CS 162 Computer Science II

Homework Assignment 5 - Linked Lists 2

Academic Integrity

You may NOT, under any circumstances, begin a programming assignment by looking for completed code on StackOverflow or Chegg or any such website, which you can claim as your own. Please check out the Student Code of Conduct at PCC.

The only way to learn to code is to do it yourself. The assignments will be built from examples during the lectures, so ask for clarification during class if something seems confusing. If you start with code from another source and just change the variable names or other content to make it look original, you will receive a zero on the assignment.

I may ask you to explain your assignment verbally. If you cannot satisfactorily explain what your code does, and answer questions about why you wrote it in a particular way, then you should also expect a zero.

In this assignment you will create an application to mange a list of items. The list will be a single link, forward linked list. The user will be able to insert, remove, update, and search the list for items.

CS 162 Computer Science II

Homework Assignment 5 - Linked Lists 2

Purpose

The focus of this assignment is working with dynamic linked lists and dynamic arrays.

After completing this assignment you will be able to:

- Write code using classes and objects that use dynamic linked lists and arrays
- Insert, update, remove, and search for elements in a linked list
- Use dynamic C-strings
- Use dynamic single link (forward) linked lists
- Avoid memory errors such as seg fault and memory leaks
- Organize source code in multiple header and implementation files (.h and .cpp files).
- Implement and use accessors and mutators for a class

Overview

This Homework Assignment includes zyBook exercises and a programming assignment. Please read all of the instructions carefully - there are a lot of details that you need to know!

This assignment is to create a program to manage a shopping list, i.e. a list of items to purchase at a store. The program maintains the shopping list as a single link, forward linked list of `item` objects. The user can add items, update items, remove items, and search for items in the list.

The program should not impose *any* limits on user input or data. For example, the list can contain any number of items, and there is no limit on the length of a c-string. All arrays must be right sized (start with nullptr and resize by +1 whenever a new value is added to the array, resize by -1 when an element is removed). All linked lists must be right sized, and contain no “dummy” or unused elements.

All lists must be single link, forward linked lists. All strings must be dynamic c-strings. You may not use *any* static or fixed-length arrays.

zyBook Exercises

You should complete the following zyLabs:

1. zyLab 22.2 Inventory. This lab introduces removing an item by name from a list.

Once you have completed this zyLab with a score 100% you will need to create a screenshot image of your completion statistics from zyBook, and include that in your algorithm document for this assignment. Note that “successfully completed” means that you received 100% on both labs.

CS 162 Computer Science II

Homework Assignment 5 - Linked Lists 2

The zyLabs a required element of this assignment - your submission is not complete unless the success statistics are included in your algorithm document.

Programming Task

Introduction

The purpose of the program is to maintain a list of items for the user. Your program should present the user with a menu of options, and **repeat** the options until the user decides to quit the program. The options must include:

- Quit the program
- Erase all items from the list (resulting in an empty list)
- Add an item to the list
- Remove an item from the list
- Update an item in the list
- Display all items in the list

The list will be a linked list of `item` objects. Each `item` object represents one item, and contains the following information about that one item:

- The name of the item (a dynamic c-string)
- The quantity of the item (an unsigned integer)
- The price of the item (a float)

As in previous assigments you must create classes, namly a `list` class to maintain the list and implement the list operations (insert, remove, find, erase), and an `item` class that represents a single item. The list maintained by the `list` class is a linked list of `item` objects.

These instructions do **not** include details of the design of the classes - you should design them in conformance with the style guidelines and programming practices learned in class. You are free to design the details of the user interface as you see fit, as long as you satisfy the requirement for a menu-driven program that supports the option requirements.

All object data must be private, all classes must include the default constructor, copy constructor, copy assignment operator, and destructor (whether you think you need them or not). The program must run with no memory errors (memory leaks or any other memory errors). Do not include any operator overloads except the copy assignment operator, and do not include any `friend` functions or classes (forget you ever heard about the `friend` keyword!).

List Requirements and Operations

You only need one linked list in this assignment (the list of items). This list should be ordered by item name, in ascending order, and should not contain any duplicate item names.

Note that “ordered by item name” means that each new item **must** be inserted into the list at the position required to maintain the list order - you are **not** allowed to sort the list after inserting a new item.

“Should not contain any duplicate item names” is pretty self-explanatory: If the user tries to add a new item with the same name as an existing item in the list then no insertion should occur. In this situation you may either (a) reject the insertion completely and return to the main menu, or (b) update the existing item with new information provided by the user.

Additional requirements for the program options include:

- The **quit option** should end the program after deleting all dynamic memory. No further user input should be required after the user selects the quit option.

- The **erase option** should delete all linked list elements and reset the head pointer to nullptr.

- The **add option** should prompt the user for item information (name, price, quantity), and if no item with the same name is already in the list, add the new item in the position needed to maintain list order. If an item with the same name already exists in the list then do not add a duplicate item.

- The **remove option** should require the user to select a match criterion - name, or price, or quantity - and then enter a match value. The remove option should remove from the list all items that satisfy the criterion.

  In other words, if the user selects name as the match criterion, then remove all items that have the given name. If the user selects price as the match criterion, then remove all items that have the given price. If the user selects quantity as the match criterion, remove all items that have the given quantity.

- The **update option** should require the user to select a match criterion - name, or price, or quantity - and then enter a match value, followed by a new value. The update option should update all items in the list that satisfy the match criterion, and change the value to the new value.

  In other words, if the user selects name as the match criterion, then find all items that have the given name (there should be only one) and change the name to the new value. If the user selects price as the match criterion, then find all items that have the given

price and change the price to the new value. If the user selects quantity as the match criterion, then find all items that have the given quantity and change the quantity to the new value.

- The **display option** should first display a line containing the total number of items in the list and the total cost of the list. Following that the option should display the information for each item on a separate line, include the item name, item price, item quantity, and total cost of the item.

  The *total cost of an item* is the price multiplied by the quantity. The *total cost of the list* is the sum of the total cost of all of the items in the list.

Classes

`item` Class

The private data of the `item` class must include the name of the item (a dynamic c-string), the desired quantity of the item (an unsigned integer), and the price per unit of the item (a float) in dollars (e.g. 0.50 is fifty cents, while 1.23 is one dollar and twenty three cents).

For example, an `item` object may contain the data:

```
name == "Apple"
quantity == 12
unit_price == 0.50
```

The total cost of this item is 12 * 0.50 = 6 dollars.

`list` Class

The `list` class is a container class that maintains a linked list of `item` object that is ordered and contains no duplicates.. The linked list must not have any empty nodes. The ordering should be by item name, in ASCII order. For example, if the list contains Apples, Oranges, and Grapes then the order would be

1. Apples
2. Grapes
3. Oranges

Sorting is **not** permitted. All insertions should be made at the position in the list required to maintain the list order. For example, if the first element inserted into the list is Grapes, then Apples would be inserted before Grapes, and Oranges would be inserted after Grapes.

Linked list pointers must be available in each element of the linked list, and a common way to provide linked list pointers for objects is to “wrap” them in a simple struct. This is the technique

used in Lab 21.10, and is fine for this assignment. A separate module for the struct is NOT required.

`main` Module

The `main()` module may contain additional user-defined functions that are not members of a class, such as custom `getline()` or `menu()` functions.

The main module should present the user with a menu of options to manage their shopping list. The user should be allowed to input in upper case, lower case, or mixed case. The user may enter invalid input, and if so then display an error message and repeat the manu of options. Continue processing options until the user exits the program.

The options must contain at least the following:

- Quit the program
- Erase all items from the list (resulting in an empty list)
- Add an item to the list
- Remove an item from the list
- Update an item in the list
- Display all items in the list

Additional Requirements For This Assignment

1. You may use features from the following libraries:
    a. <iostream>
    b. <fstream>
    c. <cstring>
    d. <cctype>

    You may not use any C++ strings, algorithm classes, container classes, or memory management classes (or features of these classes).

    Let's face it - what I am asking you to do in this assignment can be done much more easily if you were allowed to use all of the features of C++. But you are **not** allowed - one goal of this assignment is to practice with dynamic memory - so you get to do things "the hard way".

2. The linked list must be ordered by item name, and no sorting is permitted. All insertions should be made at the position required to maintain list order.

3. Your program must compile and execute on the PCC Linux server, and run with no memory or behavioral errors.

4. All arrays must be dynamic. Do not use any static arrays. All c-strings must be implemented in dynamic char arrays.

5. All arrays must be "right sized" at all times. There should never be any unused elements in any array. This means the array pointer must be initialized to `nullptr`, and new memory is allocated only when a new element is added to the array.

6. Array and list sizes should increase by 1 when a new element is inserted, and decrease by 1 when an element is removed.

7. All pointers must be initialized to `nullptr`. Arrays and lists with no data should not have any allocated memory.

General Requirements For Classes

As you create classes for this assignment, please keep in mind that there are some members that you **must** implement in every class. You may implement additional members as needed, but all classes must:

1. Implement a *default constructor* that initializes the object to an "empty" or "no data" state. The object is said to be in its "default state" when the default constructor is done.

   The "no data" state for pointers is to have a value of `nullptr`. All pointers must be set to `nullptr` by the default constructor.

2. Implement a *copy constructor* that initializes the object to be identical to a source object. The copy constructor must make a copy of all dynamic data, so that each object has its own private copy of the data. The original source object is not changed in any way.

3. Implement a *destructor* that resets the object to its default state, i.e. the same state as is set by the default constructor. The destructor is responsible for ensuring that all dynamic memory used by the object is released, so typically the destructor will include `delete` statements.

4. Implement a *copy assignment* operator that assigns the object to be identical to a source object. The copy assignment operator must make a copy of all dynamic data, so that each object has its own private copy of the data. Note that in order to prevent memory leaks the copy assignment operator must delete any existing dynamic memory before creating new dynamic memory for the copies.

5. Use only `private` data - all variables and constants must be `private` in the class. This may mean that accessors and mutators are required as well.

In addition to the “must do” items listed above, there are some “never do” items that are forbidden in CS 162:

1. Do not use `friend`. This is not part of CS 162.

2. Do not create class hierarchies or implement inheritance or virtual functions. This is not part of CS 162.

3. Do not use lambda functions in this assignment. These are discussed in CS 162, but you won’t get to use them in assignments.

4. Do not use static class members. Again, this is not a CS 162 topic.

5. Do not use template classes or template functions. Yet again, this is not a CS 162 topic.

General Requirements for Dynamic Memory

Any program written in CS 162 that uses dynamic memory should satisfy the following requirements:

1. Do not use smart pointers, helper classes, or any features from the `<memory>` library. Use “naked” pointers only.

2. Dynamic memory must be allocated and released using C++ `new` and `delete` operators. Do not use `malloc()`, `calloc()`, `realloc()`, `free()`, or any other C specific memory management.

3. Two different objects **must not share dynamic data**, i.e. two different objects should not both have pointers to the same dynamic data. This affects accessor design when private data is dynamic, since a **copy** of the dynamic data must be returned by the accessor, rather than having the accessor provide access to the object’s private data.

4. All dynamic variables must be initialized when they are allocated, just as all static variables must be initialized when they are declared

5. All dynamic memory allocated by your code must be released by your code before the `main()` function ends. SImply put, memory leaks are bad (for your grade).

6. Your program should not reference any unallocated memory. Do not reference memory blocks after they have been released, and do not reference addresses outside of

allocated memory blocks.

7. Your code must not have any memory errors when executed. Use `valgrind` or a similar profile tool to verify that your program does not have any memory errors.

Submission

1. Complete the zyBook exercises
2. Open the Algorithmic Design Document, make a copy, and follow the steps to create your algorithm.
3. Export your algorithm document to a PDF file named `algorithm.pdf`
4. Create your source code files, build and test your program.
5. All code must follow the CS 162 C++ Style Guide guidelines.
6. Transfer your source code to the PCC Linux Server
7. Compile and execute your code on the PCC Linux server
    a. Type "`script output.txt`" on the command line and it will start recording your session in a file called "`output.txt.`"
    b. Run your program
    c. Type "`exit`" to stop recording.
8. Upload your source code files, algorithm document, and output log file to D2L. Please upload them as individual files - do not zip or tar them into a single archive file.

General Requirements for All Homework Assignments

- ❏ Code may be written using any development environment, but must use Standard C++ and must be successfully tested on the PCC Linux server. Code usually is graded on Linux, so if your program doesn't work on Linux then it doesn't work.

- ❏ PSU-bound students are strongly encouraged to do all development on the PCC Linux server.

- ❏ Open the Algorithmic Design Document, make a copy, and follow the steps to create your algorithm. Be sure to include your screenshot of the zyBooks completion in the algorithm document.

  Download the algorithm document as a PDF file, and submit the PDF file to D2L as part of this assignment.

- ❑ You must express your algorithm as **pseudocode. Do not use a flowchart in CS 162**. Please note that your pseudocode must follow the syntax requirements shown in the document - you may **not** use C++ as a substitute for pseudocode.

Additional Support

- ❑ Post a question for the instructor in the Ask Questions! area of the Course Lobby.

Sample Run

The following is a sample run, with emphasis on the word “sample”. You are not required to build your user interface as shown here.

```
>main
Welcome to Shopping List Maintenance!

Please enter one of the following options:
1 - Quit the program
2 - Erase all items from the list (resulting in an empty list)
3 - Add an item to the list
4 - Remove an item from the list
5 - Update an item in the list
6 - Display all items in the list
Enter Option: X
?Invalid option!

Please enter one of the following options:
1 - Quit the program
2 - Erase all items from the list (resulting in an empty list)
3 - Add an item to the list
4 - Remove an item from the list
5 - Update an item in the list
6 - Display all items in the list
Enter Option: 3
Enter name, quantity, price: Bars Of Soap, 12, 1.25
Item “Bars of Soap” added

Please enter one of the following options:
1 - Quit the program
```

```
2 - Erase all items from the list (resulting in an empty list)
3 - Add an item to the list
4 - Remove an item from the list
5 - Update an item in the list
6 - Display all items in the list
Enter Option: 3
Enter name, quantity, price: Apples, 12, 0.25
Item “Apples” added

Please enter one of the following options:
1 - Quit the program
2 - Erase all items from the list (resulting in an empty list)
3 - Add an item to the list
4 - Remove an item from the list
5 - Update an item in the list
6 - Display all items in the list
Enter Option: 6
2 Items in the list. Total cost is $18.00
Apples (12 at $0.25) Total cost $3.00
Bars of Soap (12 at $1.25) Total Cost $15.00

Please enter one of the following options:
1 - Quit the program
2 - Erase all items from the list (resulting in an empty list)
3 - Add an item to the list
4 - Remove an item from the list
5 - Update an item in the list
6 - Display all items in the list
Enter Option: 5
Enter match criterion: 1 for name, 2 for qty, 3 for price: 2
Enter quantity to match: 12
Enter new quantity: 10
Updated 2 items

Please enter one of the following options:
1 - Quit the program
2 - Erase all items from the list (resulting in an empty list)
3 - Add an item to the list
4 - Remove an item from the list
```

```
5 - Update an item in the list
6 - Display all items in the list
Enter Option: 6
2 Items in the list. Total cost is $15.00
Apples (10 at $0.25) Total cost $2.50
Bars of Soap (10 at $1.25) Total Cost $12.50

Please enter one of the following options:
1 - Quit the program
2 - Erase all items from the list (resulting in an empty list)
3 - Add an item to the list
4 - Remove an item from the list
5 - Update an item in the list
6 - Display all items in the list
Enter Option: 1
Goodbye!
>
```