Programming Assignment 4, TCSS 333A, Winter 2021

OBJECTIVE

The objective of this assignment is to give you more practice with creating and using ADTs, multi-file program format, and text file reading.

ASSIGNMENT SUBMISSION

To get credit for this assignment, you must
- ✓ write a multi-file program in C that compiles and you wrote on your own (no copying or help outside of CSS mentors, TLC, or instructor allowed)
- ✓ submit your project as a tar file through Canvas exactly as instructed (naming, compatibility, etc.)
- ✓ submit your assignment by the due date

PROBLEM STATEMENT

For this assignment, you need to implement a program that helps real estate agents match clients with the houses they may be interested in buying. All the necessary data is provided in *txt* and *csv* files and it is your job to put this information together and produce an output file.

Processing and data structures
Option 1
To store client information described below, you will need to set up one ADT for a client and their preferences, as well as use the generic array list ADT that was set up during the lectures - you will need to add operations to the list that are needed by your program but they need to be generic in nature as well. Both client and their preferences information should be set up in an ADT (i.e. in interface and implementation file) but you do not need to follow proper information hiding principles if it makes your coding easier. However, in case of the list, you need to follow proper design practice of hiding implementation details from the user, i.e. you should not attempt to use private list components in the driver.
The order of processing is as follows:
- based on the first input file, create the client ADT instances and store them into the list
- based on the second input file, update client ADT instances stored in the list with their preferences
- based on the third input file, check client ADT instances in the list for a match

OR

Option 2
To store client information described below, you will need to set up two ADTs: one for a client and one for their preferences, as well as use an array of these structures. Both client and preferences information should be set up in ADTs (i.e. in interface and implementation files for each type) and they need to follow proper design practice of hiding implementation details from the user, i.e. you should not attempt to use private preferences or client components in any other piece of code.
The order of processing is as follows:
- based on the first input file, client ADT instances and store them into the array
- based on the second input file, update client ADT instances stored in the array with their preferences
- based on the third input file, check client ADT instances in the array for a match

Input Files and ADTs

The first input file is called *clients.txt* and contains the following information about each client:
- client id
- name
- e-mail

- phone number

Sample file is provided so that you could see the file format. In order to store this information, you need to create client data type that is capable of storing this information, with each bullet above constituting one field within the data type.

The second input file is called *preferences.txt* and contains the following information about each client's preferences:

- client id
- the minimum number of bedrooms
- the minimum number of bathrooms
- the maximum price
- neighborhood id

Sample file is provided so that you could see the file format. In order to store this information, you need to create preferences data type that is capable of storing this information, with each bullet above (other than id) constituting one field within the data type. An instance of this data type is a field in client's data type.

The third input file is called *houses.csv* and contains the following information about each house available for sale:

- MLS (multiple listing service) number
- street address
- city
- zip
- number of bedrooms
- number of bathrooms
- price
- neighborhood id

You do NOT need to set up a separate structure data type for this info. If you want to, you can just read this information as a string in the driver file.

Output Files

Your program is to produce an output file called *houseView.csv* and contain the houses for which client matches were found. Each line of that file should be the exact match to *houses.csv* file, where each row information is followed by rows containing potential client's information. Client information should include client's id and name only.
Creating a match on client's preference should be quite obvious, as if a client states they want at least 4 bedrooms, 3 bathrooms, with a maximum price of 700k, then any house with those parameters qualifies to be a match. The neighborhood should be matched within the range of 10 on either end of the original value, e.g. if the preference is 20, then the range 10 – 30 is the valid neighborhood id range. It is possible that no match for a house was made, in which case the house is not written out to *houseView.csv*. Sample file is provided so that you could see the file format and the contents to be included.

Program Specs

- At the top of EACH C file provide a comment with your name, whether you tested your code on the cssgate server or Ubuntu 16.04 LTS Desktop 32-bit, and the option you chose
- The program needs to follow object-oriented principles and use structures to define data types and their operations
    - Option 1:
        - one ADT for client objects that does NOT need to follow ADT principles of information hiding that includes another structure type for client's preferences
        - generic array ADT implementation that is to follow ADT principles of information hiding and its implementation details need to be hidden from the end user (you should not attempt to use private list components in the driver)
    - Option 2:
        - one ADT for client objects that needs to follow ADT principles of information hiding
        - one ADT for client preferences that needs to follow ADT principles of information hiding

        - an array of nested structures described above
- Assume all input file contents are valid text files and adhere to the format presented in sample input files
- Name the file containing the *main* function *driver.c*
- **You need to create a makefile for your code – the code will not be graded otherwise as there are different linking possibilities (it should be called makefile, no extensions or other names)**
- Your executable file should be called *pr4.out*
- All program files must reside in one folder and the contents of the entire folder must be compressed using tar utility (check tutorial 1 directions to make sure you are compressing an entire folder and not individual files)
- Your tar file should be called *pr4.tar*
- Your program has to follow basic stylistic features, such as proper indentation (use whitespaces, not tabs), meaningful variable names, etc.
- You need to comment your code
- You are not allowed to use global variables
- **Your program must compile in gcc gnu 90 – programs that do not compile will receive a grade of 0**

GRADING

Remember, the programming assignments are graded as pass / no pass only and 85 points (out of 100 are needed to pass). If you do not pass this assignment, you can turn it in again with assignment 5.
General program functionality (with file reading and writing) is worth 25 points
Proper decomposition into ADTs (depends on the chosen option) is worth 45 points
Clean memory report is worth 15 points
Proper coding style, meaningful identifiers, etc. is worth 15 points

PROGRAM SUBMISSION

On or before the due date for assignment 4, use the link posted in *Canvas* next to *Programming Assignment 4* to submit your tar file. Make sure you know how to do that before the due date since late assignments will not be graded until the next grading period.