COP 3502C Midterm Assignment # 1

Total points: 13

Read all the pages before starting to write your code

Introduction: For this assignment you have to write a c program that will take an infix expression as input and display the postfix expression of the input. After converting to the postfix expression, the program should evaluate the expression from the postfix and display the result. Your solution should follow a set of requirements to get credit.

What should you submit?
Write all the code in a single file and upload the .c file to Webcourses.

Please include the following commented lines in the beginning of your code to declare your authorship of the code:

```
/* COP 3502C Midterm Assignment One
This program is written by: Your Full Name */
```

Compliance with Rules: UCF Golden rules apply towards this assignment and submission. Assignment rules mentioned in syllabus, are also applied in this submission. The TA and Instructor can call any students for explaining any part of the code in order to better assess your authorship and for further clarification if needed.
Remember: Sharing code to anyone is a violation of the policy. Also, getting a part of code from anywhere will be considered as cheating.

Deadline:

See the deadline in Webcourses. The assignment will accept late submission up to 24 hours after the due date time with 20% penalty. After that the assignment submission will be locked. An assignment submitted by email will not be graded and will not be replied according to the course policy.

What to do if you need clarification?

Write an email to the TA and put the course teacher in the cc for clarification on the requirements.

How to get help if you are stuck?

According to the course policy, all the helps should be taken during office hours (not by email).

Problem

We as humans write math expression in infix notation, e.g. 5 + 2 (the operators are written in-between the operands). In computer's language, however, it is preferred to have the operators on the right side of the operands, i.e. 5 2 +. For more complex expressions that include parenthesis and multiple operators, the computer has to convert the expression into postfix first and then evaluate the resulting postfix.

Write a program that takes an "infix" expression as input, uses stacks to convert it into postfix expression, and finally evaluates it. Read further to understand the requirements.

Input Description:

- Input will be an infix expression as a string with maximum length of 50.
- The expression will contain only positive integer numbers (no negative or floating point number)
- The numbers can be single or multiple digits
- The expression will contain only the following operators and parenthesis: +, - , / , * , ^ , % , (,)
- The expression might contain imbalance parenthesis (so you have to check it before starting the conversion. If it is not balanced, you should print a message and stop the conversion process for that incorrect infix expression)
- The expression can have whitespace between symbols, but not within a multiple digit number. For example, 56 + 7, or, 56+7, both are valid input., But 5 6+7 will not be used in input

Base Requirements for the assignment:

1. You must have to use Stack during the conversion and evaluation process the way we learned in the class.

2. The main function of the code should look like this with some modification. Note that you will need to declare some variables appropriately (for example, str, postfix, result, etc.). You can add maximum 10-15 more lines of codes in the main function as needed for your logic, free any allocated memory, etc.:

```
int main(void)
{
   while(strcmp(str = menu(), "exit")!=0)
   {
      if (isBalancedParenthesis(str))
      {
         postFix = convertToPostfix(str);
          printf("Output: %s", postfix);
         evaluate(postFix);

      }
```

```
        else
         printf("Parenthesis imbalanced");

    }
    return 0;
}
```

In addition to the other functions for multiple stacks or any other function as you wish, you have to write and utilize at least the following functions in your solution:

a) char* menu(): This function displays a menu. e for entering an infix, x for exiting the program.

If the user chooses e, it takes an infix as input, copy it into a dynamically allocated string and return it to the main function. If the user chooses x, it will copy ‘exit’ to a dynamically allocated string and return it.

b) int isBalancedParenthesis(char *): This function takes an infix expression and check the balance of the parenthesis. It returns 1, if it is balanced and 0 otherwise.

c) char* convertToPostfix(char *): This function takes the infix expression as string and converts it in to postfix. Note that this function can call other functions too to take help during this process.

d) void evaluateix(char *): This function takes the postfix expression as the parameter and evaluate the expression. At the end, it prints the result of the evaluation in a new line. Note that this function can call other functions too, to take help during this process.

e) int isOperator(char c): this function takes a char and returns 1 if the char is an operator. Otherwise, it returns 0;

f) int getOperatorPriority(char c): this function takes an operator and returns its priority

g) int convetToInt(char c): this function converts a char digit into int digit

h) int calculate(int a, int b, char op): this function takes to operand and one operator and returns the result of the operation based on op; Example: calculate(5, 6, ‘+’) will return 11

Some Example test expression

Example 1

Infix expression: (7 - 3) / (2 + 2))

```
Parenthesis imbalanced
```

Example 2

(5+6)*7-8*9

output: 5 6 + 7 * 8 9 * -

evaluation: 5

Example 3:

Infix expression: (7 - 3) / (2 + 2)

Postfix expression: 7   3   -   2   2   +   /

evaluation: 1

Example 4:

Infix expression: 3+(4*5-(6/7^8)*9)*10,
output: 3 4 5 * 6 7 8 ^ /9 * - 10 * +
evaluation: 203

Example 5:

Infix expression: 1000 + 2000
output: 1000 2000 +
evaluation: 3000

Rubric:

The code will be compiled and tested in Eustis server while grading. If your code does not compile in Eustis, we conclude that your code is not compiling and it will be graded accordingly. We will apply a set of test cases to check whether your code can produce the expected output or not. Failing each test case will reduce some grade based on the rubric given bellow. There is no grade just for writing the above functions, if your code fails all the test cases. However, if your code passes all the test cases, then ***-2 penalties will be applied for not implementing each function mentioned above.*** In summary: in order to get full credit for this assignment, you have to implement all the required functions and the code has to pass all the test cases.

1) If the code does not compile in Eustis server: 0.
2) If your code has missing any function explained above but passes all our test cases, -2 will be deducted for each missing function. However, there is no additional grade just for implementing all the functions. The code will be tested with varieties of inputs).

3) Checking the balance of the parenthesis: 2 points // Not displaying incorrect parenthesis - 2 points. You should check and stop your processing an infix if the parenthesis is imbalanced
4) Incorrect postfix expression per test case: -2 points //there will be test cases with single and multiple digits.
5) Correct postfix but incorrect evaluation per test case: -1 points
6) Handling single digit inputs: maximum 10 point
7) Handling two or more-digit inputs: 100 percent (if pass all test cases)

Read it: Some hints (but it is not the only way)

1. Implementing all the above functions will help you to solve the problems in a structured way.
2. Use the uploaded multi stack code for stack implementation and modify the code as you need.
3. You will need to use stacks in three places
    a. One for the parenthesis check [char stack]
    b. One during infix to postfix [char stack]
    c. One during evaluation [int stack]

    For a and b above, you can use same array and same push, pop method as both of them are char. But for evaluation you have int stack and you might consider to create another push pop method to handle it. Maybe push_int, pop_int, etc. Or find other strategy to utilize existing push pop method
4. You can use isdigit function to check whether a char is a digit or not
5. During evaluation you will need to convert char into integer. Example for single digit:

```
char c = '5';
int x = c - '0';
```

See uploaded code String2IntegerExample.c file for more example and testing. ***Also, the string exercise solution has a similar example at the end.***

Please see the lecture slides for the explanation, steps and more test examples.

Good Luck.