CPSC 131-7 Fall 2015

Project2 – Story Backlog List Handler

Introduction

In this project you will write a handler program for Stories using Linked Lists.

The General Problem

You are part of an agile team that is involved in a Tag Team Pair Programming exercise. You have already handled the situation with respect to your teams at your desks in Project 1 using Arrays. In Project 2 you will handle the List of Stories that describes the problems to be solved for the Class with Singly Linked Lists. You will handle manipulations of these Story Lists by iterators.

Key Points[1]

1. "What is the programing problem, what are they supposed to do to solve it?"

> The programming problem is to implement the Singly Linked List manipulation algorithms represented in Stories you have created in class and those you have made up yourself in a unified well designed program that encompasses as much of the functionality in the Stories as possible. The program is meant to help you to learn and give you practice in the manipulation of Linked Lists in C++ by applying Abstract Algorithms in a specific Scenario that you are immediately familiar with through our exercises in class.

2. "Clarify details as to what the program will do, including functions with descriptions, classes/structures, and any other expectations of the program."

> You may structure your program in whatever way that you see fit that embodies the functionality of the Stories about the "Story Backlog" you receive from other teams and those you made up yourself within your team. However, good design is one of the criteria used in the assessment of the written program that implements that functionality. It is expected that the program will contain a Story object with story nodes. The program needs to allow the user to select which functions that they will exercise, and then allow the user to put in appropriate data. Then, outputs need to be generated by the program that show that the program has correctly manipulated the data that needs to be given to the user. The program is expected to run without error and perform correct data transformations on the Singly Linked List of Stories.

[1] This section was requested by Andrew Huynh for clarification the project assignment.

However, if you cannot get your program to run without error, you should turn it in on the date due and then again (after you have made further corrections) by the latest acceptable date in order to receive partial credit. Make sure that there are error reports that show where the errors are in your program that you could not solve.

3. "What to turn in, how to turn in, formatting, and ordering."

Turn in a program by uploading it to Titanium. Your program should implement the functionality in your Stories along with supporting documentation as described. If your PDF is too large to upload into Titanium, break it up into pieces and group them. Also, upload your source code in a zipped file as well. Your file titles should contain a name like **CS131-7YourFullName2015mmddProject2.pdf or zip**. The source code should appear in both formats. The zipped format is just in case I decide to try to run your code. For the most part I will be looking at the PDF file of the source code and other supporting text within the PDF, for instance, test results. A single PDF is used so that everything that is needed to grade your submission appears together.

You may also be asked to bring your finished working program to class on a laptop after the online submission date in order to demonstrate it to others in the class.

4. "Optional things to include for more credit or in case there are any errors."

There is a section in this document that says what kinds of optional material you might include. Anything marked optional is truly optional and does not need to be done to get an A. If you have extra time and you wish to try some of the mentioned optional features they will be taken into account in grading. But the main assignment is the implementation of as much of the functionality described in the Stories as possible, but condensed into a well-designed C++ program.

5. "Due date and a reminder that they can contact you if there are any questions or things that need to be clarified"

The Project 1 has a deadline on Monday 9/21/2015, 1pm. There were three weekends given for that assignment.

Project 2 is being assigned on Tuesday, September 15th in class (9/15/2015)

There are three weekends between the assignment date and the due date for this assignment. However, it is recommended that you do not wait until just before the due date to do this assignment because Project 3 will be assigned before that time.

Project 2 has a deadline of Monday October 5, 1pm (10/5/2015).

These are meant to be two week assignments but I am giving extra time and overlapping the assignments so that you can adjust your workload as necessary to fit into your schedules with other classes.

You can contact me before or after class during office hours, by email, or at 714-202-7141 voicemail. Be sure and leave your name and a telephone number if I need to call you back.

You can also contact Andrew Huynh for help, especially for questions concerning C++ programming during his office hours.

Algorithm Description Document for Stories Backlog

An Algorithm Description Document (ADD) that contains pseudocode and diagrams of the Singly Linked List regarding Stories that comes out of Exercise 2 will be provided for you either in part or wholly depending on how much is accomplished by the teams. This assignment is also based on an Abstract ADD for Singly Linked Lists done by the Instructor, which has been given to you. Between the ADD from the Exercise (complete or incomplete due to time considerations), the Abstract ADD from the Instructor, and other sources provided on the Class website, there should be enough information to write a program that implements Singly Linked lists for Stories.

Story Board Backlog for Singly Linked Lists about Stories

This set of Stories contains the Stories for the Exercise 2 we did in class. In this Exercise 5 stories were assigned to each team which were to be worked on by each team and then explained to the class. Whatever the outcome of that exercise, the entire set of stories is part of the assignment for each student writing a program for Project 2. It is recommended that you do Algorithm Sketches for all the stories for which you do not get sketches from other teams.

CS131-7 Exercise 2.1 Story: Fill Story Backlog (required)
Insert a set of stories into the Story Backlog if empty and append to end if not empty.

CS131-7 Exercise 2.2 Story: Print out data in Story List Nodes (required)
Nodes contain data about Stories, traverse the list and print out the data in the nodes.

CS131-7 Exercise 2.3 Story: Insert a Story into the Backlog (required)
Someone on the Team adds a single Story to the backlog.

CS131-7 Exercise 2.4 Story: Remove Story (required)
A Story is deleted from the Backlog from any Position. It could be removed using its name or number.

CS131-7 Exercise 2.5 Story: Find Story (required)

Find a story in the Story Backlog by story name or number. Print out its data if it exists in the backlog.

CS131-7 Exercise2.6 Story: Search Story contents within Backlog for keywords (advanced)
Find a Story with certain key words within the backlog using name or text fields.

CS131-7 Exercise 2.7 Story: Assign a priority to a Story in the Backlog (required)
Stories need to be prioritized for work within the Backlog. Assign using name or number. Priority can be 1-5. Allow zero for Unprioritized.

CS131-7 Exercise 2.8 Story: Assign a Story to a Team within the Backlog.
Stories can be reserved by a team to which it is assigned. Assign using team number 1-4. Allow zero as not assigned to a Team.

CS131-7 Exercise 2.9 Story: Order Stories by Priority and then Team within the Backlog. (required)
Stories could be ordered by overall priority within the Backlog

CS131-7 Exercise 2.10 Story: Order Stories by Team and then Priority within the Backlog.
Stories could be ordered by team within the Backlog

CS131-7 Exercise 2.11 Story: Find Stories of a given Priority (required)
Need to output all the stories with a given priority.

CS131-7 Exercise 2.12 Story: Find Stories of a given Team
Need to output all the stories with a given team.

CS131-7 Exercise 2.13 Story: Replace a Story by another Story within the Backlog (advanced)
Story has changed into another story.

CS131-7 Exercise 2.14 Story: Add or Remove First Story from Backlog (required)
Take a Story from the top of the Backlog. *First* tells you what the first story is *now*, then it adds or removes the existing one it if you agree.

CS131-7 Exercise 2.15 Story: Add or Remove the Last Story from the Backlog (required)
Take a Story from the bottom of the Backlog. *First* tells you what the *last* story is *now*, then it adds a new one or removes it if you agree.

CS131-7 Exercise 2.16 Story: Mark a Story Done (or not done) in the Backlog.
When you finish a story it may be marked done. Sometimes things are not finished and need to go back to being marked undone.

CS131-7 Exercise 2.17 Story: Done or Not Done?
Print out stories that are done and then those that are not yet done in the Backlog.

CS131-7 Exercise 2.18 Story: Change Data in a Story in the Backlog
Data that is in a story might need to be changed. Find it by name or number and then change the data in the Story fields including name and number. No two stories can have the same name or number.

CS131-7 Exercise 2.19 Story: Size of Story Board Backlog
How many stories are in the Story Board Backlog? Check stories against actual nodes present.

CS131-7 Exercise 2.20 Story: Team Done?
Has a team finished all their stories in the Backlog?

End of 'Stories' for Story Backlog

Prioritize the Stories in this Story Board Backlog. Do not implement them in the order given without deciding what should be done first with an eye toward your efficiency and effectiveness in performing the entire implementation. The Stories do not have to be done one at a time. The essential unitary structure of the transformations may be taken out and designed separately from the stories as they are presented in the Abstract or Concrete ADDs or Story List for Exercise 2.

If there are errors in any ADD that you are given, then you need to correct them in your code so that your code works properly. If you find significant errors in the given ADDs, then you should submit an error report.

Your program will be graded on how much of the functionality of the ADD you implement. Points will be subtracted for missing functionality. However, a sliding scale will be used. Pick out required functions to program first. Then add to that the other functionality. If a Story is marked advanced, then do it last.

The ADD is only meant to describe the algorithms at a high level of abstraction and the algorithm statements used are only examples which are meant to be simple, direct, and consistent so the transformations of the data structures by the algorithms are easy to understand. A lot of other considerations enter into writing code for an Algorithm. Be sure you think about the whole program. For example, you must free the memory of removed items.

Rationale

First, for Project 1 you were given a concrete ADD about Students on their Teams based on the use of Arrays of their Desks as they sat in Pair Programming Teams in class.

Then you were given an Abstract ADD about Singly Linked Lists and asked to make up your own Algorithm Sketches and to work on them in teams and present them to the entire class. If Exercise 2 goes well you will have an entire set of Concrete ADD sketches to work from. But there may not be time to complete all five Stories assigned to each team. But, in Exercise 2 you

were asked to make up your own Concrete Algorithm Sketches applying the Abstract Singly Linked List Algorithms to the case of Story Backlog List design. In this Project 2 you are asked to create a program that embodies the functionality of these Stories based on Singly Linked Lists and Iterators.

However, you may not have a complete set of Stories from other teams in Exercise 2 due to time constraints, and you may need to create your own Algorithm Sketches for the Stories that are in the list although you have not been given any Algorithm Sketches.

Sample input/output

The Concrete ADD may give some input and output examples. You need to make up your own inputs and outputs to test your program. Include enough test data to show that all the required functions work in the program and that the List mechanisms work. Your program should be written so that functionality can be selected with a menu, but one of the functions should also be to populate the lists with standard Dummy data for testing purposes. Another one of the functions should be a test script that runs to show that the core functionality of the code works properly. The input and output data from both your automated and manual tests should appear in your PDF file after your code. See the implementation of the Abstract Singly Linked List Algorithms in C++ produced by the instructor as a partial example of how code and testing data can appear in the same PDF file.

Program Design

A key part of this assignment is to learn to work with pseudocode and diagrams of Data Structures, in this case Lists and converting them into a C++ application. This application needs to be designed to take out the redundancy implied in the Stories and to consolidate the code to be as efficacious as possible in the accomplishment of the tasks that are described in the Stories. This means that the ADDs you are given are not to be directly implemented by following them by rote, but rather a program is to be designed globally that does what the Stories describe as efficiently and effectively as possible. Points will be deducted if the program too closely follows the given ADDs or Story list you have been given. Better designs of algorithms will also be rewarded. You are not required to use naming conventions or coding conventions from the given ADDs.

Code Reuse

Implementing the Story object as a Singly Linked List with its Story nodes accessed by iteration is the core work of this project. The Code you write needs to be your own solution to this problem, which is how to design the program based on the given algorithms that need to be emulated in your code. You may not use the Standard Template Library for this project. You can pick any kind of Sorting algorithm that you like to sort these lists. *Beware of following the given*

ADDs by rote as if it were code. You may use whatever C++ coding structures that you deem to be the best for implementing any part of the ADD as long as you are doing the implementation based on Linked Lists and use Iterators as appropriate. This assignment is meant to be coded from scratch by you and not copied from another source and then improved or changed.

Optional Challenge

The ADD may not be correct. Extra credit will be given for finding errors in the Abstract Singly Linked List document, or in the Algorithm Sketches of your own or those of your peers, especially if it is one that is not found by many others implementing from these ADDs.

You may have to use diagraming or other techniques such as those learned in class to figure out how your code is working. You may submit these diagrams with your work with a brief explanation for extra credit.

You may find functions that are missing in the ADD that would be either essential or useful which it does not describe. You may describe those in your own ADD sketches, but do not implement functions not included in the ADD without permission.

Code to write

Notice that there are some stories marked *required* in the Story Board Backlog and these determine the minimal functionality for a program to qualify for a C grade. There are also some stories marked advanced. These should be done last if you have time. However, stories marked advanced are expected to be part of the finished program. The reason for that is that this program is a whole system which needs all these functions to work as a whole and fullfill its necessary work keeping track of Stories for the Class. It may not be the minimal system for such a purpose and so these should be treated as the Stories that the Product Owner has requested.

1) Write an object for a Story List that implements the functions in the whole (or partial) ADD that comes out of Exercise 2 based on the Abstract Singly Linked List ADD produced by the Instructor. This list is a Singly Linked List that contains Story objects and is accessed by iterators.

2) A Story Object that contains the different kinds of data about Stories.

3) Code that relates the Story List object to its Story node objects so that they perform the transformational functions in the Stories. It is preferred that the Story Node be an object in its own right, but it may be just a **struct** instead.

Note: If you have a better way to organize the code than to use these objects, then explain this and how you have decided to organize your code in your implementation design.

4) A list of the stories that you have implemented and the reasons for not implementing those that you have not implemented from the Story Board Backlog of this Project. For instance, you could say that you could not implement the advanced stories because you ran out of time, and explain the reason.

5) Code that allows necessary textual inputs and outputs that are the needed text displays that can be printed to show conformance to the Stories.

6) Optional: Code that implements a more complex GUI that allows the various functions to be called in a WYSIWYG fashion rather than normal ASCII input and output.

7) Optional: Code that outputs the displays before and after pictures of data structures of input and output Lists in a visual and diagrammatic form to the screen that shows conformance to the ADD.

8) Optional: Connect these new objects to those done in Project 1. That means connect the Teams, Pairs, and Individual students to their assigned Stories. This would mean having Student, Team and Pair objects within their own data structures to implement that connection. For, instance you could have a Doubly Linked List with Student Object nodes within it. Or you could have a stack that implemented the Pairs within the team with two individual students that come to the board together and then sit back down. Or you could implement a double ended queue for the Stories assigned to a team to be placed within while they are being worked on by the team.

Your code should be appropriately commented. Your name, the Instructor's name, and the class CPSC131-7 should be included within your code.

Deliverables

Produce a written project report. Your report should include the following.

1. Your name and CWID and an indication that the submission is for project 2.

2. An input/output dialogue for putting new Stories into the system. You may want to have options to automatically load either the Stories about Team Desk arrays from Project 1 or the Stories about the Story Backlog Lists from Project 2 so that some testing can start with a full set of actual stories. A list of standard testing stories is necessary to show that your program works. It is also necessary to show that all the functions described in the ADDs are actualized in your program. . Both Manual Testing and a Test Script are expected to be used to show conformance.

3. An input/output dialogue using different input you choose. Different data about other Story sets can be used to test your program execution other than that which appears in any given ADD or any given Story List. Manual and Automated Testing with a script is expected.

4. Test scripts, programs, and printouts showing that the core functionality has been tested in an automated fashion for the program.

5. Your Source Code with appropriate comments including a program header that explains what the program does and gives important information like your name, class, section, CWID, and the instructors name, etc.

6. Optional: Reports of errors found in the ADDs functions with markups of the document showing where the errors occur.

7. Optional: Diagrams showing how your objects work using some diagraming technique discussed in class.

8. Optional: ADD-like sketches for the Stories that are not covered in the materials given you from Exercise 2.

Your document *must* be uploaded to Titanium as a single PDF file. If it turns out that a single file will not fit then it should be broken up in a reasonable fashion with related file names numbered 1of5, 2of5, 3of5 etc.

Your source code should be zipped and uploaded along with the PDF file so it may be run.

File names should have a form similar to this:

CS131-7YourFullName2015mmddProject2.pdf or zip

Or

CS131-7YourFullName2015mmddProject2_1ofx.pdf

CS131-7YourFullName2015mmddProject2_2ofx.pdf

Etc.

Be aware that Titanium has upload limits.

Due Date

This project was assigned Tuesday September 15, 2015.

The project deadline is Monday 10/5/2015, 1pm. Late submissions will be accepted but the worth of the assignment will be decreased based on how late the assignment is as described in the Syllabus. If your assignment is going to be late notify the instructor before the due date with an estimated time to completion. The assignment cannot be more than one week late. It is recommended that you turn in your assignment as it stands on the due date with a description of what is still to be done and then continue to work on it after that until the final acceptance date.

The latest that this project may be handed in overdue is 10/12/2015 for no more than 50% of the possible grade points.

CPSC131-7Project2AssignmentDescription20150914kdp10a.docx