Project 2 – CMSC 201 – Spring 2013

Objective: This project will give you a chance to learn about strings, lists, functions, top-down design, and the stack ADT.

Note: This project will take a long time to complete. Please get started early.

Description: You are going to write a program that will ask the user to enter a fully parenthesized expression that has non-negative integer operands and using only + - * / and (). Your program will validate the input, use a stack to solve the fully parenthesized infix expression, convert the infix expression into postfix, and solve the postfix expression using a stack.

FAQ

Q: What does it mean fully parenthesized expression?

A: This is an expression of the form (*operand1* **operator** *operand2*). Where an operand is a number or an operand is itself a fully parenthesized expression. An operator is either + or – or * or /.

Q: What does it mean an infix expression? What is postfix?

A: Infix is how we usually write an expression. It is of the form *operand1* **operator** *operand2.* Postfix is of the form *operand1 operand2* **operator**. In postfix, there is no need for ( ) and the operands are numbers.

Example

Fully parenthesized infix expression: (((56 - 73) + (98 * 4))-(32 - 17))

The same expression in postfix format: 56 73 - 98 4 * + 32 17 - -

---

Q: How can I use a stack to solve a fully parenthesized infix expression?

A: Just scan the expression from left to right. If it is anything other than a), push it onto the stack. When you encounter a), pop from the stack 4 times, do the math and push the value onto the stack. At the end you will have just one value in the stack and that will be the answer.

Q: How can I convert a fully parenthesized infix expression into postfix?

A: Just scan the expression from left to right. If it is a (, ignore it. If it is an operator, push it onto the stack. If it is a number, just add it to the output. If it is a), pop once from the stack and add the operator that just came out of the stack to the output.

Q: How can I solve the postfix expression using a stack.

A: Just scan the expression from left to right. If it is a number, push it in the stack. If it is an operator, pop twice (to get two numbers), do the math, and push the answer onto the stack. At the end you will have just one value in the stack and that will be the answer.

Rules:

1. You should validate your input in this order:

1. Check for words (e.g. hello) or expressions that are not mathematical, or expressions that use operators that are not allowed. Report the error message "Sorry. This program cannot work with this input."

2. Assuming the input passed check #1, now check for an expression where the user puts in + to mean a positive number. Report the error message "Please don't use + for positive numbers"

3. Assuming the input passed check #2, now check for an expression where the user puts in - to mean a negative number. Report the error message "Please avoid using negative numbers"

4. Assuming the input passed check #3, now check for an expression where the user does not have a fully parenthesized expression. Report the error message "Please write a fully parenthesized expression"

Note: The spacing should not matter. Therefore (3+4) is just as valid as (3 + 4) which is just as valid as (3 +4).

Hint: Read about the `eval` function, it can help with your validation.

2. As always, you should have a `main` function, but it should not contain more than 20 lines of code (comments don't count). Most of the code in `main` should just be calls to other functions. This means that your program should use top down design and contains several functions each performing a specific tasks. Your code must have the following functions:

1. `printGreeting` – It should print the greeting and be called in `main`

2. `getInput` - It should be called in `main` and take in no arguments. It should ask the user for the input and validate the input to make sure it is a fully parenthesized expression that has non-negative integer operands and using only + - * / and (). It can call other functions to help it validate the input. You can decide what these other functions should be. `getInput` should return the validated input back to `main`.

3. `evaluateInfix`- It should be called from `main`. It should take in the fully parenthesized expression and evaluate it using a stack.

4. `infixToPostfix`- It should be called from `main`. It should take in the fully parenthesized expression and return it as a postfix expression.

5. `evaluatePostfix` - It should be called from `main`. It should take in the postfix expression and evaluate it using a stack.

The 5 functions above are what you must have in your program. You may have other functions and you may call these other functions from `main` or any other function. You can decide when it would be better to have your expression stored as a string vs. as a list. For example, it may be helpful to have your expression stored in a list and pass a list into the `evaluateInfix`, `infixToPostfix`, and `evaluatePostfix` functions.

3. You must use the Stack.py given below and not add any code to it. However, you should add comments to let us know you understand the code. Therefore, you will submit two files: Stack.py (given below) and Proj2.py (it will have your code in it). The only import statement allowed in Proj2.py is `from Stack import *`

```
class Stack:
    def __init__(self):
        self.theStack =[]


    def top(self):
        if self.isEmpty():
            return "Empty Stack"
        else:
            return self.theStack[-1]


    def isEmpty(self):
        return len(self.theStack) == 0


    def push(self, item):
        self.theStack.append(item)
```

```
    def pop(self):
        if not self.isEmpty():
            temp = self.theStack[- 1]
            del(self.theStack[- 1])
            return temp
        else:
            return "Empty Stack"
```

Sample run (user input is shown in bold and explanation in text box is not part of the program's output)

```
Enter a fully parenthesized expression that has
non-negative integer operands and using only + - * / and ()
```

Greeting

```
Please enter the expression: hello
Sorry. This program cannot work with this input.
```

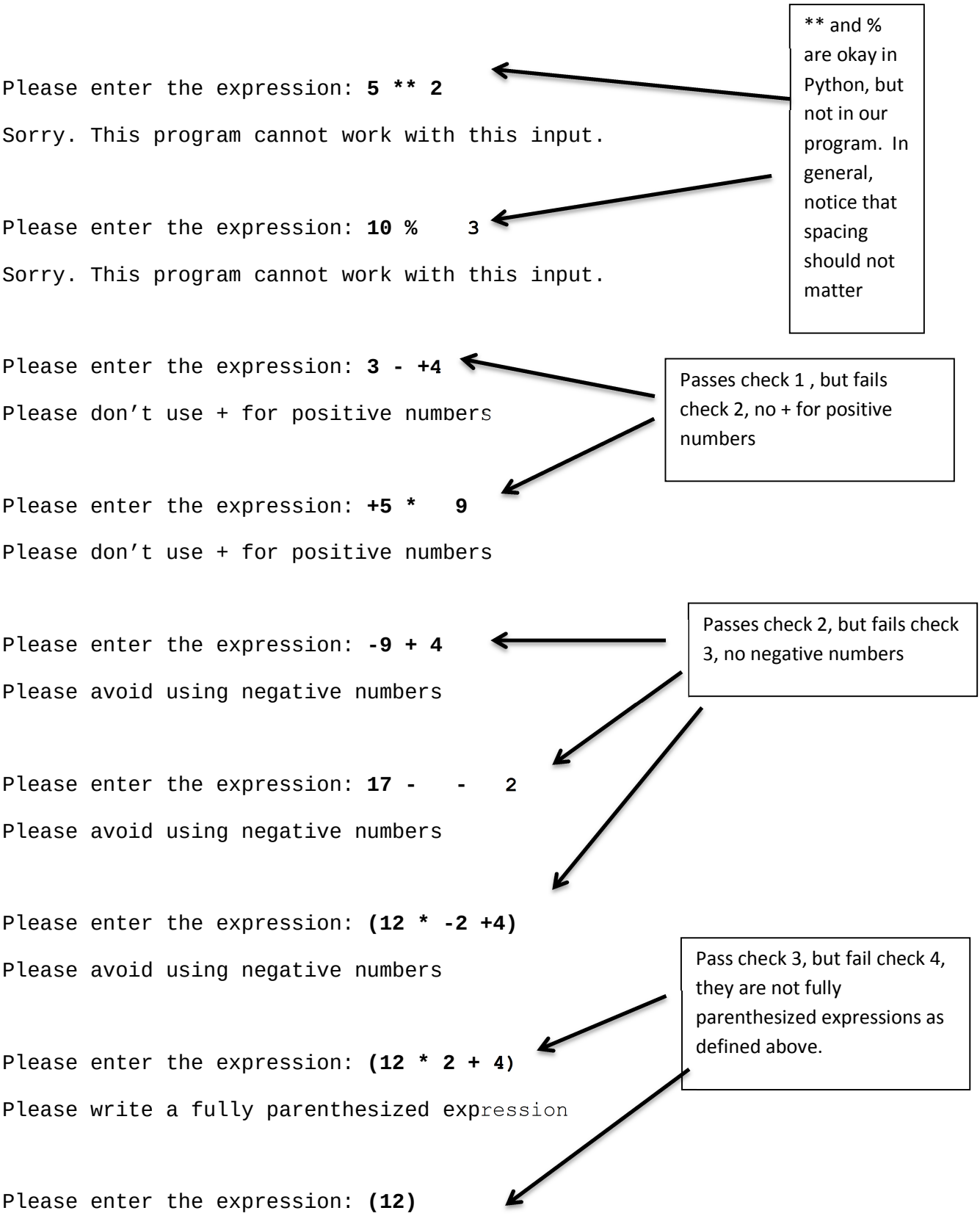
hello is not valid.

```
Please enter the expression: [1,2,3]
Sorry. This program cannot work with this input.
```

Can not work with lists

```
Please enter the expression: (3 + 4
Sorry. This program cannot work with this input.
```

Not a valid mathematical expression since it is missing closing). In general, 3 + 4 would be ok, but not for our program since it is not fully parenthesized


Please enter the expression: 5 ** 2
Sorry. This program cannot work with this input.
** and % are okay in Python, but not in our program. In general, notice that spacing should not matter
Please enter the expression: 10 % 3
Sorry. This program cannot work with this input.
Please enter the expression: 3 - +4
Please don't use + for positive numbers
Passes check 1 , but fails check 2, no + for positive numbers
Please enter the expression: +5 * 9
Please don't use + for positive numbers
Please enter the expression: -9 + 4
Please avoid using negative numbers
Passes check 2, but fails check 3, no negative numbers
Please enter the expression: 17 - - 2
Please avoid using negative numbers
Please enter the expression: (12 * -2 +4)
Please avoid using negative numbers
Pass check 3, but fail check 4, they are not fully parenthesized expressions as defined above.
Please enter the expression: (12 * 2 + 4)
Please write a fully parenthesized expression
Please enter the expression: (12)
Please write a fully parenthesized expression

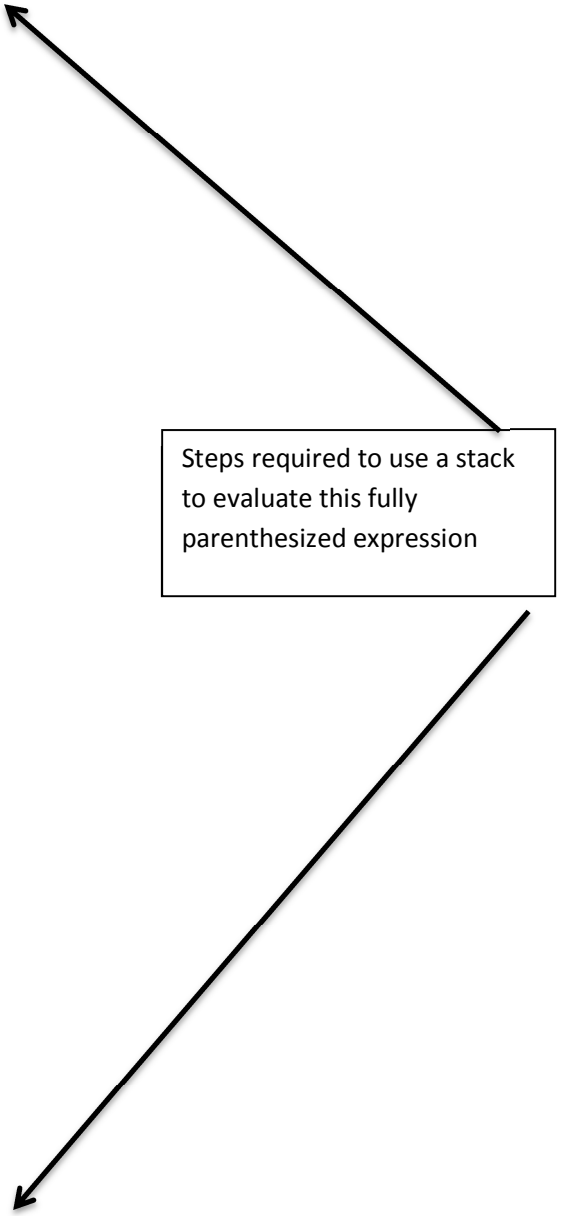
```
Please enter the expression: (12 *2) + 4
Please write a fully parenthesized expression

Please enter the expression: (((56 - 73)  + (98 *   4))-(32 -   17))
-------Using A Stack To Evaluate Infix-------------------
Pushing ( into the stack
Pushing ( into the stack
Pushing ( into the stack
Pushing 56 into the stack
Pushing - into the stack
Pushing 73 into the stack
 Popping operand 2: 73 from the stack
 Popping operator: - from the stack
 Popping operand 1: 56 from the stack
 Popping ( from the stack
Pushing -17 into the stack
Pushing + into the stack
Pushing ( into the stack
Pushing 98 into the stack
Pushing * into the stack
Pushing 4 into the stack
 Popping operand 2: 4 from the stack
 Popping operator: * from the stack
 Popping operand 1: 98 from the stack
 Popping ( from the stack
Pushing 392 into the stack
 Popping operand 2: 392 from the stack
```

Pass check 3, but fail check 4, not a fully parenthesized expression as defined above.

Finally, something that passes all the checks!

Steps required to use a stack to evaluate this fully parenthesized expression

```
 Popping operator: + from the stack
 Popping operand 1: -17 from the stack
 Popping ( from the stack
Pushing 375 into the stack
Pushing - into the stack
Pushing ( into the stack
Pushing 32 into the stack
Pushing - into the stack
Pushing 17 into the stack
 Popping operand 2: 17 from the stack
 Popping operator: - from the stack
 Popping operand 1: 32 from the stack
 Popping ( from the stack
Pushing 15 into the stack
 Popping operand 2: 15 from the stack
 Popping operator: - from the stack
 Popping operand 1: 375 from the stack
 Popping ( from the stack
Pushing 360 into the stack
The final answer is 360
```



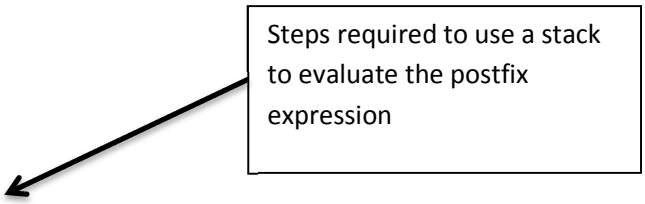
Steps required to use a stack to evaluate this fully parenthesized expression

```
-------Infix to Postfix--------------------
56 73 - 98 4 * + 32 17 - -
```

Use a stack to convert to postfix (you can display the results of the stack operation when you debug your code, but don't include in the final output)

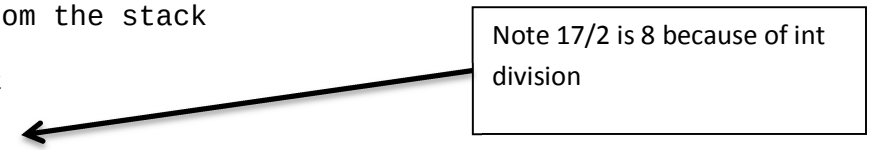
```
-------Using A Stack to Evaluate Postfix-------------------
Pushing 56 into the stack
```

```
Pushing 73 into the stack
 Popping operand 2: 73 from the stack
 Popping operand 1: 56 from the stack
Pushing -17 into the stack
Pushing 98 into the stack
Pushing 4 into the stack
 Popping operand 2: 4 from the stack
 Popping operand 1: 98 from the stack
Pushing 392 into the stack
 Popping operand 2: 392 from the stack
 Popping operand 1: -17 from the stack
Pushing 375 into the stack
Pushing 32 into the stack
Pushing 17 into the stack
 Popping operand 2: 17 from the stack
 Popping operand 1: 32 from the stack
Pushing 15 into the stack
 Popping operand 2: 15 from the stack
 Popping operand 1: 375 from the stack
Pushing 360 into the stack
The final answer is 360
```



Here is another Example run of the program

```
Enter a fully parenthesized expression that has
non-negative integer operands and using only + - * / and ()

Please enter the expression: ( ( 17 / 2) + (25 -18))

-------Using A Stack To Evaluate Infix-------------------
Pushing ( into the stack
Pushing ( into the stack
Pushing 17 into the stack
Pushing / into the stack
Pushing 2 into the stack
 Popping operand 2: 2 from the stack
 Popping operator: / from the stack
 Popping operand 1: 17 from the stack
 Popping ( from the stack
Pushing 8 into the stack
Pushing + into the stack
Pushing ( into the stack
Pushing 25 into the stack
Pushing - into the stack
Pushing 18 into the stack
 Popping operand 2: 18 from the stack
 Popping operator: - from the stack
 Popping operand 1: 25 from the stack
 Popping ( from the stack
Pushing 7 into the stack
```



```
 Popping operand 2: 7 from the stack
 Popping operator: + from the stack
 Popping operand 1: 8 from the stack
 Popping ( from the stack
Pushing 15 into the stack
The final answer is 15

-------Infix to Postfix--------------------
17 2 / 25 18 - +



-------Using A Stack to Evaluate Postfix-------------------
Pushing 17 into the stack
Pushing 2 into the stack
 Popping operand 2: 2 from the stack
 Popping operand 1: 17 from the stack
Pushing 8 into the stack
Pushing 25 into the stack
Pushing 18 into the stack
 Popping operand 2: 18 from the stack
 Popping operand 1: 25 from the stack
Pushing 7 into the stack
 Popping operand 2: 7 from the stack
 Popping operand 1: 8 from the stack
Pushing 15 into the stack
The final answer is 15
```

When you've finished your project, use the submit command to submit the files. You must be logged into your account and you must be in the same directory as the file you're trying to submit.

At the Linux prompt, type

```
submit cs201 Proj2 proj2.py
```

```
submit cs201 Proj2 Stack.py
```

After entering the submit command shown above, you should get a confirmation that submit worked correctly:

```
Submitting proj2.py...OK
```

```
Submitting Stack.py...OK
```

If not, check your spelling and that you have included each of the required parts and try again.

You can check your submission by entering:

```
submitls cs201 Proj2
```

You should see the name of the files that you just submitted, in this case proj2.py and Stack.py