Sabanci University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Summer 2021-2022

Take Home Exam 2 – Shopping List
Due: 16 August 2022 11.55pm (Sharp Deadline)

DISCLAIMER:

Only checking the sample run cases might not be sufficient as your solution will be checked against a variety of samples different from the provided samples; however checking these cases is highly encouraged and recommended.

You can NOT collaborate with your friends and discuss your solutions with each other. You have to write down the code on your own. Plagiarism will not be tolerated AND cooperation is not an excuse!

Introduction

The aim of this assignment is to practice on linked lists data structures. In this THE, you will implement a dynamic shopping linked list by using a **circular doubly linked list**. In the next sections, we will mention the types of commands that your program should handle along with details about each one.

Input to Your Program

The input to your program is a sequence of commands that will end with an "exit" command. Some commands might have relevant information entered by the user. Those details will be explained in the following sections.

Format of the Inputs

Following are the six types of commands that your program should handle. Note: you should use these commands with the mentioned spelling and in lowercase letters. If the user enters anything other than those ones, your program should print a **predefined** message (please refer to sample runs for more details).

1. **add**
    - This command allows the user to add an item to the list.
    - It will be followed by three inputs.
        1. The name of the item
        2. The quantity (count) of the item
        3. The price of the item
    - You have to search for an item to be added to the list before adding it. If the item to be added is not in the list, then it is straightforward to add it along with the related information.
    - On the other hand, if the item was in the list, then the new quantity of the item will be added to the existing quantity, and regarding the unit price, the unit price of the item will be updated only if the recent price is smaller than the existing one.

2. **delete**
    - This command allows the user to delete an item from the list, if it exists.
        - By deleting an item, we mean all the occurrences of that item, not just decreasing the quantity. The node that contains the information about that item should be deleted from the list. This means that the dynamic memory allocation needs to be released (using delete operator with the pointer of that node, and of course maintaining the connections between the rest of the list after that deleting operation).
    - It will be followed by only one input which is the name of the item to be deleted.
    - If that item doesn't exist in the list, your program should print a **predefined** message (please refer to sample runs for more details).

3. **print**
    - This command allows the user to print the whole list, sorted according to the criteria that will be mentioned later in the document.
    - It does not need any more inputs. i.e., once your program reads that command, it should not wait for more inputs, and should print the full content of the linked list in a **specific** format.
        - If the list is empty, then your program should print a **predefined** message (please refer to sample runs for more details).

4. **totalPrice**
    - This command allows the user to print the total cost of the items in the shopping list. Note here that the total price is the sum of all items prices in the list, and for

each item, the total price is the price of a unit of that item multiplied by the amount (count) of that item.

5. **clear**
    - This command frees the dynamically allocated memory and prints a **predefined** message to indicate that. Note: the command should prevent "memory leak", which is a serious issue.

6. **exit**
    - This command terminates the program. No output messages will be printed.

Notes on the item name, quantity and price:

7. The item name might consist of multiple words and there can be multiple spaces or tabs between the words. Your program should handle these cases and the final item name should have single spaces between the words (and no space in the end :)). A good way to handle this might be reading the console input by *getline* and split this line into words by using a string stream. You can use getline for reading standard input like this:

```
string line;
cin.ignore();
getline(cin, line);
```

   And the string *line* will hold the whole line of the input from the console.

8. All item names that will be used in the program are UPPERCASE. Besides, as mentioned above, for each item, there will be quantity and unit price data. These two fields should be considered as positive integers. You do not need to check for that.

Data Structure to be Used

You **must** use a **circular doubly linked list** for the implementation of the shopping list. Apart from the automated grading, we will also inspect your code in detail. Faking to use a linked list will not help you and you will end up with a grade of 0 directly. So, please do not try to get away with a vector or array implementation.

You can encapsulate the name, quantity and unit price information regarding one item in a struct, as we do for linked list examples in this course. Then, you can basically form a circular doubly linked list of these struct elements. Remember that you should also have some kind of pointer**s** in these structs for connecting them to each other. You are also allowed to keep extra fields in the struct if you feel like so, but these three data fields (and some pointer**s**) should be enough at the minimum.

Details of the Implementation, Commands and Outputs of Your Program

The list needs to be organized according to some measures. For this, you should always have the list *sorted descendingly* with respect to the total estimated prices (i.e., quantity * price) of the items. If two items have equal estimated prices, they should be sorted with respect to their names alphabetically (greater total estimated price comes first; if equal, smaller name comes first).

For the printing commands, *the list needed to be maintained sorted*. In order for your program to find the correct results in these steps, your list should follow the sorting rule (descending with respect to total estimated price and ascending with respect to name, if necessary) after each action taken in adding or deleting commands. For the format which you will show items in the console, you can see the sample runs below.

Last warning: always be reminded to check if pointers are not *NULL* before you use them :)

Sample Runs

Below, we provide some sample runs of the program that you will develop. You have to display the required information and text in the same order and with the same words as here.

TC #	Input	ExpectedOutput	Explanation
1	add BREAD 2 4 print clear exit	Item: BREAD Quantity: 2 Unit Price: 4 The list is cleared.	The first command is ("add"), and it is followed by the name of the item to the list (in a new line). In another new line of the input, two numbers, separated by whitespace, are given. The first number is the quantity of the item, and the second number is the price of one unit of the item. The second command is ("print"). That command has the effect of showing the content of the list on the screen as shown in the *Expected Output* column. Please note the output format as your program should produce the exact one.
2	print delete BREAD add BREAD 5 4 print clear	The shopping list is empty. The item BREAD could not be found in the list. Item: BREAD Quantity: 5 Unit Price: 4 The list is cleared.	- The first command ("print") resulted in printing the first line of the output as the list was empty. - The second command ("delete") was followed by the name of the item to be deleted i.e., *BREAD.* - The rest of the commands are similar to the previous test case.

	exit		
3	add BREAD 1 5 print add BREAD 1 4 print totalPrice delete bread delete BREAD print clear exit	Item: BREAD Quantity: 1 Unit Price: 5 Item: BREAD Quantity: 2 Unit Price: 4 Total price: 8 Unique items: 1 The item bread could not be found in the list. The item BREAD is deleted from the list. The shopping list is empty. The list is cleared.	- (“add”) and (“print”) commands were explained in previous test cases. - The second “add” command added one unit of BREAD to the existing amount and updated the price as the new price is cheaper than the existing one. - The “totalPrice” command shows the total price of all the items in the linked list. - When trying to delete an item with the name “bread”, a relevant message is printed to indicate that the item is not in the list. However, when deleting the item” BREAD”, it was successfully deleted, and the relevant message is printed. - The rest of commands are similar to the previous test cases
4	add FRUIT SALAD 1 22 PRINT print Add add BIG FRUIT SALAD 1 40 print add MILK 1 24 print delete FRUIT delete FRUIT SALAD print clear exit	Invalid command. Item: FRUIT SALAD Quantity: 1 Unit Price: 22 Invalid command. Item: BIG FRUIT SALAD Quantity: 1 Unit Price: 40 Item: FRUIT SALAD Quantity: 1 Unit Price: 22 Item: BIG FRUIT SALAD Quantity: 1 Unit Price: 40 Item: MILK Quantity: 1 Unit Price: 24 Item: FRUIT SALAD Quantity: 1 Unit Price: 22 The item FRUIT could not be found in the list. The item FRUIT SALAD is deleted from the list. Item: BIG FRUIT SALAD Quantity: 1 Unit Price: 40 Item: MILK Quantity: 1 Unit Price: 24 The list is cleared.	- The "Invalid command.“ message was printed as response to invalid command (PRINT). - Note here the item name consists of multiple words. - Also note the order in which the items were added (descendingly according to total item price) - When trying to delete an item with name (“FRUIT”), the program printed a predefined message as there was no item with that name in the list.

Some Important Rules

In order to get full credit, your program must be efficient, modular (with the use of functions), well commented and properly indented. Besides, you also have to use understandable identifier names. Presence of any redundant computation, bad indentation, meaningless identifiers or missing/irrelevant comments may decrease your grade in case that we detect them. When we grade your THEs, we pay attention to these issues. Moreover, **we may test your programs with very large test cases**. Hence, take into consideration the efficiency of your algorithms other than correctness.

Sample runs give a good estimate of how correct your implementation is, however, we will test your programs with different test cases and **your final grade may conflict with what you have seen on CodeRunner**. We will also **manually** check your code, indentations and so on, hence do not object to your grade based on the **CodeRunner** results, but rather, consider every detail on this documentation. **So please make sure that you have read this documentation carefully and covered all possible cases, even some other cases you may not have seen on CodeRunner or the sample runs**. The cases that you *do not need* to consider are also given throughout this documentation.

Submit via SUCourse ONLY! Paper, e-mail or any other methods are not acceptable.

The internal clock of SUCourse might be a couple of minutes skewed, so make sure you do not leave the submission to the last minute. In the case of failing to submit your THE on time:

"No successful submission on SUCourse on time = A grade of zero (0) directly."

What and where to submit (PLEASE READ, IMPORTANT)

It'd be a good idea to write your name and last name in the program (as a comment line of course). Do not use any Turkish characters anywhere in your code (not even in comment parts). If your full name is "Duygu Karaoğlan Altop", and if you want to write it as comment; then you must type it as follows:

```
// Duygu Karaoglan Altop
```

You should copy the full content of the .cpp file and paste it into the specified “Answer” area in the relevant assignment submission page on SUCourse. ***Please note that the warnings are also considered as errors on CodeRunner, which means that you should have a compiling and warning-free program***.

Since the grading process will be automatic, you are expected to strictly follow these guidelines. If you do not follow these guidelines, your grade will be zero (0). Any tiny change in the output

format will result in your grade being zero (0), so please test your programs yourself, and against the sample runs that are available at the relevant assignment submission page on SUCourse.

In the CodeRunner, there are some visible and invisible (hidden) test cases. You will see your final grade (including hidden test cases) before submitting your code. There is no re-submission. You don't have to complete your task in one time, you can continue from where you left last time but you should not press submit before finalizing it. Therefore, you should make sure that it's your final solution version before you submit it. Also, we still do not suggest that you develop your solution on CodeRunner but rather on your IDE on your computer.

You may visit the office hours if you have any questions regarding submissions.

How to get help?

You may ask your questions to TAs or to the instructor. Information regarding the office hours of the TAs and the instructor are available at SUCourse.

Plagiarism

Plagiarism is checked by automated tools, and we are very capable of detecting such cases. Be careful with that. Exchange of abstract ideas are totally okay but once you start sharing the code with each other, it is very probable to get caught by plagiarism. So, do NOT send any part of your code to your friends by any means or you might be charged as well, although you have done your THE by yourself. THEs are to be done personally and you have to submit your own work. **Cooperation will NOT be counted as an excuse.**

In case of plagiarism, the rules on the Syllabus apply.

Good Luck!
Ahmed Salem, Duygu Karaoğlan Altop