CS 474 – Object Orient Languages and Environments

Programming Project 2, Spring 2016

Inheritance of Linear Linked Data Structures

Due: Tuesday, March 1, 2016 at 11:59 pm

This project is to be written using the C++ Language. You may work on this project by yourself or with one other person. When working on the project with another person, both students will earn the same grade on the project. Only one submission of the project is to be turned in for the final grading (so make sure it clearly indicates the names of the students involved.)

For this assignment, you are to implement four linear data structures using linked lists in C++. All four data structures are to be connected in a single inheritance hierarchy. The exact form of the hierarchy is left up to you Failure to have all four connected together in a single inheritance hierarchy will result in a grade of zero for the assignment. Since this project is all about writing your own linked list classes, use of similar classes from the C++ Standard Template Library is not allowed. Use of other C++ Standard Template Library classes that are not related to linked lists are allowed.

Note: that you will need to turn in a UML Class Diagram showing the inheritance hierarchy by Thursday 2/25/16 by 11:59pm. Submission of the UML Diagram will also be used to determine who is working with another student. So again, clearly indicate who is working on the project. Group members cannot be changed after this time. Also, UML Diagrams turned in by a single student are assumed to be working alone. If a student does not submit a UML Diagram, it will be assumed that the student is working alone.

The following data structures and operations are expects to be performed by this project. To keep things simple, we will just use a single integer value as the only data item in each node of the linked list. We will allow duplicate node values. Also note that the operation "names" used below are all the same which will reflect the command names for the user interface. You may name the class methods/operations with whatever name you deem is proper.

1. an unordered list
   - Get
   - Insert
   - Remove
   - List
   - Size

2. an ordered list in increasing order
   - Get
   - Insert
   - Remove
   - List
   - Size

3. a queue
    - Get
    - Insert
    - Remove
    - List
    - Size

4. a stack
    - Get
    - Insert
    - Remove
    - List
    - Size

For the project, you are to maintain 8 different lists. Each list could be anyone of the above 4 types. The exact type of list is determined by the user commands. The user commands are the following. Note that the commands could be given in either upper or lower case.

- q – Quit the program

  The program is to stop executing after some appropriate message has been displayed. You may prompt the user with an "Are you sure you want to quit?" message, but this is not required.

- h – Help

  The command causes some help message to be displayed informing the user about the various commands being used for this program.

- a – About

  This command causes some information to be displayed about the creators of the program. (i.e. name, netID, etc)

- s <listNum> <type> - Set list number <listNum> to be of type <type>

  This command will initialize/set the given list to the specified type. The value of <listNum> must be in the range from 1 to 8. The value of <type> must be one of: u, o, q, or s. Where u implied an unordered list, o implies an ordered list, q implies a queue, and s implies a stack. When the s command is given for a non-empty list, the current values in the list should all be removed. I.E. the list will be empty after the s command is completed.

- i <listNum> <val> - Insert value <val> into list numbered <listNum>

  This command will insert a value into the specified list. The value of <listNum> must be in the range from 1 to 8. For an unordered list and a stack, the value is to be inserted at the front of the list. For an ordered list, the value is to be inserted at its proper sorted position. For a queue, the value is to be inserted at the end of the list.

- r <listNum> - Remove a value from list numbered <listNum>

This command will remove a value from the specified list. The value of <listNum> must be in the range from 1 to 8. For an unordered list and an ordered list, prompt the user for the position in the list of the value to be removed. The first node in the list is position value of 1. For a stack and a queue, remove the first value in the list. If the list does not contain the needed position, display an appropriate error message.

- g <listNum> - Get a value from list numbered <listNum>

  This command will get/retrieve (and display) a value from the specified list. The value of <listNum> must be in the range from 1 to 8. For an unordered list and an ordered list, prompt the user for the position in the list of the value to be accessed. The first node in the list is position value of 1. For a stack and a queue, access the first value in the list. If the list does not contain the needed position, display an appropriate error message. The list should never be modified by this command.

- l <listNum> - List information about list numbered <listNum>

  This command is to show the current contents of a specified list. The value of <listNum> must be in the range from 1 to 8. The contents of the list should be shown with the item at the front of the list listed first. The total number of items in the list should also be displayed .

- f <filename> - Read commands from the file in <filename>

  Execute the commands from the file as if they were typed in at standard input. The file may contain another f command, so you will need to verify and prevent against a recursive infinite loop when using the f command.

- # comment - Ignore the information contained on this line

  This command is created so that you can document the information/commands in a file created to for use with the f command. However, there is nothing to stop this command from being entered through standard input.

Any other input line should result in an appropriate error message.

Programming Style and Restrictions

Your program must be written in good programming style. This includes (but is not limited to) the follow.

- Multiple Source Code Files
- Meaningful Variable Names
- Proper Indentation of Code
- Blank Lines between Code Sections
- Use of Methods and Functions
- Use of Multiple Classes
- In-Line Commenting
- Header Comment for the File
- Header Comments for each Method, Function and Class.

The work you turn in must be 100% your own (or your own group if working with another student). You are not allowed to share code with any other person outside of your “group” (inside this class or not). You may discuss the project with other persons/groups; however, you may not show any code you write to another person/group nor may you look at any other person’s/group’s written code.

This program is expected to be written using multiple .cpp files and at least one .h file. The use of a makefile is also expected to compile your program. The linked list classes need to be in separate file(s) than the main() function. The linked list classes would ideally each have their own .h and .cpp files; however, it is acceptable to have the entire linked list class hierarchy to be in one .h/.cpp file pair. The file that contains main() should have the remaining code for the user interface.

Project Submission

Directions will be post on the class web page for submission of the project.