CS320 Assignment #2

Purpose

This assignment is designed to familiarize you with C++ programming, tokenizers and parsers.

Requirements

This assignment consists of one major requirements.

1) Developing solutions to the problems below.
2) Your assignments will be checked, submitted and graded electronically.

Problem

1) Create the following C++ programs/libraries. Your programs will follow the naming scheme specified: progX_Y.zzz Where X is the assignment number, Y is the program/library number for the assignment and zzz is the file extension for the particular language we will be writing that code in. For example, the first program for the first assignment will be in the file prog1_1.c (Capitalization matters!). Every driver you write for this class will start by printing a standardized header. The header will look like this:

```
Assignment #X-Y, <NAME>, <EMAIL>
```

   Only programs that have entry points will print the header.

   Your program must use the strings used in the examples provided. You do not have any creative leeway in the prompts or the responses. Most of the grading in this course is done automatically and the autograder is *extremely* unforgiving.

   Please ensure that you commit and push your submission files as well.

   Not all of your programs will be a driver (have an entry point). Some of the programs you write will be libraries for other programs.

prog2_1

Create a C++ Tokenizer class. This Tokenizer class will consist of both a header and a class file. You will create the prog2_1.hpp and prog2_1.cpp

This Tokenizer class will have four public methods. First, the constructor, which will setup/create any internal data structures you need. Second, the destructor, which will delete any internal data structures you created during the lifetime of the object.

Third will be a `void` function that takes a single `string` argument name `Tokenize`. This Tokenize function will tokenize the input string on the space character. Only the following tokens are valid:

push, pop, add, sub, mul, div, mod, skip, save, get and any valid integer.

If an input string contains a potential token that is not one of those values the function should throw a std::exception with the message “Unexpected token: <value>” where <value> is replaced by the unexpected token. (If the input was “foobar 3.14” only the first bad token ‘foobar’ should cause the exception, which should read “Unexpected token: foobar”.

Any valid input should then have all of those tokens saved inside the tokenizer in some sort of queue structure that will return them when requested by the `GetTokens` function.

Fourth will be a vector<string> function that takes no arguments named `GetTokens`. This function will retrieve a single set of input tokens that had previously been passed to the function `Tokenize` in a queue fashion (the first input to Tokenize is the first output from GetTokens). If there are no remaining outputs the function should throw a std::exception with the message “No tokens”.

This Tokenizer class will be used by prog2_1.

prog2_2

Create a driver program that will take a single command line argument which will be the name of a file. Your program should then Tokenize each line of that file. Your program 2_2 should Tokenize all of the input before it outputs anything. If any input line would cause a Tokenization error, print the following error message:

“Error on line <#>: Unexpected token: <value>”

And stop processing the file. No other output should occur.

If there are no tokenization errors, the tokens should be printed to STDOUT comma separated, line by line.

Example compilation:

```
g++ prog2_2.cpp prog2_1.cpp -o prog2_2
```

Example execution:

```
./prog2_2 testfile.txt
```

testfile.txt contents:

```
push 3
foobar 3.14
pop
```

Example run (User input to STDIN highlighted with yellow):

```
Assignment #2-2, Scott Lindeneau, slindeneau@sdsu.edu
Error on line 2: Unexpected token: foobar
```

Example execution:

```
./prog2_2 testfile.txt
```

testfile.txt contents:

```
push 3
push 3.14
pop
```

Example run (User input to STDIN highlighted with yellow):

```
Assignment #2-2, Scott Lindeneau, slindeneau@sdsu.edu
Error on line 2: Unexpected token: 3.14
```

Example execution:

```
./prog2_2 testfile.txt
```

testfile.txt contents:

```
push 3
push 6
pop
```

Example run (User input to STDIN highlighted with yellow):

```
Assignment #2-2, Scott Lindeneau, slindeneau@sdsu.edu
push,3
push,6
pop
```

prog2_3

Create a class called Parser (prog2_3.cpp and prog2_3.hpp). The Parser will have three public functions. The constructor, and destructor as appropriate and the boolean function Parse which will take a single vector<string> argument. The parse function will validate that the input adheres to the following rules. If an input line is valid the function should return true. If it is invalid it should return false.

The following tokens must appear by themselves on a single line:
pop, add, sub, mul, div, mod, skip
Any other line that contains a single token is invalid.

The following tokens must appear on a single line of two tokens, in the correct order:
push <int>
save <int>
get <int>
Any other input is invalid.

prog2_4

Create a driver that tokenizes all of the input lines and then parses all of the input lines. All of the input should be completely tokenized before any parsing should be done. On tokenization error your program should print an error as in prog2_2. On parseing error your program should print: “Parse error line <#>: <line>” and your program should stop executing. When all input in valid it should be printed line by line with commas separating the tokens.

Example compilation:

```
g++ prog2_4.cpp prog2_3.cpp prog2_1.cpp -o prog2_4
```

Example execution:

```
./prog2_4 testfile.txt
```

testfile.txt contents:

```
push 3
pop 3
foobar 3.14
```

Example run (User input to STDIN highlighted with yellow):

```
Assignment #2-4, Scott Lindeneau, slindeneau@sdsu.edu
Error on line 3: Unexpected token: foobar
```

Example execution:
```
./prog2_4 testfile.txt
```

testfile.txt contents:
```
push 3
pop 3
push 6
```

Example run (User input to STDIN highlighted with yellow):
```
Assignment #2-4, Scott Lindeneau, slindeneau@sdsu.edu
Parse error on line 2: pop 3
```

Example execution:
```
./prog2_4 testfile.txt
```

testfile.txt contents:
```
push 3
push 6
pop
```

Example run (User input to STDIN highlighted with yellow):
```
Assignment #2-4, Scott Lindeneau, slindeneau@sdsu.edu
push,3
push,6
pop
```

2) After you have committed your files you will have to grant me developer access to your repository. To do this you must open the project on gitlab.com, and open the Members page that is under the settings menu on the right hand side of the page (It looks like a gear). Add me to your project by typing in my username: slindeneau and make sure that the project access is set to developer.

   You will need to do this for <u>every</u> assignment.

3) The last step involves verifying that everything is working correctly. Please go to: http://cs320.lindeneau.com
   Input your first and last name (as it appears on blackboard), your gitlab username and select the assignment you would like to grade. Verify that all of the parts of your program work as expected.

<u>DO NOT ENTER ANY PASSWORDS</u>

Passwords are not required for cs320.lindeneau.com to function. If your project does not get graded, you have not given correct access to the correct developer account on gitlab.

Additional Details

- You should be able to investigate any issues you have on your own and spend the time necessary to understand what is happening overall.

Late Policy

Late programs will be accepted with a penalty of 5% per day for seven days after due date. Turn in time will be determined by either date of rubric submission OR timestamp on graded files, whichever is later.

Cheating Policy

There is a zero tolerance policy on cheating in this course. You are expected to complete all programming assignments on your own. Collaboration with other students in the course is not permitted. You may discuss ideas or solutions in general terms with other students, but you must not exchange code. (Remember that you can get help from me. This is not cheating, but is in fact encouraged.) I will examine your code carefully. Anyone caught cheating on a programming assignment or on an exam will result in a zero for the class or assignment.