ECE 361 Fall 2023
Homework #4

THIS ASSIGNMENT SHOULD BE SUBMITTED TO CANVAS BY 11:59 PM ON SUN, 19-NOV-2023. LATE SUBMISSIONS WILL BE ACCEPTED UNTIL 8:00 AM ON WED, 22-NOV-2023 WITH A PENALTY OF 2 PTS/DAY. NO HOMEWORK #4 SUBMISSION WILL BE ACCEPTED AFTER THAT.

THE ASSIGNMENT IS WORTH 100 POINTS. IT IS EASIEST TO GRADE, AND I BELIEVE IT IS EASIEST FOR YOU TO SUBMIT, SOURCE CODE FILES AND TRANSCRIPTS AS TEXT FILES INSTEAD OF DOING A CUT/PASTE INTO A `.doc` OR `.docx` FILE. CLEARLY NAME THE FILES SO THAT WE KNOW WHICH QUESTION THE ANSWER REFERS TO. SUBMIT THE PACKAGE AS A SINGLE .ZIP, .7Z,.TGZ, OR .RAR FILE (EX: `ece361f23_rkravitz_hw4.zip`) TO YOUR CANVAS HOMEWORK #4 DROPBOX

CONSOLE TRANSCRIPTS SHOULD INCLUDE YOUR NAME, USING THE `whoami` COMMAND AND THE STATEMENTS WITHIN YOUR CODE THAT SHOWS THE WORKING DIRECTORY. TRANSCRIPTS SHOULD BE A .TXT FILE SAVED FROM THE BASH COMMAND LINE. PLEASE INCLUDE A NUMBER OF TEST CASES (INCLUDING EDGE CASES) IN EACH TRANSCRIPT TO DEMONSTRATE THE FULL FUNCTIONALITY OF YOUR PROGRAM. THE TRANSCRIPT IS YOUR CHANCE TO SHOW OFF YOUR HARD WORK. NOTE: SOME OF THE PROBLEMS ASK YOU TO PROVIDE A PARAGRAPH OR TWO ON WHY YOU THINK YOUR CODE IS WORKING. JUST ADD SOME TEXT TO THE END OF THE TRANSRIPT LIKE THIS:

```
***** EXPLANATION *****
<YOUR EXPLANATION>
*****END EXPLANATION*****
```

DOING THIS ELIMINATES THE NEED TO WRITE, EDIT, AND SUBMIT A DESIGN REPORT FOR THE PROJECT. YOU'RE WELCOME 😊

ACKNOWLEDGEMENT: THE RPM CALCULATOR PROBLEM IS BASED ON A PROGRAMMING PROJECTS PROVIDED BY K. N. KING IN *C PROGRAMMING: A MODERN APPROACH. STUDENTS FROM PREVIOUS CLASSES FELT THAT THIS WAS ONE OF THE MOST INTERESTING AND FUN SO ENJOY.*

FOR THIS ASSIGNMENT, YOU WILL BE TURNING IN THESE FILES (NOTE THAT THE FILE NAMES ARE JUST SUGGESTIONS, BUT PLEASE DO NAME YOUR FILES MEANINGFULLY):

- PROBLEM1: .C, .H (SOURCE CODE FOR YOUR STACK ADT)
- PROBLEM1: TRANSCRIPT.TXT (CONSOLE TRANSCRIPT AND EXPLANATION FOR PART 2)
- PROBLEM2: .C, .H (SOURCE CODE FOR YOUR RPN CALCULATOR)
- PROBLEM2: TRANSCRIPT.TXT (CONSOLE TRANSCRIPT FOR PART3)
- PROBLEM3: .C , .H (SOURCE CODE FOR YOUR IOM361 APPLICATION)
- PROBLEM3: TRANSCRIPT.TXT (CONSOLE TRANSCRIPT SHOWING YOUR PROBLEM 3 APPLICATION WORKING)

Introduction:

Problems 1 and 2 in the assignments are related. The final application is a simple Reverse Polish Notation calculator. The calculator will be stack-based (RPN is a LIFO operation) that makes use of a Stack ADT that you will implement. Your Stack ADT should be implemented using the API for a Singly Linked List ADT provided in the "starter" code. While this may seem to be inconvenient and maybe even restrictive, I think you will find that most of the "heavy lifting" for your Stack can be accomplished with Linked List API calls.

Problem 3 makes use of the iom361 temperature and humidity sensor. It asks you to get a series of temperature and humidity readings from iom361 and insert them into sorted arrays. The sorted arrays will be displayed in a table with one entry every three hours on a typical November day in Portland. A new function `iom361_set_Sensor1_rndm()` has been added to the iom361 API that generates random temperature and humidity values that are typical for Portland

The Reverse Polish Calculator

Early HP calculators used a system of writing mathematical expressions known as Reverse Polish Notation (RPN). In RPN, operators (+, -, *, /) are placed after their operands instead of between their operands. For example, the expression 1 + 2 would be written as 1 2 + and 1 + 2 * 3 would be written as 1 2 3 * +. RPN (https://en.wikipedia.org/wiki/Reverse_Polish_notation) expressions can be easily evaluated using a stack. The algorithm involves reading the operators and operands in an expression from left to right, performing the following operations:

- When an operand is encountered, push it onto the stack
- When and operator is encountered, pop its operands from the stack, perform the operation on those operands, and then push the result back onto the stack.

In this implementation of the calculator operands are single-digit integers (0, 1, 2…, 9). The operators are +, -, *, /, and =. The = operator causes the top stack item to be displayed. The stack is cleared after the = operator is executed, and the user is prompted to enter another expression. The process continues until the user enters a character that is not a valid operator or operand. For example:

```
Enter an RPN expression: 1 2 3 * + =
Value of expression: 7
Enter an RPN expression: 5 8 * 4 9 - / =
Value of expression: -8
Enter an RPN expression: q
```

Problem 1 (35 pts) Create a linked list-based Stack ADT

1. (5 pts) Study the code for the `SLLinkedList` ADT provided in the `starters` folder. This ADT provides a Singly Linked List API that can be put to good use in building your Linked List-based Stack. The SLLinkedList ADT is based on the singly linked list implementation in Karumanchi, but I have modified it to better hide (at least I think so) the internal architecture and to make it easier to build an application with more than one Linked List (not the case for this application).
    a. Build the Linked list ADT using the `gcc` compiler chain on the command line, along with the `test_LinkedList.c` application. Run the application and produce a transcript of a successful run.
    b. Use `gcc` to generate an object (`.o`) file for the Link List ADT. You will use it when you build your Stack ADT.

2. (30 pts) Create a Linked List Stack ADT that implements the functions shown in `starters/stackADT/stackADT_LL.h`. Your Stack ADT should be implemented using the API functions in `SLLLinkedList`. I have provided a `stackADT_LL.c` file in the `starters` folder to provide a template for your ADT.
    a. Build your Stack ADT using the `gcc` compiler chain on the command line along with `test_stack_LL.c` or (better yet) a test program of your creation.
    b. Run the application and produce a transcript of a successful run. Add an explanation to the transcript you submit (I'm expecting a paragraph or two) to why you think your Stack ADT is working correctly.
    c. Use `gcc` to generate an object (`.o`) file for your Stack ADT. You will use it when you build your RPN application.

Problem 2 (45pts) The RPN calculator

Write an application that evaluates RPN expressions using the Stack ADT implementation that you created in Problem 1. The functionality is described earlier in this document. A stack is an optimal data structure for an RPN calculator because operands are pushed onto the stack as they are entered and the top two entries on the stack are popped off the stack and the operation performed with the result of the operation pushed back onto the stack. For example, consider the sequence `5 9 + =` This sequence will perform the following operations:

- Push 5 on the stack (stack: 5)
- Push 9 on the stack (stack: 9 5)
- Pop 9, Pop 5, ADD, Push 14 (stack: 14)
- Pop 14, Display the result, destroy the stack, create the stack anew (stack:)

a. (35 pts) Write and compile your RPN calculator using the `gcc` command line. Include the source code for your application and the .o files you created in Problem 1 so that the Linked List and the stack ADT are linked to the application by gcc. Include float_rndm.o in the list. The float_rndm ADT provides the ability for iom362_r2 to generate random temperature and humidity values
b. (10 pts) Demonstrate your RPN calculator and a transcript showing the output from several test cases. Your test cases should include these test cases and several your own:
    - 1 2 3 * + = (Result: 7)
    - 5 8 * 4 9 - / = (Result: -8)

Problem 3 (20 pts) Using the iom361 temperature and humidity sensor

This problem makes use of the simulated temperature/humidity sensor in iom361_r2. It is a simple application that requests temperature and humidity readings over a typical 24-hour day in November in Portland. The readings are taken every 3 hours so 8 readings should be saved in sorted arrays for temperature and humidity from lowest to highest. Your application should display a temperature table and a humidity table, along with the average temperature and average humidity for the day. *Hint: We looked at a function to insert an item into a sorted array when I introduced data structures.*

a. (15 pts) Write and compile your application using the `gcc` compiler chain on the command line along with `the .o files for iom361_r2.` Note: MAC and Linux users will have to compile the `iom361_r2` and the random float generator source code like you did for HW #3. The .o file(s) I included are for Windows.
b. (5 pts) Run the application and produce a transcript of a successful run. Add an explanation to the transcript you submit (I'm expecting a paragraph or two) to why you think your application is working correctly