CS2123 Program #2 Postfix Evaluation (50 points)

This is a continuation of Program #0 (Read Customer Data) and Program #1 (Infix to Postfix). In this assignment, we evaluate the queries. You have been provided a driver program, cs2123p2Driver.c, which greatly reduces your amount of work.

You don’t have much code to write in Program #2. This program involves understanding the concepts more than it involves coding labor. Excluding comments, your code for just Program #2 (not the parts from program #0 and program #1) should be under 100 lines of code.

This program is important in helping you understand

* Using a stack to evaluate postfix
* Integrating your code with someone else's code (mine)
* Techniques for error handling
* Technique for handling command line arguments

Command arguments:

p2 -c customerFile -q queryFile

Input:

See cs2123Driver.c

Process: (green highlighting indicates work you previously did and yellow highlighting indicates your new work):

1. Read the customer identification information and traits, storing in the structures.
2. Print the customer data (as was done in Program #0).
3. Read the queries as was done in Program #1. For each query:
   1. Convert the query to postfix (as in Program #1).
   2. Evaluate the postfix query producing a list of customers satisfying the query. The list is just a boolean for each customer.
   3. Print the query, the postfix form of the query, and print the customer id and name of customers satisfying the query.

**Approaches for Evaluating Operators**

Approach #1: Iterate over the list of customers and then evaluate the query for each customer. The operator evaluation result is simply true or false (since only one customer is considered).

* For N customers, this approach evaluates the same query N times.
* The operator result (i.e., a boolean) must be stacked.
* USE NESTED LOOPS WITH OUTER GOING THROUGH CUSTOMERS AND INNER EVALUATING QUERY

Approach #2: When evaluating an operator, iterate over the customers to determine if each customer satisfies the query. The operator evaluation result is an array corresponding to the Customer array with a boolean indicating whether the customer satisfies the result.

* A query is evaluated once.
* Each operator will iterate over the list of N customers.
* The operator result (i.e., the list of booleans) must be stacked.

We will use approach #1.

**Single Customer Operator Functions**

int atLeastOne(Customer \*pCustomer, Trait \*pTrait) // = operator

int notAny(Customer \*pCustomer, Trait \*pTrait)

int only(Customer \*pCustomer, Trait \*pTrait)

You will also have to handle AND and OR. I have provided the **notAny** function.

**Issue: two different types of stack Element.** Our stack functions can have only one typedef for Element, but we have two different things to stack:

1. Converting from infix to postfix: Element structure of Program #1 containing szToken, iCategory, and iPrecedence.
2. Evaluating Postfix needs two types of stack elements:
   * When we push SMOKING and N before evaluating the "=". the Element structure from Program #1 is needed. It contains szToken, iCategory, and iPreference.
   * The result of evaluating the operators is always a boolean value.

In class we discussed different approaches, since the evaluation needs two types of elements, we will use **union**.

Examples:

* 1. SMOKING = N AND GENDER = F 🡺   
     SMOKING N = GENDER F = AND
  2. SMOKING = N AND ( EXERCISE = HIKE OR EXERCISE = BIKE ) 🡺  
     SMOKING N = EXERCISE HIKE = EXERCISE BIKE = OR AND
  3. GENDER = F AND EXERCISE NOTANY YOGA 🡺

GENDER F = EXERCISE YOGA NOTANY AND

* 1. SMOKING = N AND EXERCISE = HIKE OR EXERCISE = BIKE 🡺  
     SMOKING N = EXERCISE HIKE = AND EXERCISE BIKE = OR
  2. ( BOOK = SCIFI ) 🡺 BOOK SCIFI =
  3. ( ( ( BOOK ONLY SCIFI ) ) ) 🡺 BOOK SCIFI ONLY

Precedence rules:

* Evaluate contents in () before others - Highest precedence
* =, NOTANY, ONLY
* AND, OR - Lowest precedence

Some Files for your use:

p2customer.txt - customer file

p2query.txt - query file

cs2123p2.h - include file for Program #2

cs2123p2Driver.c - driver programmer that I provided. It has several useful routines to help reduce your effort. Please review this code

Your coding:

* Create your code in **cs2123p2.c** (not cs2123p2Driver.c). Based on what the driver calls, you need to create (at least) these functions:

void printCustomerData(Customer customerM[], int iNumCustomer) // you did this in program #0

int convertToPostFix(char \*pszInfix, Out out) // you did this in program #1

It returns 0 if it converted successfully. Otherwise, it returns a value which indicates an error in

the infix data (e.g., missing left paren, missing right paren)

It populates the out array using the addOut function (provided in the driver).

void evaluatePostfix(Out out, Customer customerM[], int iNumCustomer, QueryResult resultM[])

It returns the query result based on evaluating the query. resultM is an array of booleans corresponding to customerM.

int atLeastOne(Customer \*pCustomer, Trait \*pTrait)

int only(Customer \*pCustomer, Trait \*pTrait)

* To compile the driver use (producing cs2123p2Driver.o):

gcc -g -c cs2123p2Driver.c

* To compile just your code, but not link it, use (producing cs2123p2.o):

gcc -g -c cs2123p2.c

* To link your object, the driver, and C libraries, producing the executable named p2:

gcc -g -o p2 cs2123p2Driver.o cs2123p2.o

* To execute your program p2:

./p2 -c p2customer.txt -q p2query.txt

Requirements:

1. Your code must be written according to my programming standards.
2. You should not have to modify cs2123p2Driver.c. If you want to modify it, please see me before you do that.
3. Turn in your C code, include files, and output generated using the specified input file.
4. Make certain you free up allocated memory (e.g., stack).
5. Modularity matters.

Hint:

There are many useful functions in cs2123p2Driver.c. These can greatly simplify your code.

Meaning of the =, NOTANY, and ONLY Operators

* The "=" operator is interpreted as "at least one". "EXERCISE = HIKE" means at least one of the EXERCISE traits must be HIKE. There can also be other EXERCISEs that are not HIKE for the customer.
* The NOTANY operator means that the specified trait must not exist for the customer. If a customer had the traits EXERCISE HIKE and EXERCISE YOGA, do not include the customer in the operator result when the query specifies EXERCISE NOTANY YOGA.
* The ONLY operator means that the specified trait value is the only value allowed for the specified trait type. That specified trait type and trait value must exist. If the customer has other trait values for the specified trait type, do not include the customer in the operator result. If the customer's traits include additional trait types, ignore those trait types since they have no affect on the result of ONLY.