CSE 320 Spring 2019 HW #2 Part II

March 7, 2019

Deadline: March 18, 23:59 Stony Brook time (Eastern Time Zone)

1 Introduction

Goal of this homework is to help you understand better how memory allocation/management works and give you better understanding of pointers. On top of that, you will also learn how to create your own `Makefile` and practice using `gdb`. This homework consists of two parts. We found that readings are important and crucial in succeeding in the homeworks. Thus, in the first part you are required to read provided material that serves as a basis for doing the second part. The second part of the homework is a programming exercise where you need to apply gained knowledge from the readings.

In this second part you need to implement a simple interactive application that will mimic a database of art collections. You will have a set of warehouses where you can store those collections. In some way, warehouse represent a memory block and an art collection represents an actual payload. To operate that database you should implement an interactive shell where user can input commands. Please read all the sections prior attempting writing the code as it will give you a clearer picture of the whole program.

2 Part I

You program should be interactive. After starting the program you should keep prompting user commands and perform various actions depending on the command. The list of acceptable commands is as follows:

- help
- load warehouse *filename*
- load art *filename*
- printall
- print public

- print private
- add art *name size price*
- delete art *name*
- utilization
- exit

Now let's go over each command.

help. This command should print a help message giving a brief description of each command. Exact format is up to, just make it readable and actually helpful. You can check help messages of Linux tools to get an idea of how a help message may look like, e.g. "ls –help".

load warehouse *filename.* This command should populate your linked list (exact details on the linked list are in the later sections) of warehouses. The *filename* specifies a file name that contains information about warehouses. You should check that such file exists and you can open/read it. Each line in the file represents information about warehouse and have the following format:

ID SIZE TYPE

where *ID* is a warehouse ID and should be a positive number, SIZE is a size/capacity of the warehouse, and *TYPE* defines its ownership, which is either "public" or "private". Please note, that this command can be called multiple times to add more warehouses.

load art *filename.* This command should populate your warehouses with art collections provided in the file with the *filename*. You should check that such file exists and you can open/read it. Each line in the file represents information about art collection and have the following format:

NAME SIZE PRICE

where *NAME* is the name of an art collection, *SIZE* is a size of an art collection, and *PRICE* is a price of an art collection. Please note, that this command can be called multiple times to add more art collections to the database.

printall. This command should print information about all art collection in the database. Each art collection should be on a separate line and should adhere to the following format:

NAME SIZE PRICE

where *NAME* is the name of an art collection, *SIZE* is a size of an art collection, and *PRICE* is a price of an art collection. After all art collections being printed you should print the total worth of art collections on a separate line.

print public. This command should print information about all art collections in the database that are stored in public warehouses. Each art collection should be on a separate line and should adhere to the following format:

NAME SIZE PRICE

where *NAME* is the name of an art collection, *SIZE* is a size of an art collection, and *PRICE* is a price of an art collection. After all art collections

being printed you should print the total worth of art collections stored in public warehouses on a separate line.

print private. This command should print information about all art collection in the database that are stored in private warehouses. Each art collection should be on a separate line and should adhere to the following format:

NAME SIZE PRICE

where *NAME* is the name of an art collection, *SIZE* is a size of an art collection, and *PRICE* is a price of an art collection. After all art collections being printed you should print the total worth of art collections stored in private warehouses on a separate line.

add art *name size price*. This command should attempt to add a new art collection to the database with provided attributes.

delete art *name*. This command should attempt to delete all art collections with matching name.

utilization. This command should print two numbers each on a separate line. First number is the ratio of occupied warehouses versus total number of warehouses. The second number is the ratio of total size of all art collections versus total capacity of all warehouses. Basically, you need to print two utilization statistics.

exit. This command should exit your program. Don't forget to perform any necessary clean up.

NOTE: you program should never crash, get stuck, or have memory leaks. We will deduct points for every such event.

If user happened to input a wrong command then please print that such command is not correct and print a message saying that list of available commands is available through "help" command. Again, exact text to print is up to you.

If at any point in time you will not be able to place a new art collection due to the lack of warehouse capacity then you should print an appropriate message and return to the prompt. In case of a quiet mode (see next section) there is obviously no returning to the prompt.

NOTE: if a string in a file/user prompt consists of multiple words then these words will be surrounded by double quotes.

NOTE: to avoid any capitalization issues just convert everything into lowercase.

REQUIREMENT: you should name you executable as `art_db`.

3 Part II

Now let's discuss supported flags in your program. Your program should support two flags: **quiet mode** initiated by "-q" and printing **sorted output** initiated by "-s".

In the quiet mode you don't fire up the interactive shell. But rather provide two files names. The first file is the file with warehouses information and the second file is the file with art collections information. To make support for quiet

mode easier let's add two more flags: "-w" and "-a" that will be followed by the file name for the file with warehouses information and the file name with art collections information respectively. An example of running program with enabled quiet mode is below:

```
$ art_db -q -w w_filename -a a_filename
```

NOTE: the "-q" flag takes no parameters and should print all the records in the database.

The second, "-s" flag indicates that records should be printed in sorted order. This flag takes a parameter that decides what should be used as a key. If the parameter is "s" then records should be printed in sorted order using the art collections size as a key. If the parameter is "-p" then records should be printed in a sorted order using the art collections price as a key. This flag affects both quiet and interactive modes.

Thus, you program should run as follows:

```
$ art_db [-q -w w_filename -a a_filename] [-s s|p]
```

as before square brackets mean being optional.

4 Part III

Now let's talk about internals of your database. More specifically, how do you need to store records. In this assignment, you need to replicate segregated fits to some extent. We provide `struct` constructs that you need for this assignment in the `warehouse.h`.

As you may notice, each warehouse has a size. So you need to create a linked list to store all warehouses of the same size in the same linked list. However, you may have warehouses of different sizes. Thus, you need to have a separate linked list for every size you have. In order to maintain these linked lists you need to create another linked list. Each node of that linked list should point to the corresponding linked list of a particular size.

Every node of the linked list of a particular size should contain a pointer to the warehouse, pointer to the next node in the list, and a meta data information. The meta data information should contain information about size of the warehouse that this node points to, information if the warehouse is currently occupied, and information if the warehouse is private or public. That meta data mimics memory block header and should be 64 bits wide (one word in a 64-bit system). Additional information can be found in the `warehouse.h`.

The warehouse struct contains information about its ID, size, and pointer to the art_collection unit. The art_collection struct contains information about its name, size, and price.

Please use our `warehouse.h` file.

5 Part IV

Now we will briefly go over inserting art collections into database and removing them.

Similar to the actual memory management you need to search for the linked list that corresponds to the size that can fit an art collection. Once you located such linked list you need to iterate over it and find first non-occupied warehouse to place your art collection there. Obviously, after such placement you need to mark warehouse as being occupied.

You may face a situation when either there is no linked list of the exact size you are looking for or there are no non-occupied warehouses in such linked list. In that case, you simply go to the next available linked list with bigger size and repeat that procedure.

In the unfortunate event when you cannot fit an art collection you need to print an appropriate message. If it was a manual placement of an art collection through interactive shell then after error message you simply return to the interactive shell. However, if the source for the art collection was the file then you need to print an additional error message saying that parsing file has failed and exit the program. As usual, do not forget to clean your data structures and memories.

6 Part V

You may remember that we spoke about coalescing in the class. So your task will be to implement something similar. However, within current setup there is nothing to coalesce as we never move our warehouses yet to other linked lists. So first, when you allocate a warehouse for the art collection and if there is some space left in it (difference between the size of the warehouse and the size of the art collection) then you need to split this warehouse into two and move them to the appropriate linked lists. If there is no appropriate linked list then simply create one. As this operation will result into creating a new warehouse you also will need a new ID for it. Our only **REQUIREMENT** is to make sure that new ID you will choose for it will be unique.

Now we have an ability of splitting warehouses into two. So let's come up with a procedure of coalescing two warehouses. Let's coalesce two warehouse when both of them are not occupied and they are directly surrounded by occupied warehouses. In other words, if you face a situation when there is an occupied warehouse followed by two non-occupied and then followed by an occupied warehouse in the linked list then you need to coalesce two non-occupied warehouses and place them in the appropriate linked list.

7 Part VI

Finally, let's discuss some extra credit opportunities. Both of them target only interactive mode of your program.

Extra Credit 1. In the first extra credit opportunity, you need to implement a new command "print warehouse" that will print to the console a nice layout of the warehouses with them being marked if they are occupied or not. Be creative, we would love to see some ASCII visual magic.

Extra Credit 2. Recall that if you get the art collection from the file and it fails you need to exit the program. In this extra credit opportunity, you need to implement a rollback feature. It means that if you tried to load art collection from the file then you need to return to the interactive shell but also undo all the changes that were introduced to the database while parsing that particular file.

8 Last Remarks

NOTE: Failing to follow any of the requirements will result into 50% grade penalty.

As you may see, you are given a lot of freedom. Basically, we give some information about input, functionality, and restrictions. Everything in between is up to you. Do not hesitate to ask questions and seek out for help (from us) if you need it. We are ready to help.

9 Code Hand-out/in

To start please click on this link so your repository will be created for you. There were cases when GitHub will stuck trying to create a repository for you and it should not take more than few minutes. In such case write me an email so I can manually create a repository for you.

Please note, that this time you need to keep your source files in the folder name "src" instead of having them in the root directory of your repository.

REQUIREMENT: you need to put linked list related code to the file named "linked_list.c" and also place it into "src" folder. Thus, you will have at least three files in that folder. Note, that you are not allowed to include ".c" files in other source files using `#include` directive and they should be compiled using `gcc`.

REQUIREMENT: your `Makefile` should be placed in the root directory of your repository. To compile your code we will simply type `make`.

To submit your code please push it to the GitHub. We will grade the latest push within the deadline.

As before, keep pushing your code, ask us any questions, and good luck.