Curtin University — Computing Discipline

UNIX and C Programming (COMP1000)

Semester 1, 2020

Final Assessment

Due: 4.25 hours after commencing, up until 1 PM Saturday 20th June 2020

Weight: 50% of the unit mark.

This is an online **OPEN BOOK Assessment**. There are **Thirteen (13) tasks**. Answer all of them. You have **4.25 hours** to complete this assessment.

You **are** allowed to compile and test your code as you desire. However, it is your responsibility to ensure that you have a working environment you can test in.

1 How To Submit

At the end of your 4.25 hours, you are to submit your code to blackboard.
Your Submission is to consist of a single **tar.gz** file that contains at *minimum* the following files (You can use multiple .c and .h files if you wish):

- main.c
- linkedList.c
- linkedList.h
- Makefile
- auto.sh

Note: Any .o files that are included will be deleted when testing

Warning: No Makefile will result in '0' being awarded to compilation and running

2 Overall Program

The overall goal of the program is to read a file consisting of each line representing a house for sale. This file is to be read into a generic linked list and then filtered by user input. The filtered linked list of properties is then to be saved in a new output file.

3 Things to skip

Below are a few things that you **do not** have to do for this assessment:

- Comment your code (Other standards still needs to be followed)
- The input file being in the incorrect format
- User input being the wrong data type

Task 1 - House Struct (4 marks)

The input file that you will be reading consists of a number of properties (houses). These properties are represented by 4 fields

- bedroom count
- bathroom count
- price
- address

The first 3 values are all integers. The address is a string that will be a maximum of 100 characters long and includes white spaces.

In a header file, create a struct that will be able to represent these fields appropriately.

Task 2 - Linked List (4 marks)

The following program is going to be using a linked list. The linked list must be a generic linked list and you are able to use one you have previously written.

Note: It is assumed you already have one written for the Assignment and the structs practical session.

Task 3 - Reading The File (10 marks)

Function Prototype:

```
linkedList* readFile(char* filename);
```

Write a function called readFile that will create a linked list of properties from the input file. Each line of the file consists of a single house in the following format.

<bedrooms amount>,<bathrooms amount>,<price>,<address>

While the address field is made up of a number, street name, and suburb, it can be read and stored in a single field. The suburb section will also always be CAPITALIZED (e.g 'BENTLEY'). You can assume that the suburb is always a single word.

The function is to return the linked list populated with all the properties. In the case of the file failing to open, it should return NULL.

Note: The only required error checking is that whether the file can be opened correctly

Task 4 - Writing The File (8 marks)

Function Prototype:

```
void writeFile(linkedList* list, char* filename);
```

Write a function called writeFile that imports a linked list and a filename. The purpose of this function is to save a linked list of properties in a file.

The format of the file is to be the exact same as what the input is.

Task 5 - Filter By Max Price (8 marks)

Function Prototype:

```
void filterByPrice(linkedList* list, int price);
/* OR */
linkedList* filterByPrice(linkedList* list, int price);
```

The purpose of this function is to filter the linked list of properties to only contain properties where the price is less than or equal to the imported price.

As shown above, you have 2 options on how to implement this and you are able to pick whichever you feel more comfortable doing.

Option 1 - The first function prototype would be a function that modifies the imported linked list and removes all the unwanted properties from the list. No new list would be created or returned

Option 2 - The second function prototype is where your function would generate a new linked list that only contains the matching properties from the original linked list. A new list is to be created and returned (more memory to free).

Note: In the case where there are no matches, the list should be empty. **Not NULL**

Task 6 - Filter By Suburb (8 marks)

Function Prototype:

```
void filterBySuburb(linkedList* list, char* suburb);
/* OR */
linkedList* filterBySuburb(linkedList* list, char* suburb);
```

The purpose of this function is to filter the linked list of properties to only contain ones where the address is located in a given suburb.

The suburb of an address is located at the end of the string, and will only ever consist of one (1) word. For simplicity, you can assume that a street name will never match a suburb name. The filter is to be case insensitive when doing string matching as well. For example, the inputs "BentLEY" and "BENTLEY" should be a match.

Just like before, you have 2 options on the function prototype however it is highly suggested to do the same one you did for the previous function.

Task 7 - Flash Sale (5 marks)

Function Prototype:

```
void flashSale(linkedList* list);
```

The purpose of this function is to reduce the price of all properties by half. The challenge however is that you **CANNOT** use the divide or multiple operations, but rather have to use bit shifting to half the prices.

Warning: Marks will only be awarded if done with bit shifting

Task 8 - Write a main (12 marks)

Now that all the functions are written, its time to write the main.

Your program is to take in 2 command line arguments being the input and the output filename.

Example:

```
./ucp_exam input.txt output.txt
```

It is then to do the following tasks in order:

- Use the function "readFile" to create a linked list
- Prompt user input to select a filter method and get the appropriate user input needed. (Integer input)
  - 1: filterByPrice
  - 2: filterBySuburb
- Have a 25% chance for the flashSale function to be used. (to be done utilizing random numbers)

- use the function “writeFile” to save the filtered linked list.
- Memory Cleanup.

Note: You are not required to loop through the menu. It only needs to be done once.

Task 9 - Makefile (6 marks)

Have a valid makefile with all required variables and rules.

It should be following the proper standard of the unit and have a clean rule. Failure to have a makefile will result in failure in compilation.

Task 10 - Scripting Time (5 marks)

Create a bash script file called “auto.sh”.

This script is to check if the executable file exists and if it does not, then compile it using the makefile.

After compilation (only if it needs to), run the program with the following predetermined input:

- inputfile: “input.txt”
- outputfile: “output.txt”
- filter: filterBySuburb
- suburb: ”WANO”

The script should also redirect all terminal output from the program to “/dev/null” to prevent it from appearing in the terminal.

Note: Your .c files should not be changed for this, but rather you should be utilizing input and output redirection. (You may need another text file)

Task 11 - Compilation + Execution (10 marks)

These marks will be awarded on how well your program compiles and what components are testable.

Task 12 - Memory Management (5 marks)

Memory errors and leaks will affect your mark here. The goal is to have 0 errors and 0 memory leaks.

Task 13 - Coding Standard (15 marks)

As with any code for this unit, it is to abide by the Coding Standards that have been set from the start.

Marks will be lost for breaching this policy or if there is insufficient amount of code to demonstrate you understand the standard.

4 All Finished

Now that you have finished, it is time to tar up your code and submit it.

Note: .o files don't need to be submitted. They will be deleted when we do the testing.

```
[user@pc]$ tar -czvf <studentID>_Exam.tar.gz Makefile main.c <other files needed>
```

End of Challenge