CS320 Assignment #4

Purpose

This assignment is designed to familiarize you with python programming and virtual machines.

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

This assignment is due the night of the final. I EXPECT that you will finish this before then and have sufficient time to study, but I am giving you greater freedom to plan your study time.

1) prog4_1.py will be implemented in Python. In prog4_1.py you will implement the following functions:
    a. Tokenize(str): This function will take in an input string and tokenize it according to the same rules in assignment #2. You do not need to store the tokenized string, instead your function should return the list of tokens if is valid. Otherwise, it should raise a ValueError with the message "Unexpected Token: <token>" as in Assignment #2.
    b. Parse(tokens): This function will take a list of tokens as input that have previously been parsed by the Tokenize function. This will validate the parse rules given during Assignment #2 and will raise a ValueError with the parse error message as appropriate.

    prog4_2.py In python, Implement a StackMachine class. Your stack machine class should have the Execute(tokens) public function and the CurrentLine property (intial value zero). You may implement whatever internal data structures and functions are necessary such that your stack machine is easy for you to implement.

    The Execute(tokens) function should accept a list of tokens that has previously been Tokenized and Parsed correctly. The Execute function will then perform the operation defined in the list of tokens as specified by the operation. (Not all operations will return a value, some will):

    - push # -- Pushes the number onto the stack. Returns None.
    - pop -- Pops the top of the stack. Returns the popped value.
    - add -- Pops two values off the stack, adds them pushes the result. Returns None.
    - sub -- Pops two, subtracts the second from the first, pushes result. Returns None.
    - mul -- Pops two, multiples, pushes result. Returns None.
    - div -- Pops two, divides the first by the second, pushes result. Returns None.

- mod -- Pops two, remainder of first divided by second, pushes result. Returns None.
- skip -- Pops two, if the first value is ZERO, changes the CurrentLine property by the second value. If the first value is not zero, nothing extra occurs. Returns None.
- save # -- Pops one, saves that value for future retrival. Returns None.
- get # -- Pops zero. Gets a previously saved value and pushes it on the stack. A saved value may be gotten multiple times. For example, if the top of the stack is 3, and the 'save 1' operation occurs, the 3 is popped off the stack and saved to some ancillary memory (probably a list). Then whenever a 'get 1' operation is executed that value of 3 is pushed onto the stack until the value stored in memory 1 is replaced with a different 'save 1' operation. (You can think about this as hard array access if you like… because that's what memory really is). Returns None.

Whenever the Execute(tokens) function finishes executing the operation specified by the tokens, the property CurrentLine is incremented by 1.

If, at any time, your program attempts to pop a value that doesn't exist, or get a value that has not previously been saved, raise an IndexError with the message "Invalid Memory Access".

prog4_3.py In python, you are going to implement a driver program. You should import the functions/classes you have implemented in prog4_1.py and prog4_2.py. Your driver should use a main function specification and read the first command line argument as a file. Your program should tokenize and parse all of the lines of the file. You should tokenize all of the lines before you begin parsing (same behavior as assignment #2). Your program should then parse all of the lines. As all of the lines have been tokenized and parsed, they should be stored in an indexable structure so that you can randomly get the tokens for any line by line number (start indexing at 0). Your program should then instantiate a StackMachine class and it should then begin executing operations one line at a time, dictated by the StackMachine CurrentLine property. This should continue until the StackMachine CurrentLine property is equal to or greater than the number of lines in the file (because the first line is index at zero and is line zero internally). If at any time the StackMachine raises an IndexError your program should print the IndexError message along with the line number that currently caused it with the following message: "Line #: '<tokens>' caused Invalid Memory Access." For example: "Line 3: 'pop' caused Invalid Memory Access". If any of the Execution(tokens) calls returns a non-None value, it should be printed on its own line to STDOUT. If at anytime the CurrentLine property becomes negative, print the following message: "Trying to execute invalid line: #" and quit. For example: "Trying to execute invalid line: -5"

When your StackMachine stops running naturally (CurrentLine >= # of lines in input file) your program should print: "Program terminated correctly" and quit.

1) 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.

2) 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.