CPSC 131-7 Fall 2015

Project 3: Classroom Student List with Doubly Linked List and Recursion (called Approach 2)

and

Project 4: Simulation of Student movement during Tag Team Pair Programming in Class using various Data Structures like Queues and Stacks (called Approach 3)

Introduction

In these projects you will program various data structures that are related to each other in order to simulate Tag Team Pair Programming in the Class. This Project has been split in half due to size and made into two projects. But there is one set of instructions for both. The concept is to use the Data Structures we are learning in order to simulate the Students in the Class on their teams and in their pairs working on Stories. It uses the modules already done in Project 1 and 2 and combines them with other Data Structures to simulate the Students moving through the Classroom and working in their teams as pairs who come to the board and solve problems based on Stories together.

There are two different Projects which are called Approaches here. One is to implement the Student Classroom List using doubly linked lists and recursion.. The other is to build a simulation of movement of students and stories that uses multiple data structures. The two projects fit together into an overall whole and so that is why they are described together.

The General Problem

You have been given the job of creating an infrastructure for implementing a simulation of Tag Team Pair Programming in the Classroom. You will use the data structures that you have learned when you create this simulation. The simulation will include the module of teams at their desks in pairs that was done in Project 1 by using Arrays and the module that organized the Story Board Backlog and by using the Singly Linked Lists that was done in Project 2. To this it will add the Set, Queue, Stack, Double Ended Queue. It will simulate the students coming into the room, taking their seats, forming teams, forming pairs, getting their assigned stories and then leaving the room. Simulation of the Actual work of solving problems as teams at the board and in the teams is Optional. But it may be that as part of your team you are assigned to implement the Classroom Student List completely. It could be that as part of your team you are assigned to implement the rest of the simulation except for the core functionality of the Classroom Student List needed for the simulation. It could be that as part of your team your partner and you were

assigned the two halves of this larger problem which need to be integrated together. It could be that you will implement the two halves of the assignment as two different projects.

Key Points[1]

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

> The programming problem is to create a simulation of the movement of students in the room during the Tag Team Pair Programming exercise that you have experienced in class. It uses the data structures we have learned in order to simulate this movement. It uses the Set to express the unordered state of the students outside of class. It uses the Queue to simulate the movement of the students into and out of class by the doors. It uses the Doubly Linked List to simulate the status of the students while they are in the class. It uses a large array (the size of the seats) in the class to simulate the occupation of seats within the class. It uses the Team Desk and Team List objects from Project 1 to simulate the Students on their teams. It uses the Stack to simulate the pairs that come to the board in order to solve problems and then return to their seats. It uses a double ended Queue to simulate the assignment of stories to be worked on by the team. The simulation contemplated here simulates the movement of the pairs to the board and then back to their seats within the team. In this Project we are simulating the movement of the students as it takes place in class as closely as we can using the data structures that we are learning. The actual connection from one data structure to the next is conveyed by rules that also appear in the ADD portion of this document. The complete program should simulate the movement of the students into the room, taking their seats (as part of their teams and pairs), and going to the board and returning to their seats. Then, at the end of class, it should simulate the students leaving the class by the two doors to the outside. In this simulation the students may be represented by their initials everywhere in the various data structures they visit. The students in class are represented in a doubly linked list. However, other information about the students exists in the double ended Queue that keeps track stories assigned to a team. Information in the various data structures connect to information in the other data structures so that the entire simulation will be coherent and demonstrate a unified and total simulation of the movement of the students and the assignment of stories to teams. Optionally, the actual work of the individuals, pairs, and teams may be simulated as well.

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 in the Story Backlog for this combined Project and the rules

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

for the interaction of the data structures. 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 Class Student List that will contain Student object nodes. It is expected that the movement of the students through the various data structures will be tracked by their initials. Data structures will be filled and emptied based on a generalization of what actually happens in class during the exercises you have experienced. 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. Thus, the user should be able to trigger different events such as the students coming into the room, being placed in the Class Student List, then placed in their seats, then given their seating position. They should be part of the teams represented by the Team Desk object and the Team List object from Project 1. Then, the teams will be assigned to pairs which are entered into a stack object. After that, the teams are assigned Stories from the Project 2 story list by placing them in a Double Ended Queue associated with the Team. The team will then work on these stories one by one until they are finished by the pairs going to the board in the order that they are in with their pairs in a stack. At this point the user should be able to trigger the *end* of class and the *exit* from the class via the door queues back into *two unordered* Sets outside the class. 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 to show that the various simulated movements have occurred. There should be a mode in which the entire simulation runs on its own. There should be a mode that tests the various data structures and how they work within the program via an automated test script. There should be the option to populate a given data structure with data to test it without putting in the data each time separately. The program should allow the user to do individual operations on the various data structures so that they can run their own simulation manually if they so desire. The program is expected to run without error and perform correct data transformations on the data structures described in the Stories.

However, if you cannot get your program to run without error, you should turn it in on the date due and then hand it in 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.

This is an advanced use of data structures that is more challenging than merely dealing with one data structure at a time. In actual programs in industry many different data structures are used together and data is shuffled around between data structures in order to produce correct outcomes for the application being implemented. Not all programs are based on databases where the data just sits in one place and is accessed by the program with results computed on the basis of the data from the data base and its changes over time. Some data needs to move in a dataflow fashion through different parts of the program in certain cases. Thus, this exercise gives some idea of how data structures can

be tied together and related to each other to accomplish a task, in this case, the movement of students in the class during the Tag Team Pair Programming Exercise which allows you to physically experience how the data flow in the program actually works by incorporating individuals, pairs, and teams. It is interesting that the data structures that you are learning can be used this way to emulate what you experience physically and mentally as you take part in this Tag Team Pair Programming exercise regime. One of the objects of these projects is to clarify the relationship between your own experience and its representation in programming information about situations in the real world. Many times programs are about things you have not experienced personally. In this case you have experienced this Agile way of doing exercises and you are programming from your own experience of those exercises.

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 **CS131-7_LastName-FirstName2015mmddProject3versionN.pdf or zip**. The source code should appear in both formats. The zipped ASCII 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. Or you may be asked to do a peer review by trading your program with a partner.

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

There is a section in this document that states 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. The main assignment is the implementation of as much of the functionality described in the Stories associated with Exercises 3 and 4 as possible, but condensed into a well-designed C++ program. If there are errors in your finished program that you cannot resolve, be sure to include error reports.

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

Project 3 will be assigned Tuesday, October 6, 2015 (10/6/2015).

There are three weekends between the assignment date and the due date for this assignment.

The project deadline is Monday 10/26/2015, 1pm.

- Make sure you hit the submit button because that is what records the time that your assignment is logged by the learning environment.

Project 4 may be assigned before this project is due. Project 4 will be Approach 3 in this Assignment.

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 or other SI tutors for help, especially for questions concerning C++ programming during their office hours.

There is other departmental tutoring available. Please inquire at the Computer Science office for other tutoring opportunities.

Algorithm Description Documents

You have been asked to create the Algorithm Sketches for Placing Students in a Doubly Linked List representing who is in the classroom at any given time. You have been asked to do this using Recursion and to take advantage of the symmetries of the Doubly Linked List. You have not been given either the Abstract or Concrete ADD for these stories but are expected to create these yourself as needed. Exercise 3 asked you to create these Algorithm Sketches for the Class Room List Stories using Recursion.

Story Board Backlog for Doubly Linked Lists of Classroom Students List

This set of Stories contains the Stories for the Exercise 3 that you have done individually, and which we have discussed in class.

Data Structures for Algorithms:

- The Student Objects are initialized with the student's information before being placed into any of these data structures.
- The Doubly Linked List of all the students in the room with the nodes being Student Object instances.

Stories:

Project 3 -- Exercise03: Classroom Student List

Create a Class Room List of Students using a Doubly Linked List and Recursion. Sort them in various ways using different Sorting Algorithms.

Data Structures for Algorithms:

- The Student Objects are initialized with the student's information before being placed into any of these data structures.
- The Doubly Linked List of all the students in the room with the nodes being Student object instances.

CS131-7 Exercise 3.1 Story: Create Student Instances of Student Objects for every student in the class (core required)
Produce one instance for each student in the Class of a Student Object containing Name, Initials, Classroom Seat, Team assignment, Pair assignment and Major then place in a Student Classroom List.

CS131-7 Exercise 3.2 Story: Fill Student Classroom List (core required)
Fill all students from Set into Doubly Linked List within the classroom.

CS131-7 Exercise 3.3 Story: Print out students in Student List (core required)
Allow print out of Name, Initials, Seat, Pair, Team and Major. Students introduce themselves.

CS131-7 Exercise 3.4 Story: Insert a Student into the Student List (core required)
Some Student comes in late.

CS131-7 Exercise 3.5 Story: Remove Student from the Student List (core required)
A Student has to leave the classroom due to an emergency.

CS131-7 Exercise 3.6 Story: Find Student in the Student List (core required)
We need to find out if a Student is present.

CS131-7 Exercise3.7 Story: Search contents within Student List for data on student.
Find a Student by First Name, Middle Initial if any, Last Name, Initials used in Class, Seat Assignment, Team or Major. (core required but optional for anyone doing the simulation, also required for those implementing all of the Student Classroom List)

CS131-7 Exercise 3.8 Story: Assign a Student to a Team within the Student List (core required)
Students may be assigned to teams.

CS131-7 Exercise 3.9 Story: Order Students by Team within the Student List (core required, but optional for those doing the entire simulation, also required for those implementing all of the doubly linked list)
We need to sort out the students by their teams within the classroom because they need to sit with their already assigned teams for the next exercise.

CS131-7 Exercise 3.10 Story: Find Students of a given Team in the Student List
Need to output all the students on a given team.

CS131-7 Exercise 3.11 Story: Replace a Student by another Student within the Student List.
One Student dropped the class and another Student took their place.

CS131-7 Exercise 3.12 Story: Add or Remove First Student from the Student List
Add a student either at the Front of the Student List or Remove a student from the Front of the Student List. Give the user a choice of which operation to perform.

CS131-7 Exercise 3.13 Story: Add or Remove the Last Student from the Student List
Add a student either at the Back of the Student List or Remove a student from the Back of the Student List. Give the user a choice of which operation to perform. (This is different for doubly linked lists as opposed to singly linked lists.)

CS131-7 Exercise 3.14 Story: Mark a Student as Absent
Did the Student sign the Attendance Sheet?

CS131-7 Exercise3.15 Story: Which students are Absent?
Which students were absent for the attendance?

CS131-7 Exercise 3.16 Story: Mark which Students are officially in Attendance but have physically left the classroom. (optional)
What if a student leaves the class for an emergency and is no longer there to be part of the exercise. He/She got an emergency call from home and had to leave.

CS131-7 Exercise3.17 Story: Which students are Physically in the classroom?
There is a new exercise and so we need to know which students are at their desks and ready to participate and which ones are missing.

CS131-7 Exercise 3.18 Story: Change Data in a Student instance record.
Student has changed his major, or perhaps her name because she was married, or there was an input error that needs to be corrected. (core required)

CS131-7 Exercise 3.19 Story: Size of the Student List
How many students are in the Student List regardless of whether they are in attendance or physically absent?

CS131-7 Exercise 3.20 Story: Reverse Doubly Linked Classroom List of Students
We need to reverse the list of Students.

CS131-7 Exercise 3.21 Story: Reorganize the Classroom List of Students based on a Sort.
We need to have the Doubly Linked List ordered the same as the Sort. (optional)

CS131-7 Exercise 3.22 Story: Order Students in Alphabetical order by Initials, by Last Name, by First name, or by Major (required)
We may need to access the data in different ways depending on what we need to do in class.

CS131-7 Exercise 3.22a Service: Randomize Student List (advanced)
CS131-7 Exercise 3.22b Service: Put Out an Array of Fields to be Sorted and Position in List (advanced)
CS131-7 Exercise 3.22c Service: Quick Sort Student List ((optional pick one)
CS131-7 Exercise 3.22d Service: Merge Sort Student List (optional pick one)
CS131-7 Exercise 3.22e Service: Shell Sort Student List (optional pick one)
CS131-7 Exercise 3.22f Service: Bubble Sort Student List (optional pick one)
CS131-7 Exercise 3.22g Service: Bucket Sort Student List (optional pick one)
CS131-7 Exercise 3.22h Service: Selection Sort Student List (optional pick one)
CS131-7 Exercise 3.22i Service: Insertion Sort Student List (optional pick one)

End of Students in Classroom Doubly Linked List

Project 4: -- Exercise 4: Story Board Backlog for Movement of Students in and out of the Classroom, and their Assignment to Teams and Pairs including the Assignment of Stories to each Team.

Data Structures for Algorithms:

- The Sets in the hallway before class and those in the hallway after class (having exited through the two doors) are made up of nodes of strings with the initials of the students.
- The Doubly Linked List of all the students in the room that has as its data: Student Initials, Student Name (First, Middle and Last) in separate fields, the Major of the Student, Team Assignment, whether present or absent.
- The Queues at the Doors are 5 students deep and contain (as their data) the initials of the students.
- The Stack of Pairs contains the initials of two students in strings, which is the output of the pairing algorithm from the Team Desks object. Each Pair is referred to by its team number and its pair number within the team within the Stack of Pairs for each team.
- The Students at the Board in Pairs are signified by the initials in Arrays of two strings each for each team depending on the number of teams in the configuration.

- The Seats in the classroom form a single big array of strings with some seats marked non-existent if the seat is missing to make the seating array irregular. Students are assigned to their seats using their initials. But, each seat may be empty or full at any given time, for instance, if the student has not sat down yet, or has gotten up for some reason, such as going to the board with their pair, or because they are an emissary, or because they have gone to talk to the instructor.
- Each Team is assigned Stories from the Story Board Backlog which exists in a Double Ended Queue. All Stories may be assigned to all teams at the beginning of the exercise period, or they may be assigned to the teams in sets. Assume that they are assigned randomly to the teams. But, they may be assigned to each team in the order that they appear in the Backlog of Stories.

Stories:

CS131-7 Exercise 4.1 Insert Students into a Set (core required)
Students are in an unordered Set in the Hallway or outside the Classroom before class, or after class when they have left the class by a doorway queue.

CS131-7 Exercise 4.2 Remove Student from Set (core required)
Students enter the Classroom by the door and file into the Classroom using a Queue which introduces linear order to the students as they enter the room.

CS131-7 Exercise 4.3 Insert Student into the back of a Queue at the Door. (core required)
Students need to form a Queue to get into the classroom after waiting in the hallway.

CS131-7 Exercise 4.4 Remove Student from the front of a Queue at the Door. (core required)
Students enter the class one at a time after waiting in the hallway. Or, they leave one at a time at the end of class when the Queues are reversed. There are two doors out of the classroom. All students must arrive by the hallway door, but they may leave by either the hallway door or the outside door.

CS131-7 Exercise 4.5 Insert Student from the Queue into the Doubly Linked List of Students in the Room. When they are in the Classroom they are inserted into the Doubly Linked List which represents their ability to move about the classroom and to take any seat in the classroom. (core required)

CS131-7 Exercise 4.6 Look to see who is next in the Doubly Linked List and assign them a seat in the room based on some criteria. As long as they are in the classroom they will remain in the Doubly Linked List of Students. (optional)
Assignment of Seats may be according to some attribute of the student such as their team, or where they were last sitting in another class period.

CS131-7 Exercise 4.7 Randomly place Students into Seats in Room.
As Students file in they sit where ever they wish. (core required)

CS131-7 Exercise 4.8 Place Students in Seats in Room by a criteria. (optional)
Students need to sit in Alphabetical Order by Last Name, or by Team, or by Major.

CS131-7 Exercise 4.9 Assign Students to a Team. (required)
Students need to join a team in order to do an exercise.

CS131-7 Exercise 4.10 Assign Students to a Pair within a Team. (required)
Pairs work together and Students are part of a pair.

CS131-7 Exercise 4.11 Treat pairs within a Team as a Stack and Assign Pairs to the Stack. (required)
Team members need to come up to the board in pairs when they are tagged. They must know what pair they are in and the order they are assigned when they to come to the board. However, they do not lose their seats when they come to the board. And, the stack is circular in the sense that it merely starts from the top again with the same pairs. Allow for any pair in the stack to be called to the board. Must be able to push and pop pairs of students to the Pair Stack.

CS131-7 Exercise 4.12 Assign pairs from Teams to positions at the board. (required)
First set of pairs are called to the board by being popped from the Team Pair Stacks unless they are called to the board out of the stack order.

CS131-7 Exercise 4.13 Teams at the Board tag the next pair in their team using the Stack Structure. (advanced, but required for those doing simulation)
Pair at the board is frustrated and wants to make the next pair suffer as they have. Pair at the board takes their seats and the next pair from the stack comes up. The stack is renewed from the Team Pairing when all the students in the Stack have come to the board.

CS131-7 Exercise 4.14 Fill Pairs into Pair Stack using Push for a Team. (core required)
Students join a team which may be horizontal or vertical within the desk setup. But, the order of those pairs that come up to the board when tagged is fixed by their place in the stack.

CS131-7 Exercise 4.15 Pop Pairs from Pair Stack. (core required)
When a pair is tagged by the other pair at the board they must come up to the board while the other pair sits down after explaining how far they got on the problem. But, teams may be asked to come to the board out of order by the Instructor.

CS131-7 Exercise 4.16 Students leave the Room by its two doors using Queues. (advanced, but required for those doing the simulation)
This requires two queues, one for each door. There is a transition through the Queues to a Set that is outside the classroom after they have left the class by either door.

CS131-7 Exercise 4.17 Set in hallway before class should equal two sets outside class after students leave unless someone left in the middle of class or entered after class started. Account for all Students before and after the class and those entering and leaving during class and make sure no student was lost in the process. (advanced)

CS131-7 Exercise 4.18 Place Pair at the Board. (required)
Pair comes to the board from their team.

CS131-7 Exercise 4.19 Remove Pair at the Board. (required)
Pair goes back to take their seats with the team.

CS131-7 Exercise 4.20 Identify which Seats are occupied at that moment by a Student and which ones are empty within the classroom at that moment despite being taken already. (optional)
Seats are occupied or unoccupied at any given time.

CS131-7 Exercise 4.21 Identify which Seats are not Occupied within the Array of Seats in the Classroom before and after Class, or, if a Student leaves class, their seat must not be taken by anyone else. (optional)

CS131-7 Exercise 4.22 Which Student has taken what Seat within the Classroom Desk Array? (required)
Each Student has a specific seat even if they are not in it at the moment.

CS131-7 Exercise 4.23 Which Stories have been assigned to each team? (optional)
Teams work on different stories in parallel together. The stories assigned to a team may be in a Double ended Queue for each Team.

CS131-7 Exercise 4.24 Set up Team Configuration for Exercise based on Seating in Classroom. (optional)

- There may be one, two, three, or four pairs from different teams at the board at the same time.
- There may be one, two, three and four teams.
- Teams may be one, two, three, four or all rows in the room.

- Make allowances for the fact that the row by the door has two seats. The configuration of the Room is 4 tiers of 7 rows plus 2 tiers of 1 row. But, make your algorithm so it is general and can handle any classroom seat array including ragged ones.
- Whiteboard space is divided according to how many teams have been assigned for the exercise.

End of Story Board for Movement of Students through Classroom.

Rules relating Data Structures in Class Movement Simulation

Rule 01: If a student is created then place in Set in the Hallway with initials of student in the Set.

Rule 02: If student in Set is in the Hallway, then queue up through back of queue to enter room randomly by Hallway door Queue, which has five slots with initials of student in the Queue.

Rule 03: If student is in Queue at Hallway Door, take students from front of Queue and place in Doubly Linked Classroom Student List as they file into the room.

Rule 04: If student is in Doubly Linked Queue Classroom Student List, then fill in data such as name, major, prior seat, assigned team, etc. This data may be filled in from storage but must be taken from an actual student list, such as the team lists for Exercise 2 or 3.

Rule 05: If students are all in Doubly Linked Class Student List, then allot students to Seats in the Array of seats in the room by placing initials in the seat they occupy. This may be done based on the team and position they last occupied, or randomly.

Rule 06: If students have been given seats, then determine their team based on how many teams are called for in the exercise (if they were not already in a team and if that team membership persisted).

Rule 07: If students are in teams and in seats, then determine pairings if not already existing.

Rule 08: If students are paired, then place pairs in stacks related to each team to manage the pairs coming to the board and sitting back down.

Rule 09: If students are stacked in pairs, then assign stories from Story List to double ended queue of the team that holds the stories that the team is working on, which provides that team with its own backlog of work to be done.

Rule 10: If a stack of pairs exists, then cause pairs to come to the board and cycle through the exercises at the board until all stories are done in their priority. Mark stories done when they are finished. (advanced)

Rule 11: If class has ended, then students leave randomly by either the hallway door or the outside door through reversed queues leading out. (advanced)

Rule 12: If students are in the room, then place them at the back of one of the exit queues randomly to leave the room in an orderly fashion. (advanced)

Rule 13: If students are in either one of the exit queues, then remove them from the front of the queue and place them in unordered sets outside the room either in the hallway or outside the building. (advanced)

End of rules for class movement simulation.

Exercise 5: Stories about Simulation of Work by Team: (all optional)

CS131-7 Exercise 5.01 Story: Team Starts a Story that is not finished. (optional)
The Team picks a story from its Story Queue that was assigned to it from the Story List. Team writes Start Time on the Board Log. This is an event which takes a random amount of time (under 2 minutes) and is logged in the Board Log.

CS131-7 Exercise 5.02 Story: Pair writes Diagrams that implements Story on the Board. (optional)
During the Exercise (while at the Board) the pair draws diagrams of data structure being used on the board. This is an event which takes a random amount of time (under 10 minutes) and is logged in the Board Log.

CS131-7 Exercise 5.03 Story: Pair writes Algorithm that implements Story on the Board. (optional)
During the Exercise (while at the Board) the pair writes an algorithm for using the data structure being used on the board. This is an event which takes a random amount of time (under 10 minutes) and is logged in the Board Log.

CS131-7 Exercise 5.04 Story: Team Finishes a Story. (optional)
Team writes Finish Time on the Board. This is an event which takes a random amount of time (under 2 minutes) and is logged in the Board Log.

CS131-7 Exercise 5.05 Story: Team finishes testing a Story. (optional)
Writes ‘Testing Done’ Time on the Board. This is an event which takes a random amount of time (under 10 minutes) and is logged in the Board Log.

CS131-7 Exercise 5.06 Story: Write Something to the Board. (optional)

Something is written on the Board. This is an event which takes a random amount of time (under 5 minutes) and is logged in the Board Log.

CS131-7 Exercise 5.07 Story: Erase Something to the Board. (optional)
Something is erased from the Board. This is an event which takes a random amount of time (under 2 minutes) and is logged in the Board Log.

CS131-7 Exercise 5.09 Story: Semi-Randomly generate Pair, Emissary, or Team Output to Board Log with Simulation Time and Initials of random Team Member who wrote it following roughly the same method for the steps given previously. This is an event which takes a random amount of time (under 5 minutes) and is logged in the Board Log.
(optional)

CS131-7 Exercise 5.10 Story: Assign Emissary from Team to Pair at the Board (optional)
Randomly pick a team member to be the Emissary for a given round and randomly decide if he goes to the Board to help the pair at the Board when in Simulation time during a given exercise on a particular story. The Emissary cannot be one of the pair at the board. If the Emissary's pair is picked then a new Emissary must be picked to transfer information from the Team to the Pair at the Board based on Sheets of Paper which carries the information from the Team to the Pair at the Board. This is an event which takes a random amount of time (under 10 minutes) and is logged in the Board Log.

CS131-7 Exercise 5.11 Story: Semi-Randomly generate Team Output to Sheet Log with simulation time and Initials of the random Team Member who wrote it following (roughly the method steps given previously). This is an event which takes a random amount of time (under 10 minutes) and is logged in the Board Log. (optional)

CS131-7 Exercise 5.12 Story: Team ends a Story.
Mark the Story done. This is an event which takes a random amount of time (under 2 minutes) and is logged in the Board Log. (optional)

End of Simulation Stories

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 Exercises.

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 your own ADDs, then you should submit an error report.

Your program will be graded on how much of the functionality of the Stories 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 the other functionality incrementally. 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 that 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. Functions that act on a data structure should pass on its state to the next function that accesses it in most cases. The Stories should not be treated as separate exercises as in mathemantics. The object is to design a whole program that works on its data structures in a synthetic way with a unified design.

Rationale

First, for Project 1 you were given a concrete ADD about Students on their Teams based on the use of the 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 Concrete Algorithm Sketches and to work on them in teams and present them to the entire class. From Exercise 2 you will have almost an entire set of Concrete ADD sketches to work from. Almost all the Concrete ADD stories were done by the class as a whole and the photos posted to the Class website. In Exercise 2 you were asked to make up your own Concrete Algorithm Sketches applying the Abstract Singly Linked List Algorithms to the Story Backlog List design. In Project 2 you were asked to create a program that embodies the functionality of these Stories based on Singly Linked Lists and Iterators.

In Exercise 3 you were asked to create a complete set of concrete ADDs for the Classroom Student List using Doubly Linked Lists and Recursion. This ADD forms a basis for Project 3. Another Story Board Backlog, Exercise 4, deals with the other kinds of Data Structures (Set, Queue, Stack, DEqueue, White Board positions of Pairs at the board, etc.) that are part of Project 4. In these two Projects we are simulating the movement of the Students in and out of class during the Tag Team Pair Programming exercise as well as the assignment of stories to the teams. Simulating the actual work of the Teams at the Board and on Sheets of paper at their seats as part of the team is Optional.

The whole of this combined project would be quite large (over 1000 lines of code) and so the work is broken down into three possible pieces two of which will be assigned as separate

Projects that should yield programs of the appropriate size on average. This large project was reworked and split on 20151004 due to the size of Project 2 when the data was analyzed.

Sample input/output

For Project 3 the input is a Class Room Roster. You can use the data from the team signup sheets produced in class for Exercises 1 or 2 as an example. You need to make up your own inputs and outputs to test your program. The ***test*** Student list does not need to be as large as the actual Student List for this class. Include just a few Students in your test data loaded by a special "Load dummy data" menu item. Include enough test data to show that all the required functions work in the program and that the List mechanisms work. However, the program you write should be able to handle real data, not just test data. Your program should be written so that functionality can be selected with a menu, although one of the functions should also be to populate the lists with standard Dummy data for testing purposes such as the Class Student List. 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. If you do the simulation one menu item should be the simulation script.

You should have appropriate displays after each function is performed in order to know that the code worked properly and that the data structures changed properly. You should have separate menu items to display the current contents of any data structure within your program.

Program Design

A key part of this assignment is to learn to work with pseudocode and diagrams of Data Structures as well as Object and Sequence Diagrams for program design, in this case the focus is on Double Linked Lists in Project 3, Stacks, Queues, DEqueues, Arrays in Project 4 which together made a whole simulation of the movement of Students and Stories as 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 created 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 you have been given. Better designs of algorithms will also be rewarded. You are not required to use naming conventions or coding conventions from your ADDs. The Rules specify one way in which the different data structures can be related to each other so as to simulate the movement of students entering class, teams getting assigned Stories, students going to their seats to work in teams, pairs of students going to the board, and then students exiting the class. These rules are given as a guideline. If there is a better way to attain the goal of implementing the intent of these rules then you may use your best judgement as to how to interlink these data structures in order to achieve the kind of flow that occurs in the

actual class performing the Tag Team Pair Programming exercise. Fidelity to what actually happens in class is a virtue in fulfilling this assignment.

Code Reuse

Implementing the Simulation of Movement of Students in and out of the class and moving from their seats to the board in Pair Programming along with the assignment of stories to teams, and Individuals to Pairs is the core work of this exercise. Simulating the actual work of the students at the board and in their seats is an optional embellishment. The Code you write needs to be your own solution to this problem, which is how to design the program based on the cited algorithms explained in the Textbook (or in the Lecture) that need to be emulated in your code. ***You may not use the Standard Template Library for this project except in the case of Approach 4.*** You may use whatever C++ coding structures that you deem to be the best for implementing any part of the Stories you represented in your given ADDs as long as you are doing the implementation based on Doubly Linked Lists with Recursion, Queues (either double or singly ended), and Stacks, and Sets. ***This assignment is meant to be coded from scratch by you and not copied from another source and then improved or changed.*** From scratch means you open an empty file and you start placing lines of code in it, until your program is complete not copying chunks or snippets of code from any other source. You must save versions of your code in order to show that it evolved and you used incremental development. You must save these versions and produce them if you are asked for them by the instructor. You are meant to reuse your own code from Exercises 1 and 2 as modules to be combined into the code that you will write in this project to make one program. However, you do not have to retest the code that was produced before unless it fails in some way during incorporation into the larger program.

If your previous code did not work or cannot be integrated for some reason you can stub out its functionality with test code. However, make clear in comments that these are testing stubs rather than your original code from Projects 1 and 2 that you have integrated into this project.

Optional Challenge

Turning the simulation of movement of the individuals, pairs, and teams within the class into a simulation of the work on problems represented by stories at the board and in their seats is an optional enhancement to the program that you may want to make so that you can create a complete simulation. This optional challenge is represented by the Work Simulation stories of Exercise 5 that are in their own separate section in this Assignment Description.

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 and not described in the Stories that would be either essential or useful. You may describe those in your own ADD sketches, but do not implement functions not included in the Stories without permission.

You may want to implement multiple options for sorting the students using different sorting algorithms in order to learn these.

\

Code to write

Approaches -- There are five ways to do this assignment:

1) ~~Do 'core required' only, but in an excellent way, as minimum for a C grade. This will implement 20 Stories. In this case all the functions must work on all the basic data structures that are marked 'core required' and there must be movement of this data between these data structures in a minimal way. The test script will place the various student initials in the various data structures in the order specified by the rules and then take them back out and move the students to the next data structure. In this case the test script will move the students from the Set to Door Queue, then to the Doubly Linked List Classroom Student List, then to the array of seats in the room, and back out of the room through the Queues into a Set outside the class. Pairs need to be pushed into and taken back out of the stack of Pairs. Stories need to be pushed into and taken out of either end of the Double Ended Queue. Iteration may be used for lists. Everything except the stories and the Classroom Student List may be in arrays. You must use at least three students between the different data structures that are part of the core requirements. You cannot get better than a C if you choose this approach unless you surpass your previous performance on Project 1 and 2 with excellence and craftsmanship that is explicitly demonstrated.~~ It turns out that this option would be too big and so it has been dropped as a possible approach.

2) Project 3: Decide to implement a doubly linked list of Students in the Classroom completely without the simulation part of the assignment. But, if you decide on this approach then you must use recursion as much as is appropriate instead of iteration. In this approach you can use a sort algorithm from a library.~~. Also, in this case you must implement your own Sort function from those that were given out as part of Exercise 3. It is preferred that the Sort algorithm should be the one assigned to your pair, although you may substitute another kind of Sort if the one assigned to your pair was difficult or not well understood by you. If you do a substitution, Quick Sort is preferred.~~ This approach encompasses 20 stories of which 8 are required using recursion. No other data structures other than the doubly linked list are necessary if this approach to the assignment is taken. Craftsmanship in programming will be rewarded. Taking chances and doing new things and demonstrating learning in relation to the past baseline of Project 1 and 2 will be rewarded. In this project the difference between the C level and the A/B level of performance is

the use of encapsulated Objects which may be friends rather than a structs, and the use of recursion rather than iteration.

3) Project 4: Decide to implement the Simulation with different objects, ~~and to only implement the part of the doubly linked list of Students in the Classroom that is necessary for the simulation to work~~. In that case you would implement multiple objects and make them work in relation to each other to make the simulation operate properly**.** The Test Script would show that the core functionality of each data structure works independently of the simulation. There would be a separate movement simulation script what would move the students' initials through the various data structures. Key test is that there are the same number of students in the Sets before and after the movement simulation. Ideally, you would move all the students into and out of the class through the various data structures as described in the rules. But you must at least move five students through the simulation. And you must at least assign four stories to the double ended Queue of Stories associated with each team. And you must at least cycle each pair to the board from the Stack of pairs of students. In this approach you do not have to use recursion. In this approach you can use a sort algorithm from a library. This encompasses 17 stories 11 of which are required from Exercise 4. Craftsmanship in programming will be rewarded. Taking chances and doing new things while incorporating skills learned in Project 1 and 2 will be rewarded. Some integration of the results of Projects 1 and 2 are also necessary.

4) ~~Do everything together as a single Project to the extent you decide that makes a whole program. This encompasses 46 stories (25 of which are required) simulating movement and giving a complete doubly linked list of Students which is sortable. *In this case you do not have to use recursion and you can use a sort algorithm from a library.* Craftsmanship in programming will be rewarded. Taking chances and doing new things while incorporating skills learned in Project 1 and 2 will be rewarded.~~

5) Optionally, do everything from the Stories of Exercise 3 and 4 plus the Optional simulation Stories of Exercise 5 which simulates not only movement, but work done by the individuals, pairs, and teams that make it a complete simulation of Tag Team Pair Programming Exercises. Craftsmanship in programming will be rewarded. Taking chances and doing new things while incorporating skills learned in Project 1 and 2 will be rewarded.

See separate table to see the estimated size of these project approaches and to compare their requirements. This table has been updated as of 20151014 due to the average size of Project 2 being over 800 lines. This document has also been reworked to make the two projects that it encompasses smaller in order to keep the lines of code required for these two projects down.

Guidance:

This project has been split in half to create two smaller projects.

I recommend attempting approaches 2 or 3 unless you are completely sure you cannot implement them so that you are able to get the highest grade possible for your work. Only consider choosing approach 1 if you received C or lower on both Projects 1 and 2.

If you got As and Bs on previous assignments you should choose either approach 2 or 3.

You really should only choose Approach 3 if you got As or solid Bs on Project 1 and 2. If you choose approach 3 then you will get the widest possible experience of data structures taught in the first half of this class. This will also involve the integration of previous Project code as modules into the code for this project. This approach also attempts to introduce the idea of a simulation to make the assignment more interesting. However, approach 3 may be larger than approach 2 even though there are the same number of stories, because there are more data structures to implement and relate to each other in approach 3.

It is recommended that most students choose Approach 2 unless they have a very good reason to choose another approach. Approach 2 is the conservative alternative between approaches 1 and 3 because it is like the singly linked lists which you have already done, it only has one data structure, but it needs recursion and the production of an implemented sort routine. The sort routine was added to try to make it more equal to Approach 3.

You should choose approach 4 only if you are extremely confident that you can do a project of that size which might be over 1000 lines of code. However, if you choose this approach you may use the Standard Template Library. This approach implements 46 stories.

Approach 5 is above and beyond the call of duty. No one is expected to choose this approach and it is only here for the sake of completeness. But it makes this not only a real simulation of both the movement of Stories and Students, but also the work done in pairs and teams.

Notice that there are some stories marked ***core required*** in the Story Board Backlog and these determine the minimal functionality for a program to qualify for a C grade.

- If you decide to attempt to get higher grade than a C grade (either A or B) then you can choose to implement all the doubly linked lists with recursion and a sort algorithm of your choice.
  Or…
- If you decide to attempt to get a higher grade than a C grade (either A or B) then you can choose to implement all the various data structures necessary to simulate student movement, and story movement within a simulation.

I offer these alternatives because a large number of students opted to do the minimal requirements for Project 1, which is a valid choice. This also makes the number of stories necessary to achieve that grade less (20 rather than 25) so that the resulting program would be

~~smaller. This is an attempt to fit the challenges to the capabilities of the students within the class. Students themselves decide which challenges are appropriate for them.~~

As in Project 2 some stories are marked required and so they are the minimal requirements for a C grade. Unmarked and Advanced stories are expected. But Advanced stories should be done last if you have time. Other stories are marked optional and these do not have to be done in order to get an A grade. 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 in order to fulfill its necessary work which is to keep track of Simulating the movement of the Students and Stories in Class during the Tag Team Pair Programming exercises. 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. **Objects:**

1) Write an object for a Class Student List that implements the functions of your ADD that comes out of Exercise 3 based on the Abstract Doubly Linked Lists with recursion. This list is a Doubly Linked List that contains Student objects and is accessed by recursive functions where appropriate. If you decide not to use recursion then make that known. Major Object for Approach 2 and Project 3.

> Create an input/output dialogue for putting new Students into the system. You must also have options to automatically load a set of test Students into the doubly linked list of Students in the Classroom. Both Manual Testing and a Test Script are expected to be used to show that this doubly linked list of Students works satisfactorily.

2) A Student Object that contains the different kinds of data about Students. These are the nodes of the Class Student List. Major Object for Approach 2 and Project 3.

3) Door Queues for students filing in and out of class at the beginning and end. There should be two instances of this object in the program. An Object for Project 4.

4) Array of Seats in the Room that Students can occupy within the classroom. The initials of the students can be put into this array of seats to show that they occupy that desk even if they are not in that seat at the moment. In other words their books or belongings or papers are in the seat but they have gotten up perhaps to go to the board with their partner in a pair. Some way to tell if the student is in his seat at the moment should be incorporated into the design. An Object for Project 4.

5) Teams that are sub-arrays of the seats in the room are already dealt with in Project1 that may be integrated into Project 4. The working code from Project 1 should be made a module and integrated into this program. This means that the design of the relationship between the teamDesks/teamNow arrays and the array of Desks in the Room needs to be designed and specified. There are many ways to do this. Pick a design that gives each a significant role.

6) Pairs that are determined within the teams and made up of two individuals who come to the board together are already dealt with in Project 1, and this should be integrated into your Project 4 code.

7) Stories objects that appear in a singly linked list have already been dealt with in Project 2. This code from Project 2 may be integrated as a module into the Project 3 code so that you can reuse it.

> Create an input and output dialogue for putting Stories into the System from Project 2 (as a Singly Linked List) so there will be stories to assign to teams within the system. This involves the integration of the code from Project 2 into the new Project 4. Some of the required functionality from Project 2 will be reused in this program. It is recommended that you include everything that worked from Project 2 as a module in Project 4.

8) Stack of Pairs that determine the order in which students come to the board when they are tagged by the pair currently at the board. An Object for Project 4.

9) Places for the two individuals currently at the board together for each team.10) Sets (unordered) outside the East or West doors into the Hallway or outside between the buildings. An Object for Project 4.

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

Note 1: The design of the whole system is important. Decide how many of the stories you will implement in your code and then make sure that works as a whole. Pick a reasonable subset if you decide not to do all the stories. Decide what kind of Design you are going to use to make the functionality work and be clear as to what this design is and how it works. Make sure that the code that implements your design is tested, and write appropriate test code, and test cases to show that your code works.

Note 2: Innovative approaches, creativity, and elegant ways of doing things within your code to make your system work will be noted and they will affect your grade. Demonstrated learning of new concepts and programming structures will be rewarded.

Note 3: It is possible to do this with no objects implementing the required functionality in a way that is flat, i.e. has no object hierarchy, and no instantiation. It is preferred that you use objects because that will make your implementation conceptually simpler and easier ultimately, but if you decide not to do that, then merely make your design decision clear. If you have not used objects in earlier Projects then this would be a good one to try to use them. Pick something out to implement in an Object Oriented way and show that you tried to make it work. The more object oriented the design and implementation are the better.

12) A menu item to allow the user/tester to load dummy test data into your program for testing starting from a known state. However, serial manual entry of data should also be available to the user separately.

13) A menu item that runs a test script that shows core functionality is working in the program.

14) Menu items to allow the contents of each data structure to be displayed at any time by the user/tester which is separate from normal displays.

15) After each function is performed based on menu inputs create a display that shows that the data structure was updated with the correct contents to show it is working.

16) Memory allocation and deallocation, object creation and destruction at appropriate places in your program.

17) 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.

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

19) Code that outputs the displays ASCII representations that shows that changes to data structures worked at the end of each menu selection.

20) The test status of each of your functions in the comments of the code.

21) If you are doing the movement simulation then include a Menu option that runs the simulation separate from the test script that shows that core of each data structure functionality works. The movement simulation will involve accessing each data structure repeatedly as students move from one to the other by the shifting of their initials or pointers to the student objects.

21) A UML object diagram and a sequence diagram for how the objects of this simulation work together. This is part of Exercise 4.

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

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

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 for CS131-7.

2. A description of your design for this project.

3. A list of what you decided to implement with a rationale for that selection of stories. That list must show which requirements you have met and which stories you have working which are tested. Untested or non-working stories must be indicated in some way.

4. 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.

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

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

7. The overall estimate of how much time it took you to do the production of the working code for the project, including the percentage of time spent in Design, Code and Test phases.

8. An error log which gives a description of the errors encountered that took more than 30 minutes to fix and how much time was spent working to fix that error.

8) Optional: A description of each test case you tested with the inputs for that test case, the expected outputs and the results.

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

10. Optional: Diagrams showing how your program works using some diagraming technique discussed in class.

11. Optional: ADD-like sketches for the Stories that are not covered in the materials given to you from Exercise 3. For instance, you might want to create ADD sketches related to the Queues, Stacks and other Data Structures not part of Exercises 3 or 4.

12. You should keep versions of your code to produce if asked by the instructor. Your grade can be severely affected if these versions do not exist when you are asked for them. Versions show that Incremental development was practiced by the Student.

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.

Both the PDF with code and tests and other elements that are not optional in this section and the Zip file with your source code and anything else relevant must be separately submitted together.

File names should have a form similar to this:

CS131-7_ LastName-FirstName2015mmddProject3versionN.pdf or zip

Or

CS131-7_ LastName-FirstName2015mmddProject3versionN _1ofx.pdf

CS131-7_ LastName-FirstName2015mmddProject3versionN _2ofx.pdf

Etc.

Be aware that Titanium has upload limits.

Due Date

This project is assigned Tuesday, October 6, 2015 (10/6/2015).

The project deadline is Monday October 26th before 1pm (10/26/2015). 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 for 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 if you did not finish it on time and then continue to work on it after that until the final acceptance date.

There is only one week between the due date and the final acceptance date. The latest that this project may be handed in overdue is Monday, November 2, 2015 (11/2/2015) at 1pm for no more than 50% of the possible grade points.

Even if your program is not finished please mail the current version on the final acceptance date to your instructor to let him know you have been working on it and the final result of your efforts

Criteria for Grading

Include but are not limited to:

- Submission instructions have been followed
- Stories that Work
- Inventiveness and Sophistication of Design
- Reasonableness of Layout
- Testing has been shown to have been done

- Testing has been sufficient
- Test results have been indicated
- Menus are reasonable and easy to understand and use
- The Simulation as a whole works where appropriate
- Dummy test data can be loaded
- There is a testing script that works and is a menu item
- There is a simulation function that works where appropriate
- Recursion is used where appropriate
- A separate Sort function is implemented where appropriate
- Whether input and output dialogs are reasonable
- Whether the user has been taken into account and the testing job made easy for the tester
- Object oriented design has been used
- Objects communicate with each other correctly
- Whether a list of stories done has been provided
- Whether there are annotations as to what menu items are tested and which are not tested.
- The program does not crash or end its execution other than through an Exit menu item.
- The program is easy to test for the tester
- The program works without having to be altered by the tester
- There are separate versions of the program to be presented on request by the instructor
- The student has improved over the baselines set by Projects 1 and 2
- The student has exhibited learning of data structure concepts that are relevant
- The student has exhibited better programming skills than on previous projects
- The student has exhibited a better program layout and design than on previous projects
- Whether the project was submitted on time and how late it was
- Whether there is a log of large errors when relevant
- Whether the testing status of functions are indicated in the code
- Whether things that don't work at all are commented out
- Whether things that are executable in the program are marked as having lingering errors that become obvious with instructor testing
- Whether they stated what approach they have chosen to follow to fulfill the assignment and stuck to it throughout implementation
- Whether there is a good correspondence between what the Stories ask for and what the Program does
- Whether the Program Design has been explained appropriately and in sufficient detail
- Whether the Variable and Object names are indicative of their functions within the program and the external objects they represent rather than cryptic or arbitrary