CS417 Programming Assignment 8

Frequently-asked Questions

Q: How do I convert an infix expression to postfix?

A: Here is the pseudocode:

```
          make an empty stack, which holds operators
          make an empty postfix expression to begin with
          for each token in the infix expression,
              if token == '(',
                  push it
              else if token is an operand,
                  append token to postfix expression
              else if token == ')',
                  while stack is not empty and its top isn't '(',
                      pop from S, append to postfix expression
                  pop the ')'
              else (it's an operator),
                  while stack is not empty and the precedence of its
top
                         is higher or equal to token's precedence,
                      pop from S, append to postfix expression
                  push token
          when done, stack may hold some pending operators.
          if so, pop them, and append to postfix expression
```

Q: How do I evaluate a postfix expression?

A: Here is the pseudocode:

```
      make an empty stack
      for each token in postfix expression,
          if token is an operand,
              push token
          else (it's an operator)
              pop right operand
              pop left operand
              apply operator
              push result
```

Upon completion, the expression's value will be at the top of the stack.

Q: If I have a token, how do I know it is an operator, a number, or a variable name?

A: This problem is part of a current lab, but here is the gist: (this is not python, it's pseudocode)

```
           if token is one of "+-*/=",
               it's an operator
           else,
               try:
                   x = float(token)
                   it's a number, because no exception was raised
               except ValueError:
                   if token obeys the rules for python variable names,
                       it's a variable
                   else,
                       the expression has an error, and can't be
evaluated
```

Q: How are variables different from numbers?

A: You actually know this ☺, but it's worth spelling out. Variables are treated differently, depending on context.

- If a variable is the right operand, you should get its value from the symbol table.
- If a variable is the left operand of `"+-*/"`, you should also get its value from the symbol table.
- If a variable is the left operand of `"="`, you should set a value in the symbol table.

All of this must be handled in your `evaluate_postfix` function.

Q: What is the precedence of each operator?

A: Again, you know this, but there are some subtleties:

- `* /` : high precedence
- `+ -` : medium precedence
- `= ( )` : low precedence

Parentheses should have low precedence, because of this code in the second question above:

```
         while stack is not empty and the precedence of its top
               is higher than token's precedence,
              pop from S, append to postfix expression
```

Suppose you are converting this expression, and you are currently at the `'-'` token:

```
1 / (2 + 3 - 4)
          ^
```

At this point, the stack would have `/ ( +` . The top of the stack is `+` . If `(` had high precedence, then the while loop will pop the `+`, then the `(`, then the `/`. You would, at that point, have this partially-completed postfix expression:

```
1 2 3 + ( /
```

which is obviously wrong, because postfix expressions have no parentheses. The solution is to give parentheses low precedence, so the `while` loop will exit as soon as it finds the `(` on the top of the stack.

Q: How do I implement the `"="` operator? I know that the `+`, `-`, `*`, `/` operators produce a value. Does `=` produce a value too?

A: Absolutely **YES**. Here is why: Suppose you have this infix expression:

```
a = b = 20
```

The expression `b = 20` should have the value `20`. Thus, the above expression would change the value of `b`, and proceed to evaluate `a = 20`. As a result, both `a` and `b` get the value `20`.

To make this happen, in `evaluate_postfix`, you should push a value on the stack, when you process a `=` operator. That value is simply the value of the right operand.

Q: The `"="` operator is still giving me problems. Why does `a = b =  10` give an error?

A: The problem is that `=` is **right-associative**. In other words,

```
a = b = 10
```

is equivalent to

```
a = (b = 10)
```

Which, in postfix, is:

```
a b 10 = =
```

In other words, `b = 10` is evaluated **first**. Unfortunately, our infix-to-postfix conversion assumes that all operators are **left-associative**. For example, we expect that

```
a - b + c
```

should be interpreted as

```
(a - b) + c
```

Which, in postfix, is

```
a b - c +
```

In other words, `a - b` is evaluated first. Notice how the parentheses group differently in this case.

The problem, ultimately, lies in this `while` loop:

```
            while stack is not empty and the precedence of its top
                   is higher or equal to token's precedence,
                pop from S, append to postfix expression
```

If you have several equal-precedence operators, they will be popped in left-to-right order. But if there is an `=` operator on the stack, and you get another `=` operator, you should *not* pop from the operator stack. So, the test in the `while` loop should be changed. You need to figure that one out.