CSC402 - Assignment #4

Introduction

The goal is to implement an interpreter for an abstract **stack** machine language. Consider the following abstract stack machine instructions:

push <number> - pushes the integer value <number> on the stack.

push <name> - pushes the integer value stored in variable <name> on the stack.

pop - pops the value on the top of the stack and discards it.

print <msg> - pops the value on the top of the stack and prints it to the terminal screen using an *optional* message <msg>.

store <name> - pops the value on the top of the stack and stores it in the variable <name>.

ask <msg> - asks the user for an input value using the *optional* message <msg> and then pushes that value on the stack.

dup - duplicate the value on the top of the stack; pop top of stack → temp, then push temp, and push temp again.

add - pop top of stack → temp1, pop top of stack → temp2, then push value temp2 + temp1.

sub - pop top of stack → temp1, pop top of stack → temp2, then push value temp2 - temp1.

mul - pop top of stack → temp1, pop top of stack → temp2, then push value temp2 * temp1.

div - pop top of stack → temp1, pop top of stack → temp2, then push value temp2 / temp1.

equ - pop top of stack → temp1, pop top of stack → temp2, then push value (temp2 == temp1)?1:0.

leq - pop top of stack → temp1, pop top of stack → temp2, then push value (temp2 <= temp1)?1:0.

jumpT <label> - pop top of stack → temp, if temp != 0 then jump to instruction labeled <label>.

jumpF <**label**> - pop top of stack → temp, if temp == 0 then jump to instruction labeled <label>.

jump <**label**> - jump to instruction labeled <label>.

stop <**msg**> - halts execution with an *optional* message string <msg>.

noop - does nothing.

Notes

- <number>, <name>, <msg>, <label> represent literals generated by the lexer.
- The language allows for C++/Java style '//' comments.
- Execution halts with an error if not enough operands are available on the stack for an operation to complete.
- All instructions have to be followed by a semicolon ';'.
- Comments and messages can be encoded in the lexer by the following rules:

```
def t_COMMENT(t):
    r'//.*'
    pass

def t_MSG(t):
    r'\".*\"'
    t.value = t.value[1:-1] # get rid of quotes
    return t
```

- The language allows for labeled instructions and optional messages for print, ask, and stop instructions, e.g.,

```
L1:
      push 1;
      push 1;
      sub;
      dup;
      print "The value is: ";
      jumpT L1;
      stop "all done";
```

Example Programs

1. ---

```
push 3;
print;
```

2.

```
    push 3;
    push 2;
    add;
    push 2;
    mul;
    print;
```

3.

```
    push 3;
    store x;
    push x;
    print "The value of x is ";
```

4.

```
    ask "Enter an integer value: ";
    dup;
    push 0;
    leq;
    jumpT NEG;
    stop "You entered a positive number";
NEG:
    stop "You entered 0 or a negative number";
```

5.

```
//print out a sequence of integers
    push 10;
L1:
    dup;
    print;
    push 1;
    sub;
    dup;
    jumpT L1;
    stop "all done!";
```

Problems:

1. (50pts) Implement an interpreter for this language (**Hint:** you will essentially have to construct an abstract machine that has stack in the state in addition to a symbol table, label table, and a list of instructions, see the implementation of the exp1bytecode interpreter discussed in class). Show that your interpreter works with the programs in the section of example programs - successful execution of each program is worth 10pts.

2. (Extra Credit - 20pts) Write a program in the stack machine language that asks the user for an input value and then computes and prints the factorial value of that input value. Your program should test to make sure that the input value is a valid value for the factorial computation and if not it should terminate its computation with an appropriate message. Definitions of the factorial computation can be found here:

   http://en.wikipedia.org/wiki/Factorial

Hand in your source code together with a Jupyter Notebook that shows that your programs work. To submit your work create a zip file of your sources and the notebook and submit it through BrightSpace – make sure that you include ALL necessary files, even files that you borrowed from Plipy. Points will be subtracted if files are missing. Assignments submitted in formats other than Jupyter Notebooks and the zip file format will not be graded and a failing grade will be recorded.