

Parsing Delimited Input

Most, if not all, of the programming projects this semester will require the processing of a command file or script that contains lines of the form:

```
<command string><tab><tab-separated command parameters><newline>
```

In addition, the command file specification will denote that lines beginning with a semicolon (` ; `) are comments, which should be ignored. For example, consider the command file **(required name: ParseIn.txt)**:

```
; Test File 01
;
; Test the add command:
add 17 43 29 32
; Test the sort command:
sort foo bar widget kludge
; Test the max and min commands:
max 17 43 29 32
min 17 43 29 32
; Test the match command:
match widget foo bar widget kludge
; Halt processing:
exit
```

For this project, you will implement a program that will parse such a command file, and execute the given commands. The input file is guaranteed to be syntactically correct, so you do not have to worry about error-checking.

The input file may contain an arbitrary number of comment lines and command lines, and the ordering of the commands may be different from the given example.

Commands

For this assignment, you must support the commands described below. In each case, the command string will be followed immediately by a tab, and each command line will be terminated by a single newline character. Command strings are case-sensitive.

`add` <list of integers>

Compute the sum of the integers in the list and print the list and the sum, formatted as in the output sample given below.

`exit`

Terminate execution of your program immediately.

`match` <target string> <list of strings>

Determine whether the given target string occurs as a member of the given list of strings. The list of strings must be decomposed at the embedded tabs. If the target string does appear, print its location in the list (number locations starting at 1, not 0). If the target string does not appear, print an error message. Conform to the output formatting in the output sample.

`max` <list of integers>

Determine the maximum value occurring in the given list of integers, then print the list and the maximum, formatted as in the output sample.

min <list of integers>

Determine the minimum value occurring in the given list of integers, then print the list and the minimum, formatted as in the output sample.

sort <list of strings>

Sort the given list of strings alphabetically, using any sorting algorithm you like. Print the sorted list of strings, as in the output sample.

In the commands described above:

<list of integers> denotes a tab-separated list of integer values

<list of strings> denotes a tab-separated list of string values. Strings may include embedded spaces, but not tabs.

Sample Output:

Here is the output file (required name: `ParseOut.txt`) that corresponds to the input file given on page 1:

```
Programmer: William D McQuain
CS 2604 Homework 1
-----
Sum of { 17, 43, 29, 32} is 121
Sorted {bar, foo, kludge, widget}
Max of { 17, 43, 29, 32} is 43
Min of { 17, 43, 29, 32} is 17
"widget" is word number 3 in {foo, bar, widget, kludge}
-----
```

Since this assignment will be graded automatically, it is important to follow the specified formatting precisely. Read the *Student Guide* to the Curator (URL is below) for a detailed description of how the scoring is actually done.

Design and Documentation

You should design your program so that the parsing and processing logic is as flexible, and reusable, as possible. There is no explicit requirement that you implement classes in this assignment. The parser and the command processor may reasonably be designed as classes, or in a purely procedural manner. The choice is entirely yours.

You should document your implementation in accordance with the Standards Page on the course website. It is possible that your program will be evaluated for documentation as well as for correctness of results. If that is done, your submission that achieved the highest score will be evaluated.

Note well: If you make two or more submissions that are tied for the highest score, the earliest of those will be graded.

Moral: code and document to meet requirements from the beginning, rather than planning to retrofit documentation into a finished program.

What to turn in and how:

Submit your C++ source code file to the Curator System. Instructions for submitting to the Curator are given in the *Student Guide* at the Curator website: <http://ei.cs.vt.edu/~eags/Curator.html>. Be sure to follow those instructions carefully. You will submit your assignments via the URL:

<http://spasm.cs.vt.edu:8080/curator/>

You will be allowed to submit your solution up to five times. Your highest score will be counted.