ECE244 Programming Fundamentals

Lab Assignment #3: A Command Parser

1. Objectives

The objective of this assignment is to provide you with practice on the use of C++ I/O facilities and the C string library. This assignment requires you to write a command parser that takes input from the terminal or a file, parses this input and verifies that the input is correct. You will be using the functionality of this command parser to exercise several of the subsequent assignments.

2. Problem Statement

You will write a command parser that provides a textual input interface to your program. The parser should take a sequence of commands as input. Each command consists of an operation followed by its arguments. The command and the arguments are separated by white spaces. Your parser should process commands, handle errors in input and print output as described below until the end of input is encountered. All input and output must be performed using C++ I/O facilities. All string manipulation must be performed using the C string library (not the C++ string class).

3. Input, Output and Error Processing

Please read these instructions carefully.

All input should be read via standard input (`cin`). All output and error messages should be printed to standard output (`cout`). All commands should print either one line of output or one error message (but not both). If a command has multiple errors, only the first error during the parsing of the command should be printed. To distinguish output from error messages, error messages MUST begin with "`Error: `" and end with a period ("`.`"). Whenever an error message is printed, the parser should skip all input until the end of the line and then take a new command as input from the next line, i.e., it should ignore this command. When an end-of-file (i.e., end of input) character is pressed then the program must end. The end-of-file character is generated on the ECF lab machines by pressing CTRL-D on the keyboard.

Recall from Section 2 above that a command consists of an operation and a sequence of zero or more arguments. The next section describes the valid operations and their arguments. An operation is a string. If a user types an invalid operation, then the error message "`Error: unknown command.`" should be printed. Arguments are either numbers or strings. If a number argument is expected and the user does not type a number or the number is too large to fit into an integer (greater than 32 bits on ECF machines), then the number is considered invalid and the error message "`Error: argument is not a number.`" must be printed.

Below, the argument named *`stnum`* is an integer and it MUST be printed as a 9-digit integer. For example "`303`" is printed as "`000000303`". If the user types a valid number (see definition of invalid number above) but the number is larger than 9 digits, then the error message "`Error: <number> is too large.`" should be printed. If the user types a negative number (i.e., number < 0), then the error message "`Error: <number> is`

`negative.`" should be printed. The `number` value in the error message is the actual number.

You must use the `cin` input operator (>>) in this assignment. Do not use `getline` or any function that converts strings to integers or vice versa. You can assume that the input will be provided one command per line. The operation and the arguments will be separated by one or more spaces or tab characters. There may be zero or more space or tab characters before the operation or after the last argument of a command (before the newline character).

4. Commands and Arguments

The commands and their arguments are shown below. All commands are in lower case. The operations are shown in bold and the arguments are shown in italics.

- **`new`** ***`stnum`***. This command should print "`New: stnum`". The ***`stnum`*** argument is described above.

- **`locate`** `st`***`num`***. This command should print "`Locate: stnum`". The ***`stnum`*** argument is described above.

- **`updatename`** ***`stnum lname fname`***. This command should print "`Updatename: stnum fname lname`". It takes three arguments. The ***`stnum`*** argument is described above. The last two arguments are strings. Note the reverse order of the last two arguments in the output. Assume that ***`lname`*** and ***`fname`*** are no more than 80 characters long.

- **`updatemark`** ***`stnum idx mark`***. This command should print "`Updatemark: stnum idx mark`". This command takes three arguments all of which are numbers. The ***`stnum`*** argument is described above. The ***`idx`*** argument must be in the range 0-4 (inclusive), otherwise the error message "`Error: <idx> is out of the range 0-4.`" should be printed. The ***`idx`*** argument should be printed as a single digit. The ***`mark`*** argument must be in the range 0-100 (inclusive), otherwise the error message "`Error: <mark> is out of the range 0-100.`" is printed. The ***`mark`*** argument should be printed as an integer number (it can be 1-3 digits).

- **`delete`** ***`stnum`***. This command should print "`Delete: stnum`". The ***`stnum`*** argument is described above.

- **`printall`**. This command should simply print "`Printall`".

- **`deleteall`**. This command should simply print "`Deleteall`".

The following is an example of input commands, outputs and error messages. The output and error messages are shown in red.

```
% ./Driver
new 123456789
New: 123456789
locatee   5555555
```

```
Error: unknown command.
locate 5555555
Locate: 005555555
locate -20
Error: -20 is negative.
updatename  1234567890 Abdelrahman Tarek
Error: 1234567890 is too large.
updatemark f123456789 Abdelrahman Tarek
Error: argument is not a number.
updatename 123456789 Abdelrahman Tarek
Updatename: 123456789 Tarek Abdelrahman
updatemark 123456789 2 500
Error: 500 is out of the range 0-100.
updatemark 123456789 2 99
Updatemark: 123456789 2 99
updatemark 123456789 10 99
Error: 10 is out of the range 0-4.
updatemark 123456789 4 f9
Error: argument is not a number.
updatemark   123456789 10   f9
Error: 10 is out of the range 0-4.
delete      987654321
Delete: 987654321
printall
Printall
deleteall
Deleteall
(eof)
```

Note that the command "`updatemark 123456789 10 f9`" prints only the first error even though the command has two errors.

5. Procedure

Create a sub-directory called `lab3` in your `ece244` directory, using the mkdir command, and secure it with the chmod command. Make it your working directory. Make a `main` function in the file Driver.cc that calls your command parser function. Write a Makefile that generates an executable called Driver from the Driver.cc file.

6. Deliverables

Your program should be in the file Driver.cpp. Submit this file and the Makefile using the command

```
submitece244f 3 Driver.cpp Makefile
```