COP 3540 – Data Structures with OOP

Project 3 – Fall 2009

Due: 26 October (Monday) at 2pm

Doubly-Linked Lists

Using NetBeans 6.7, you are to write a Java program using OOP principles to accommodate the following functionality

Assignment #3

Objectives:

Provide student with additional experiences with file input output.
Provide student with exercises in learning UML (architectural design)
Provide student with exercises in developing detail design pseudocode
Provide student with exercises in Javadoc and its various formats
Provide student with exercises in building a doubly-linked list of State objects.
Provide student with experience in inserting, deleting, and searching the doubly-linked list
Provide student with exercises in updating a linear linked list

Functionality:

Given a sequential file, **States.Fall2009.txt** on my web page, you are to build a doubly-linked list. You are to update the doubly-linked list using a transaction file (for testing) posted on this web site. The test file is: **StateUpdate.Fall2009.txt.** You are familiar with the formats of the inputs and they are also described below.

So, to get this started:

Task 1: Build an array of State objects from the input data set. (same as programs 1 and 2)

Read the input data set one record (line) at a time; create a State object from each line and insert this State object into an **array** of State objects in **main**(). You've done this a number of times.

Task 2: Sort the array of States using an Insertion Sort based on state name. (only slightly different from Program 1 and 2)

Task 3: Display the array of sorted State objects. (same as programs 1 and 2)

Skip a few lines and display the sorted list of State objects. The header should read: Array of State Objects – Sorted via Insertion Sort.

Task 4: Build the doubly-linked list. Using the array of sorted State objects, you are to use this array as input and – from this array - build a doubly-linked list of state objects, such that all nodes (State objects) will appear in the same order as they are in the sorted

array. (You will need the **insertLast**() routine to build this doubly-linked list so as to maintain the sorted ascending order. Routines are in book; discussed in class.)

Task 5: Display the Doubly-Linked List using Forward Pointers. After building the doubly-linked list, skip a couple of lines, display the doubly-linked list with the header: Doubly-Linked List of States Using Forward Pointers. Skip a line after this header, and **display** the nodes - one per line. Display the State, its abbreviation, its population, and the region number. Space these out 'nicely' on a line and align with the column headers.

Task 6: Display the Doubly-Linked List using Rearward Pointers. Use almost the same header**,** you are to **display** the same list using the **rear** pointers. Start at the end of the list and using the rear pointers, display the list (same format) advancing to the front of the list. Each line is formatted as above. Header is to be: Doubly-Linked List of States Using Rear Pointers. This means that you are to display the linked list in reverse order starting with the 'last' state and proceeding to the first state in the linked list.

Task 7: Now, here's the fun…Update the doubly-linked list. Using the inputs provided to you in **StateUpdate.Fall2009.txt**, you are to update the linked list. You may access this file one 'record' at a time and perform the update to the linked list or build an array and then process them one at a time. It doesn't matter. Here are the criteria: Please look at the records in this file.

If the Code in position 1 is an A (meaning **Add**), this means you are to attempt to create an object and add it to the linked list in its proper place. Validation: But, in searching to find the appropriate place, if a link with the same state name **already exists,** you cannot perform this update. You are to print out a header that says: Dupe Add Attempted (left justified). Underneath this line, print the input transaction also left justified. Skip two lines (must have two blank lines prior to the next output) after this.

If the Code is an A and you find the appropriate place to add the link, then do so. You are create a full link as those you already have. Then, display a single header line that says: (left justified) State, Capital Population Region (only). Skip to the next line and print the attributes just cited from the new node – left justified and nicely spaced. Skip two lines after doing this as above.

If the Code is a C (for **Change**) and you **find** the link, you are to **only change** the population and print it with the header: as above: State Old State Population New State Population, and Number of Nodes Searched. Underneath, print the data for the changed node. All formatting is left justified as previously explained. If, on the other hand, you **do not find** the desired node to change, then you must print out "Link Not Found." Underneath this left justified text, print the input transaction. Skip two lines after this.

If the Code is a D (for **Delete**), then (as usual) two things can happen. You can **find** the node to delete. In this case, Print out the header: Link to be Deleted Number of Nodes Searched and then the name only of the state deleted underneath followed by the number of nodes searched to find this node. If, on the other hand, you **cannot find** the link to delete (that is, it is not in the linked list), then you are to print out the header: 'Link Not Found." Underneath this left justified text, print the input transaction. Skip two lines after whatever action you take.

All transactions will be either A, C, or D type. There will be no illegal transaction types in the input file.

Task 8. At the end of all this, you are to print out the linked list using forward pointers. Print out the header: Linked List Printed using Forward Pointers and underneath, print out all the links and their attributes, one per line, as you have done several times.

You have PLENTY of time if you start right away and work slowly and methodically.

Procedure:

I urge you to tackle this problem incrementally. Using a small sample of the input file to test your procedure. Then build the entire linked list. Verify as you go. Use the toString method or other display method to your advantage and VERIFY that you are in fact building the doubly-linked list correctly!

Deliverable: Your zipped folder to me MUST include copies of your data files. I will need these to run your program and to test it. Do not provide me with output your program generates. I will get your program to generate your outputs. As noted, your outputs will be displayed on the screen. **I very well may, however, substitute a different UpdateStates file when I test your program.**

You are to zip all files in your COP3540 folder being certain to have a Main class and your project named, project3yourname as expected. Zip your project file and submit it to me via Blackboard Assignment links as you have done in the past. Check out your work prior to submitting it to me.

Grading – Project 3 – COP 3540
Fall 2009
Program is worth 120 points

Source Code – 20 points

Looked at closely:

Indentation; Internal comments; Scope terminators; Overall program documentation.