CST8234 – C Programming W18 (Assgn1)

Programming Exercise

The purpose of this assignment is to demonstrate that you can do dynamic allocation of memory, using singly-linked lists

This function builds on some of the work that you did in Lab 4. You may be able to reuse much of the code from your previous assignment!

Statement of Problem

Work with a partner to write a program that represents a Tindr-like app for rental properties.

I.e., you'll have a list of rental opportunities that you can dismiss (i.e., "swipe left)", or add to a list of properties that you'd like to further consider (i.e., "swipe right")

You will maintain **TWO singly-linked lists**. One is for unviewed properties (called "default"), and one for your rental properties that you'd like to consider further (called "favourites"). At any time, one of these lists will be the current list.

You will build an interface that will do the following (see "User Interface" section below, for more details)

- Let the user get a list of commands
- View the "default" list (possibly adding a new item to the "default" list)
- View "favourites" list
- Next (displays the next rental property in the current list, without making a decision)
- Discard (i.e., swipe left)
- Add to favourites (i.e., swipe right)
- Set the sorting mode, to one of four criteria
- Exit the program

Generating New Rental Properties

When the program starts, you will randomly create 6 random properties, and append them to the "default" list.

Every time the user enters the "View New Properties" command (i.e.,***DEFAULT*** from the list of commands, below), there is a 50% chance (i.e., 1 in 2) that you will generate a new property, and append it to the "default" list. This simulates new listings that periodically come into being.

Random Generation of New Rental Properties

For each new property that you randomly generate you need to generate some random properties.

You will have a fixed list of 10 streets. Each street will have a **name** of your choosing (e.g., "Cat St.", "Tiger Boul.", "Ferret Ave."), and a randomly-generated **base distance** from campus, from 0.5km – 4km, in 100m increments (e.g., 100*random(5,40)). Clearly, you'll need an array of structs to hold all of your street information.

Like in Lab 4, each rental property will be a struct. In this assignment, it will have members for the **street**, the **street number**, the **price per bedroom**, and the **number of bedrooms**.

For each new property, you will

- randomly select a street to use, and store a pointer to that street struct
- randomly generate a number between 1-200 for the street number
- randomly number of bedrooms, from 1-4
- randomly generate a rent per bedroom from $200-$600 in $50 increments (i.e., 50 * random(4,12))

The random values need to differ every time you execute the program.

Property Functions

In addition to the standard setters and getters, you'll need to define a function to calculate the distance from the College.

The distance of a property is defined by the base distance of the street, plus 20 m per street number. E.g., if "Fox St." is 600 m distant from the College, and the street address is 38, then the total distance of the rental property is 600+20*38 = 676 m.

Property Display Functions

The address of a property is listed as the street number concatenated with the street name. E.g., "38 Fox St."

Distances are stored in metres, but are display in km's, expressed as a float with **two decimals**. E.g., 676 m will be shown as "0.68 km".

You will need a function that will accept a pointer to a property structure, and print a nicely-formatted summary of a property (e.g., the formatted address, the rent/room, the total distance from campus, and the number of bedrooms).

You will also need a function that will accept a pointer to a property structure and print out the information in a form suitable for display in a table (like Lab 4).

User Interface

The basic sequence is that the user types a brief command as listed below, and then they get some visual feedback (as described in the Details for each command).

Key	Command	Details
h	HELP	Print out a brief list all the available commands
q	QUIT	Exit the program
a	ALL	Sort the current list of rental properties ("default" or "favourites"), and then show them in a tabular form (as done in lab 4). After showing the user the entire list, ask what they think about the first rental property in the list. If there are no rental properties in the current list, display "There are no more rental properties".
d	DEFAULT	Generate a new rental property (50% of the time) and add it to the "default" list. Set the current list to "default", and do an ***ALL***.
f	FAVS	Set the current list to "favourites" list, and do an ***ALL***.
s<n>	SORT	Organize the properties according to the desired sort method **<n>** **r** – Rent per Room (ascending) **d** – Distance (ascending) **n** – Number of rooms (descending) **a** – Address (ascending) And then do an ***ALL***. Sorting rules for the different fields (e.g., rent) are the same as in Lab 4.
n	NEXT	Either a) display the next rental property on the current list, or b) print "There are no more rental properties" if you were already at the last one on the list.
l	LEFT	Remove the just-viewed property from the current list, delete it, and either a) display the next rental property on the current list, or b) print "There are no more properties" if you were already at the last one on the list.
r	RIGHT	If you are currently viewing the favourites list, print out "This property is already on the favourites list". Otherwise, remove the just-viewed property from the default list, and append it to the favourites list, and either a) display the next rental property on the default list, or b) print "There are no more rental properties" if you were already at the last one on the list.
Anything else	UNSUPPORTED	Print out "That is not a supported command", and then do a ***HELP***.

Because this assignment is about use of linked lists, and not input validation, you may assume that any command that instructors enter will only be, at most, a few characters, and you are permitted to use a input buffer of 8 chars, without worrying that the instructors will type in long inputs and force buffer overruns.

Sorting

You will remember the current sorting preference (e.g., by price, by distance, by number of bedrooms), and use this mode.

As in Lab 4, when sorting by address, when you'll sort by street name, and if the street names are equal you'll then use street number. I.e., you'll need to ensure that "22 Fox St." comes after "3 Fox St.", but before "1 Goose Rd.". I.e., don't do a dictionary sort based on the formatted address. See Lab 4 for hints.

Linked List Management

You must write functions that can do the following

- Count the number of items in a list.
  - E.g., "`int getCount( Node *pHead );`"
- Find the i-th element in a list.
  - E.g., "`Node *getNodeAtIndex( Node *pHead, int i );`"
- Append an item to the end of a list.
  - E.g., "`void appendNode( Node **ppHead, Node *pNewNode );`"
- Insert an item into a list, at position i.
  - E.g., "`void insertNode( Node **ppHead, Node *pNewNode, int i );`"
- Remove the i-th item from a list.
  - E.g., "`Node *removeNodeAtIndex( Node **ppHead, int i );`"

The functions above must be able to operate on **EITHER** your "default" list or "favourites" list. I.e., you may not have a two "count the number of items" function (one for "default" and one for "favourites").

You'll have to decide if you want a dedicated "Node" struct that has a pointer to a rental property struct, and a pointer to the next "Node", and whose only purpose is to create a singly-linked list, or whether you want to add the self-referential pointers to the rental property structure. I **recommend** having a dedicated Node struct, because it neatly separates the content of the property structs from the way in which they are being organized (e.g., linked list vs arrays). This also simplifies sorting (see 'Hints' below).

Multiple Files

You must split your code between multiple source files. I strongly recommend that you write it this way from the beginning. I.e., think like you're using Java, and decide which functionality belongs in which class (i.e., file). Putting everything in main.c, and then splitting it up at the end "just to satisfy the instructor" is a really dumb way to approach your program's structure.

You will be marked on the sensible way you divvy up the functionality into multiple files. Conversely, you will lose a lot of marks if you just arbitrarily carve up your functions in ways that don't make sense.

You will need to define .h files for most source file that contains the 'extern' declarations to the functions that will need to be accessed from other files.

My implementation had the following files.

- main.c — initialization and input parsing
- rental.c — all functions related to properties, e.g. initialization, setters/getters, display functions
- sort.c — all functions relating to comparators, sorting and swapping
- node.c — all functions relating to linked lists, e.g., creating nodes, appending, removal, etc.

You're not obligated to use the same division of functionality into files, but you'll want to use something sensible, as opposed to randomly chucking a bunch of functions into different files and hoping it will satisfy the instructions (hint: it won't, and you'll lose marks).

Special Restrictions

You MUST work in pairs on this assignment. **Solo submissions will not be accepted.** You must register your pairs with Basim, so he can configure Blackboard for your pair's submission.

You must implement your lists of rental properties with linked lists**. Implementations that use arrays of rental property structs instead of linked lists WILL BE GIVEN A GRADE OF ZERO**.

In an effort to force you to pass information via parameters, rather than relying on global variables, you are not permitted to define any global variables.

Structs must be passed into functions (e.g., your comparison functions, or your swap routine) by ***reference*** (i.e., as pointers) rather than by ***value*** (i.e., as copies of the structs). This means that you'll end up using "->" notation, not the "." notation. Failure to use pointers will result in deducted marks. Yes, you will need to pass in double pointers to many of your linked-list manipulation routines.

Miscellaneous Hints

I incrementally built on Lab 4. I.e., I changed the Rental Property struct to conform to this assignment's data requirements, and updated the tabular list functions to show the new data, and added new sort comparators. I then changed the implementation to use linked lists. Once Lab 4 was working with linked lists, and the new data structure, I implemented the additional Assignment 1 functionality… e.g., maintaining a "default" and "favourites" list, and accepting input.

I used function pointers to simplify picking which sorting comparator to use (i.e., rather than having to use multiple if-then-else / switch statements all over the place). E.g., I defined a function pointer called "pComparator" as follows:

```
int (*pComparator)( RentalProperty *a, RentalProperty *b);
```

You may want to divide up the work by logically grouping. E.g., one person can be working on adapting Lab 4 to use linked lists, while the other is working on the user interface.

If you implement a distinct Node data structure that is separate from the Rental Property data structure, then you can actually implement a swap algorithm by changing the "payloads" rather than having to re-organize the linked list (much, much simpler!)… i.e., I have a swap routine that takes a pointer to the two nodes that are being "swapped" and just exchanges their payloads.

```
typedef struct _Node {
    RentalProperty  *pRental;           // 'payload' of this list node
    struct _Node    *pNext;             // pointer to next node
} Node;

void swap( Node *pNode_A, Node *pNode_B )
{
    RentalProperty temp = pNode_A->pRental;
    pNode_A->pRental = pNode_B->pRental;
    pNode_B->pRental = temp;
}
```

Submission

When you are complete submit your programs to Blackboard.

But… before you do, please check that you've satisfied the following submission requirements

1. Did you confirm that your zipped submission is a ".zip" file (not a '.rar' or '.tar.gz' or '.7z' file)?
2. Did you zip up a folder that includes both partner's *user names*, and the lab/assignment indicator? I.e., if you open up your own zipped submission, you should see a *folder* called (for example) "smit9112_jone0001_A1". If you just see *file(s)*, you've done it wrong, and you'll need to go to the *parent* folder, and try zipping your lab folder.
3. Did you remove all the unnecessary files from the folder contained in your zipped submission? I.e., you open up your own zipped submission, and click on the folder called (for example) "smit9112_A1", you should see *just your makefile and your source files, and include files.*
4. Is your makefile actually called 'makefile' or 'Makefile' (without any filename extensions?)

If you don't satisfy submission requirements #1 or #2 you will get ZERO on this assignment. If you realize afterwards that you've made a mistake, don't panic! you are allowed to correct your mistake and re-submit. But you ONLY get TWO submissions per lab/assignment, so try to make sure you double-checked everything *before* doing your first submission… and only use the second submission in case of emergency.

Sample Session

```
$ ./ass1

Undecided Rental Properties
Address                           # Rooms  Rent/Room   Distance
-------------------------         -------  ---------  ---------
102 Mouse Ave.                          4        350    4.94 km
 22 Fox St.                             2        400    3.04 km
 57 Coyote Crt.                         4        400    4.54 km
  5 Cat St.                             4        400    3.00 km
 91 Cat St.                             1        500    4.72 km
168 Squirrel Cres.                      2        600    4.56 km

What do you think about this rental property?
        Addr: 102 Mouse Ave., # Rooms: 4, Rent/Room: $350, Distance: 4.94 km

Command ('h' for help): h

Valid commands are:
        h  - display this help
        a  - display all the rental properties on the current list
        f  - switch to the favourites list
        d  - switch to the default list
        l  - 'swipe left' on the current rental property
        r  - 'swipe right' on the current rental property
        n  - skip to the next rental property
        sa - set the sorting to 'by address'
        sn - set the sorting to 'by number of rooms'
        sr - set the sorting to 'by rent'
        sd - set the sorting to 'by distance'
        q  - quit the program

Command ('h' for help): sd

Undecided Rental Properties
Address                           # Rooms  Rent/Room   Distance
-------------------------         -------  ---------  ---------
  5 Cat St.                             4        400    3.00 km
 22 Fox St.                             2        400    3.04 km
 57 Coyote Crt.                         4        400    4.54 km
168 Squirrel Cres.                      2        600    4.56 km
 91 Cat St.                             1        500    4.72 km
102 Mouse Ave.                          4        350    4.94 km

What do you think about this rental property?
        Addr: 5 Cat St., # Rooms: 4, Rent/Room: $400, Distance: 3.00 km

Command ('h' for help): r
```

```
Rental property moved to your favourites list

What do you think about this rental property?
        Addr: 22 Fox St., # Rooms: 2, Rent/Room: $400, Distance: 3.04 km

Command ('h' for help): l

Rental property deleted

What do you think about this rental property?
        Addr: 57 Coyote Crt., # Rooms: 4, Rent/Room: $400, Distance: 4.54 km

Command ('h' for help): l

Rental property deleted

What do you think about this rental property?
        Addr: 168 Squirrel Cres., # Rooms: 2, Rent/Room: $600, Distance: 4.56 km

Command ('h' for help): r

Rental property moved to your favourites list

What do you think about this rental property?
        Addr: 91 Cat St., # Rooms: 1, Rent/Room: $500, Distance: 4.72 km

Command ('h' for help): r

Rental property moved to your favourites list

What do you think about this rental property?
        Addr: 102 Mouse Ave., # Rooms: 4, Rent/Room: $350, Distance: 4.94 km

Command ('h' for help): l

Rental property deleted

No more rental properties

Command ('h' for help): f

Favourite Rental Properties
Address                          # Rooms  Rent/Room   Distance
-------------------------        -------  ---------  ---------
  5 Cat St.                            4        400    3.00 km
168 Squirrel Cres.                     2        600    4.56 km
 91 Cat St.                            1        500    4.72 km
```

```
What do you think about this rental property?
        Addr: 5 Cat St., # Rooms: 4, Rent/Room: $400, Distance: 3.00 km

Command ('h' for help): sr

Favourite Rental Properties
Address                            # Rooms  Rent/Room   Distance
-------------------------          -------  ---------  ---------
  5 Cat St.                              4        400    3.00 km
 91 Cat St.                              1        500    4.72 km
168 Squirrel Cres.                       2        600    4.56 km

What do you think about this rental property?
        Addr: 5 Cat St., # Rooms: 4, Rent/Room: $400, Distance: 3.00 km

Command ('h' for help): r

This rental property is already on your favourites list

What do you think about this rental property?
        Addr: 5 Cat St., # Rooms: 4, Rent/Room: $400, Distance: 3.00 km

Command ('h' for help): n

What do you think about this rental property?
        Addr: 91 Cat St., # Rooms: 1, Rent/Room: $500, Distance: 4.72 km

Command ('h' for help): l

Rental property deleted

What do you think about this rental property?
        Addr: 168 Squirrel Cres., # Rooms: 2, Rent/Room: $600, Distance: 4.56 km

Command ('h' for help): a

Favourite Rental Properties
Address                            # Rooms  Rent/Room   Distance
-------------------------          -------  ---------  ---------
  5 Cat St.                              4        400    3.00 km
168 Squirrel Cres.                       2        600    4.56 km

What do you think about this rental property?
        Addr: 5 Cat St., # Rooms: 4, Rent/Room: $400, Distance: 3.00 km

Command ('h' for help): q
```