4th Refinement)

User Input	Result
Enter -1 to test loop termination.	Success: Loop quits if -1 is the first <i>Number</i> entered.
Enter 89 76 57 93 -1	Success: Outputs 57.
Enter 300 600 999 89 34 -1	Success: Alerts user to invalid number, inputs another (no matter how many times they make the error).
Enter 89 34 -55 -1	Success: Alerts user to invalid number.

Your Task

Complete the following problem solutions. In each case, you must *craft your problem statement (assumptions), write a test plan, and write the algorithm.*

- Given a sequence of numbers (don't know how many), find the number with the *highest value*.
- Given a sequence of numbers (don't know how many), find the *average* of those numbers.
- Given two numbers, stored in variables, *exchange* the values stored in those two variables.

Important Note: You will probably go through multiple refinements as you work through all the algorithms. You do not need to show me the interim versions. When you submit your work, you need only show the final version (including all the successful test cases).

Submission Standards

- Due date posted on my website.
- Cover page:
 - Course Name / Number
 - Lab Number
 - You and your partner's names
 - Date of Submission

- Three algorithms each containing:
 - Problem Statement (Assumptions)
 - Test Plan
 - Algorithm

Submission Process

- Staple all paper components together with the evaluation chart shown below.

Evaluation Criteria

See the attached marking guide.

CST8110: Introduction to Computing Lab 3: Common Computing Algorithms: Evaluation

Your lab assignment has been evaluated using the following criteria:

Find Highest Value

- Problem Statement (Assumptions)
- Test Plan
- Algorithm

Find Average of Values

- Problem Statement (Assumptions)
- Test Plan
- Algorithm

Exchange Values in Two Variables

- Problem Statement (Assumptions)
- Test Plan
- Algorithm
- Format and Organization of Documents

Total /10