CST8234 – C Programming F17 (Lab 4)

Programming Exercise

In this lab, we will gain experience with

- Using memory allocation to maintain dynamic lists

This assignment is to be done in pairs.

Statement of the problem

We'll pretend we are building a ticketing system like you might find in the student services office. This simulated ticketing system supports automatically generates a sequentially-increasing ticket number for each new ticket (e.g., 1, 2, 3, 4, 5…) and supports the following master list categories of tickets

- Name: Academic Registrar, code: AR
- Name: Financial Services, code: FS
- Name: IT Support, code: IT
- Name: Parking Police, code: PP
- Name: Coop Placement, code: CP

However, only some of these categories will be used during each simulation. The user will enter the number of categories (nNumCategories) to use (ranging from 1 to 5), and you will randomly select that many categories from the master list above, and use only those categories. E.g., in one run, the user may want to use 3 categories, and randomly get AR, IT, PP. On the next run they may again select 3 categories and randomly get AR, FS, CP. Or if they select 2 categories, they may get IT, CP, etc.

The user will then enter a number of tickets (nNumTickets) to simulate (ranging from 0-1000).

The selected number of tickets will then be individually created and randomly assigned to one of the nNumCategories categories, where it will be appended to that category's dynamically-allocated array.

You will then print out each category and its tickets, in which the presentation of each ticket will be a 5-character ID that combines the category code it was assigned to, and the sequential number (e.g., "AR001", "IT002", "IT003", FO004", "FS005", "PP006", …).

You will then randomly select one ticket (from any list) and tell the user which ticket you have deleted, and re-print the category lists.

You will repeat the process of selecting a ticket to delete, and then reprinting the lists, until there are no tickets left, at which point your program will exit.

A sample run might look as follows

```
$ ./lab4.exe
Enter the number of categories to use: 3
Enter the number of tickets to generate: 8

Initial ticket distribution
Financial Services:
FS001, FS008
Parking Police:
PP002, PP003, PP006
Coop Placement:
CP004, CP005, CP007

Removed ticket PP003
Financial Services:
FS001, FS008
Parking Police:
PP002, PP006
Coop Placement:
CP004, CP005, CP007

Removed ticket FS008
Financial Services:
FS001
Parking Police:
PP002, PP006
Coop Placement:
CP004, CP005, CP007

Removed ticket FS001
Financial Services:
- no tickets -
Parking Police:
PP002, PP006
Coop Placement:
CP004, CP005, CP007

Removed ticket PP006
Financial Services:
- no tickets -
Parking Police:
PP002
Coop Placement:
CP004, CP005, CP007


[some iterations deleted for brevity]


Removed ticket CP004
Financial Services:
- no tickets -
Parking Police:
- no tickets -
Coop Placement:
- no tickets -

$
```

Important Rules

Note, the formatting (and wording) must match the example above, perfectly, including spacing and use of comma delimiter.

The program must generate different results each time it is run (i.e., use srand).

NO statically-sized arrays may be used for ANY purpose. You may ONLY use malloc, calloc and/or realloc to get space for your arrays. Use of any statically-size arrays (event for your master list of categories) will result in a ZERO on the coding correctness portion of the marking.

NO global (a.k.a., file scope) variables may be used. You must pass parameters between functions. Use of any statically-sized arrays will result in a ZERO on the coding correctness portion of the marking.

You may not use linked lists to implement the project. (There will be another lab to demonstrate that functionality. This lab is about manipulating dynamically-allocated arrays.)

You must put your code in to logically-grouped files. Suggestions include having a file to collect the desired number of categories and tickets, a file to initialize the list of categories, a file to handle appending an removing items from a list, a file to encapsulate all the formatting routines. Each file doesn't have to be big… but it does have to be logically organized.

You must have a header file for each .c file.

You can expect the instructors to test your program with large numbers, just to see if it'll break.

You must free all memory before exiting.

Data Structures

You might already have surmised, there is a probably going to be a struct called "Category" that has a full name (e.g., "Coop Placement"), a code (e.g., "CP"), a dynamically-allocated list of tickets, and a number of items in that category's dynamically-allocated list.

Each ticket must be implemented as a struct consisting of an integer ID (e.g., 3), and a long timestamp (which can be left as zero for this lab). I.e., your category dynamically-allocated arrays must be arrays of structs… not just simple integers.

Hint: use typedefs for your structs.

Hint: have a ticket formatter that combines the category code (e.g., "CP") with the ticket id (e.g., 3) to create "CP003".

Hint: you might find it easier to create and initialize the master list of all five categories, and then randomly delete some. E.g., if the user only wants 2 categories in the simulation, then randomly delete 3 of the entries in the master list, leaving you with only the two desired categories.

Requirements

1. Create a folder based on both of your Algonquin User ID's (e.g., "smit9112_sing0003"). Do all of your work in this folder, and when complete, submit the zipped folder as per the "Lab Instructions" posted on Brightspace.
2. Write a program that satisfies all the requirements articulated above.
3. You still must provide a makefile with the standard compiler flags (`-g -Wall -w -pendantic -std=c99`)

Marking

This assignment is out of 30

- 10 for coding correctness (i.e., correct results)
- 10 for appropriate and efficient logic (i.e., no spaghetti code!)
- 10 for clear comments and coding convention (not necessarily K&R, but it should be clean and consistent)
- A -3 penalty applies for work submitted individually

You can also lose marks for incorrect submission (e.g., including unnecessary files in the zipped folder), compiler warnings, typographic errors (including in comments).

Submission

When you are done, submit your program to Brightspace. Make sure that you have the appropriate header in your source file(s), and have zipped up the appropriately name directory. Only include the source code (.c, .h) and your makefile (mandatory!).

You ***must*** include a makefile, as part of your submission! When grading your reports I will unpack your zip file and type '`make`'… and if I don't end up with a 'lab4.exe' to run, ***I will not grade your assignment***.