

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`):

```
; CS 2704 Parsing Homework Input
; The names used below are place names from New Mexico.
; A number of the names DO contain spaces.
;
;
; Let's try a few chop to commands:
;
chop to 7    De Baca
chop to 2    Window Rock
chop to 810  Sandoval
;
; Let's try a longest of command:
;
longest of   Guadalupe  Rio Arriba
;
; Let's try a reverse command:
reverse Rio Arriba
; Let's try a longest of command:
longest of   San Ildefonso Pueblo  Truth or Consequences  Socorro
;
; Time to quit:
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 syntactic 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. In the following command descriptions no string value will contain a tab or newline character. Strings may well contain spaces though.

chop to<tab><length><tab><string>

Print the first `length` characters of the given string. If `length` exceeds the number of characters in the string, print the entire string. Length will never be negative, and the string will never be empty.

reverse<tab><string>

Print the characters of the given string in reverse order.

longest of<tab><string>{<tab><string>}*

Print the longest of the given strings. If two or more strings are tied, print the one that occurs earliest in the list. There will always be at least one string, but there is no upper limit on how many strings may be given.

exit<tab>

Terminate execution of your program immediately, after printing a confirming message.

The input file will contain an `exit` command. It is possible the input file will contain additional data after the `exit` command line; your program will not process that data. A robust design will terminate cleanly even if the `exit` command is missing, but we will not test for that.

Sample Output:

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

```

Programmer: Bill McQuain
CS 2704 Homework 1: Parsing a Script File
-----
Command: chop to      7      De Baca
Chopped:  De Baca
-----
Command: chop to      2      Window Rock
Chopped:  Wi
-----
Command: chop to     810      Sandoval
Chopped:  Sandoval
-----
Command: longest of  Guadalupe  Rio Arriba
Longest:  Rio Arriba
-----
Command: reverse      Rio Arriba
Reversed:  abirrA oiR
-----
Command: longest of  San Ildefonso Pueblo  Truth or Consequences  Socorro
Longest:  Truth or Consequences
-----
Command: exit
Exiting...
-----

```

Note that each command must be echoed to the log file.

Since this assignment will be graded automatically, it is important to follow the specified formatting precisely:

- Spelling and capitalization do matter.
- Spacing between tokens does matter, but only in the following ways.
 - Do not run things together that are separated by whitespace in the output above.
 - Do not separate things with whitespace that are not separated above.
 - Otherwise, the quantity and quality of whitespace does not matter.
- Blank lines do matter. Don't include them where they aren't shown or omit them where they are.

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.

Your submission must be a single uncompressed source file.

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 by one of the TAs, who will assess a deduction (ideally zero) against your score from the Curator.

Note well: if you make two or more submissions that are tied for the highest score, the earliest of those will be graded. There will be absolutely no exceptions to this policy!

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://eags.cs.vt.edu:8080/curator/>

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