CPSC 131-7 Fall 2015

Project 5 Binary Search Trees

Introduction

In Project 5 you will program Binary Search Trees (BSTs) on the basis of the Doubly Linked Classroom Student Roster List of Project 3.

The General Problem

You have been given the job of creating fast ways to search the Class Roster using Binary Search trees. You will use the class roster file which you will *read in* to your program either from memory or from a file into the DLL, and then you will use that to produce several Binary Search Trees to access data within the Student Roster list, such as Initials, Last Name (LN) with First Initial (FI), First Name (FN) with Middle Initial (MI) and Last Initial (LI), etc. You will create your BSTs such that you can insert and delete nodes, search within it, and search the information in the DLL using it.

Key Points[1]

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

    The key problem is to produce BSTs that represent the information in the Student Roster DLL in order to allow fast searches of that information.

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 as long as it embodies the functionality of the Stories in the Story Backlog for this Project 5. 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 Roster Doubly Linked List (DLL) reused from Project 3 that contains Student object nodes. Data will be extracted from this DLL using arrays and BSTs will be built based on the information in the DLL for fast searching. BSTs are the focus of this assignment rather than the DLL. The stories of the assignment deal with the ways that a BST may be manipulated and used. The BST, like the DLL data structure, 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 to search the BST. The user

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

should be able to create a BST of a certain type, and then change it through insert and delete and other operations. The program is expected to run without error and perform correct data transformations on the BSTs associated with the Student Roster DLL data structure 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.

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-FirstName2015mmddProject5versionN.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 Exercise 8 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 4 was assigned Tuesday, November 17, 2015 (11/17/2015).

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

The project deadline is Monday 12/7/2015, 1pm.

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

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

There is no ADD for this project. But there are stories that summarize the operations on the Classroom Student Roster DLL already programed as well as the operations for this assignment on the Binary Search Trees. We leave in 'greyed out' the stories on the Student Roster DLL just as a reminder of Project 3.

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.

Project 3 – Already done -- Exercise03: Classroom Student Roster List Stories

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)

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)
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?
This 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

Assignment for this Project:

Project 5 Exercise 8 Binary Search Tree Stories

Exercise 8.1: Produce BST1 for Student FirstName (FN), MI Key by Insertion from class roster file or student roster DLL.

Exercise 8.2: Produce BST2 for Student LastName (LN), FI Key by Insertion from class roster file or student roster DLL.

Exercise 8.3: Produce BST3 for Student Initials Key by Insertion from class roster file or DLL. (required)

Exercise 8.4: Display BST with selected contents FN, MI **or** LN, FI, MI **or** Initials **or** node numbers on screen. (required)

Exercise 8.5: Display BST with node numbers and initials only on screen. (required)

Exercise 8.6: Use Object Oriented Design and instantiate separate trees from the same class.

Exercise 8.7: Insert new node into BSTs that exist already. (required)

Exercise 8.8: Find key using recursion in BST by printing out search path (node numbers traversed) and show the results of your search using PreOrder traversal.

Exercise 8.9: Find key using recursion in BST by printing out search path (node numbers traversed) and show the results of your search using InOrder traversal. (required)

Exercise 8.10: Find key using recursion in BST by printing out search path (node numbers traversed) and show the results of your search using PostOrder traversal.

Exercise 8.11: Delete from BST a node with no children. (required)

Exercise 8.12: Delete from BST a node with one child. (required)

Exercise 8.13: Delete from BST a node with two children. (required)

Exercise 8.14: As a test case produce a deletion scenario in which multiple chaining deletions occur and do those deletions.

Exercise 8.15: Handle chaining of deletion rules by using Stack to keep track of nodes in chaining deletions. (advanced)

Exercise 8.16: Determine the Balance of BST to see if it needs rebalancing by a chosen method. (advanced)

Exercise 8.17: Rebalance BSTs using AVL Tree, Red/Black Tree or Scapegoat Tree Methods. (advanced)

Exercise 8.18: Produce a test scenario with an Unbalanced BST in order to show how to balance it. (advanced)

Exercise 8.19: Traverse Initials BST using Preorder using Recursion and print node numbers and keys of nodes traversed as a result.

Exercise 8.20: Traverse Initials BST by using Inorder and by using Recursion and print node numbers and keys of nodes traversed as a result. (required)

Exercise 8.21: Traverse BST by using Postorder and by using Recursion and print node numbers and keys of nodes traversed as a result.

Exercise 8.22: Build an SLL of the result of traversals of BST. (optional)

Exercise 8.23: Determine Size of BST in number of nodes and print out result.

Exercise 8.24: Determine height of BST from any node with default case root node. (required)

Exercise 8.25: Determine depth of BST from any node with default case root node. (required)

Exercise 8.26: Determine predecessor of any node in BST order with default case node found.

Exercise 8.27: Determine successor of any node in BST order with default case node found.

Exercise 8.28: Read Class Roster data into DLL from students file as source for data. (optional)

Exercise 8.29: Read Student Classroom Roster data into DLL from constants within program. [Alternatively, read Class Student Classroom Roster data directly into BST, but this is not recommended. The alternative allows BSTs to exist without the Classroom Student Roster DLL.]

Exercise 8.30: Load Dummy data for 8-12 students for testing into DLL and build associated BSTs. [Alternatively, read Dummy Data for 8-10 students directly into BSTs, but this is not recommended. The alternative allows BSTs to exist without the Classroom Student Roster DLL.] (required)

Exercise 8.31: Produce a Test Script showing core functions work for Initials BST in Project 5 program. (advanced)

Exercise 8.32: Produce a performance profile for Time sorts verses BST searches and compare times for same data. (optional)

Exercise 8.33: Turn Pointers into node number positions in ‘trees as integers’ and then store as text file using Preorder traversal. (optional)

Exercise 8.34: From file of stored node number positions reconstitute BST with pointers that follow node number links. (optional)

Exercise 8.35: Put out an array with an order of Binary Tree as a basis for reordering DLL with student data. (optional)

Exercise 8.36: Reorder DLL with student data based on array from BST. (optional)

Exercise 8.37: Find key node using recursion by its number, or key contents within the BST. (required)

Exercise 8.38: Search for key or node number using recursion and update node contentsfound in BST. Assure tree key values are still in order. (required)

Exercise 8.39: Search BST for key and update node in Student Classroom Roster DLL. (optional)

Exercise 8.40: Delete node in BST based on Node number or Key contents.

Exercise 8.41: Insert a node in BST with new key contents at correct position to maintain order in BST.

Exercise 8.42: Print Tree Structure shape of BST with key values and node numbers. (optional)

Exercise 8.43: Produce a Random BST as dummy data for testing with random initials and random BST shape in terms of the connection of nodes. (optional)

Exercise 8.44: Have pointers or node numbers in BST for fast access to data in Student Roster DLL. Keep BST and Student Roster DLL coordinated. (optional)

Exercise 8.45: Load random BST dummy data into data structure for testing. (optional)

Exercise 8.46: Create a structure Treap ('Tree-Heap') instead of a BST and exercise it like the BST in this project. (optional)

Exercise 8.47: Create a 2-3 structure Tree instead of a BST and exercise it like the BST in this project. (optional)

Exercise 8.48: Create a Heap structure instead of a BSTand exercise it like the BST in this project. (optional)

Exercise 8.49: Create a Hash using FirstName, MiddleInitial, and LastName key *plus modulo* to retrieve Initials. (optional)

Exercise 8.50: Create a Priority Queue based on initials to retrieve FirstName, MI and LastName. (optional)

Exercise 8.51: Make sure that numbers in nodes correspond to the pointer structure of the nodes in a BST. (required)

Exercise 8.52: Produce Priority Queue of Majors to retrieve students of a given major. (optional)

Requirements Distribution

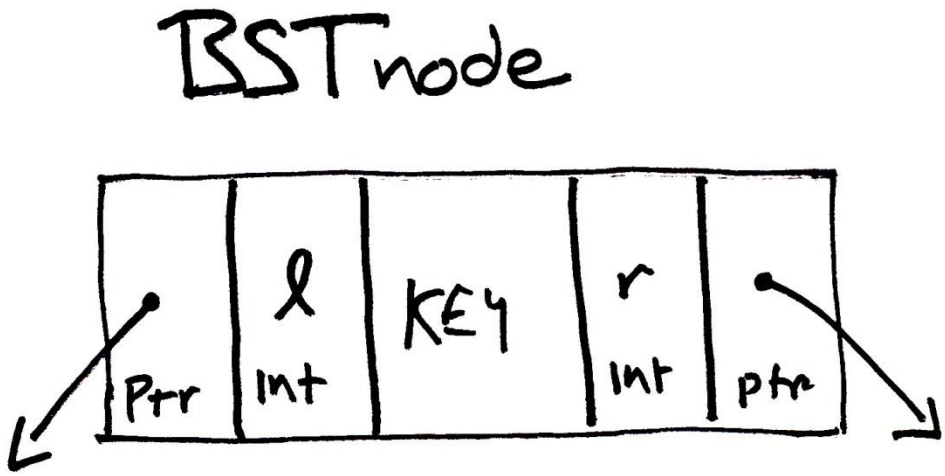
Normal: 13

Required: 15

Advanced: 5

Optional: 18

BST Node Structure



Node has integer number.

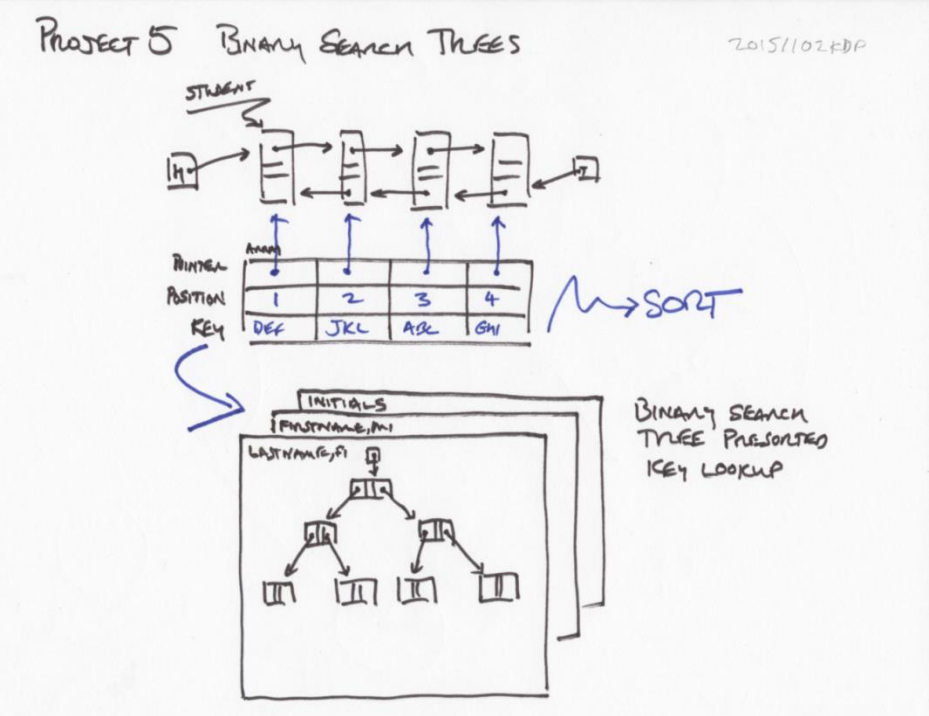
Node has key value.

Node has integer numbers of left and right nodes to which it is attached with zero if no child is attached

Node has pointers to left and right nodes

Node may have node numbers or pointers to nodes in Student Roster DLL for fast access to other data in DLL.

Note: Node numbers are for keeping track of nodes and for printing out structure of BST.



End of Story Board for Binary Search Trees Project 5

Prioritize the Stories in this BST 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 Story List for Exercises.

If there are errors in any Stories that you are given, then you need to correct them in your code so that your code works properly. Please make an error report of any errors you find in the Stories.

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 of the normal categorized requirements incrementally. If a Story is marked advanced, then do it last.

The Stories are only meant to describe the function of the program you will write at a high level of abstraction and the algorithm or design statements used are only examples that 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

mathematics. The object is to design a whole program that works on its data structures in a synthetic way with a unified design. Make sure that you always do the complementary operation that is necessary to complete or reverse any operation within the program as part of the whole design of your program.

Rationale

In Project 3 you programmed the Classroom Student Roster Doubly Linked List (DLL), and then it was used as a central data repository in Project 4. Here we will use it again as the basis for forming separate BSTs in order to provide a faster search of the information held in the DLL. The object of this project is to completely understand Binary Search Trees through their use in a program.

Sample input/output

For Project 3 the input was a Classroom Student Roster. You can use the data from the team signup sheets produced in the class exercise as an example. A datafile with this information has been provided for Project 4. In Project 5 the inputs are from the Classroom Roster File or the Classroom Student Roster DLL and are placed into various Binary Search Trees (BSTs) that will allow fast searching of different keys for the Students in class. You need to make up your own inputs and outputs to test your BST program. You should also provide dummy data for testing besides the entire data set from the File. You should program your application to handle any size of class. Here we are normally searching the whole class so we need to include all the Students in your test data. However, you may want to load a smaller number, say 8-10, to be loaded by a special "Load dummy data" menu item in order to do testing so that all the results can be seen on the display at one time. Include enough test data to show that all the required functions work in the program and that the DLL and BST mechanisms work. 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 provided 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.

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 Binary Search Trees. We will use the Double Linked Lists in Project 3 as a basis for this program to be integrated into our program on

BSTs. However, the BSTs need to be a separate C++ application which can be counted separately from the module that comes from Project 3. 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. Better designs of algorithms will also be rewarded. If there is a better way to attain the goal of implementing the intent of these stories, then you may use your best judgement as to how to interlink these data structures (DLL and BSTs) for the best possible result. Use what you know about the Tag Team Pair Programming exercises performed in class as a guide. Fidelity to what actually happens in class is a virtue in fulfilling this assignment.

Code Reuse

Implementing the Binary Search Trees that give access to the data in the Classroom Student Roster DLL data is the core work of this exercise. The code you will reuse will be your own Project 3 code for Doubly Linked Lists. The Code you write will be that which deals with Binary Search Trees using different keys to access Classroom Student Roster data which needs to be your own solution to this problem as described in Exercise 8.The design the program will be based on the cited algorithms explained in the Textbook (or in the class Lectures) that need to be emulated in your code. ***You may <u>not</u> use the Standard Template Library for this project.*** You may use whatever C++ coding structures that you deem to be the best for implementing any part of the Stories of Exercise 8 as long as you are doing the implementation based on Doubly Linked Lists with Recursion used in Project 3 and in this Project 4 by using Binary Search Trees and Recursion. ***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. Everything handed in for this assignment must be your own work. You must save versions of your code in order to show that it evolved and that you used incremental development. You must save these versions and produce them if you are asked for them by the instructor. It is a good idea to save these prior versions of your program to multiple storage devices so you do not lose your project through a hardware failure or some other catastrophe. Using Cloud Storage to keep the versions of your program as you work is a good idea. You are meant to reuse your own code from Project 3 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 3 that you have integrated into this project.

Optional Challenge

Within these Stories some are marked optional in order to give more advanced challenges to those who desire to further their understanding, but these are not necessary to get an A on the assignment.

Code to write

Approaches to writing the Code:

1) Already accomplished in Project 3: Decide to implement a doubly linked list of Students in the Classroom completely. You must use recursion as much as is appropriate instead of iteration. In this approach you can use a sort algorithm from a library.-This approach encompasses 20 stories of which 8 are required using recursion. No other data structures other than the doubly linked list are necessary.

2) Project 5: Implement the stories of Exercise 8 concerning BSTs. In this project we will take the information from the Class Roster File or the DLL produced from that File and produce various instantiated BSTs that allow that data to be searched quickly. We will do the maintenance operations on these BSTs which involve insertion, and 'change and deletion of nodes' from the BST. We will do rebalancing of those BSTs. Basically almost everything that can be done to a BST is part of the assignment. Optionally we allow the further creation and exercise of Treaps ('Tree-heaps'), Heaps and Hashes as well.

Craftsmanship in programming will be rewarded. Taking chances and doing new things and demonstrating learning in relation to the past baseline of Project 1, 2, 3 or 4 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 `struct`s, and the use of recursion rather than iteration. 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 the necessary work of the BSTs.

The most important thing about fulfilling the assignment is that you select a set of requirements that you can do in the time allotted and that you do them in a high quality way that is tested and works when you hand in the assignment. Explain your selection of the requirements to implement in your project and your design that meets those requirements.

Objects in the Code:

1) Doubly Linked List filled with Student Classroom Student Roster Data taken over and reused from Project 3.

2) Binary Search Trees which allow search of different keys from the Class Roster. Make objects that can be instantiated and use `private`, `public`, and `friend` container primitives as appropriate.

3) Nodes, either `structs` or objects, that carry the BST node data. Object nodes are preferred. But the `struct` may be part of an object. Those nodes should include key values, node number, left and right node numbers, and left and right pointers to other nodes. Left and right node numbers are the node numbers that the left and right pointers (if any) point to. Nodes can also optionally point at the nodes or give the node numbers of nodes in the DLL for quick access to the full information from the class roster. Using an object for the node will be rewarded even though it is not necessary.

Note 1: If you have a better way to organize the code to implement the functionality in the stories than to use these specified objects, then explain this and how you have decided to organize your code in your implementation design.

Note 2: 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 3: 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 4: 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 time 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 it will be. One of the goals of this class is for you to learn object oriented design and implementation in C++.

Elements of the Code:

1) 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.

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

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

4) 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.

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

6) 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.

7) 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.

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

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

10) 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 5 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 are working and have been 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 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 that 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.

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

10. Optional: Reports of errors found in the stories for the assignment.

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

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

13. 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-FirstName2015mmddProject5versionN.pdf or zip

Or

CS131-7_ LastName-FirstName2015mmddProject5versionN _1ofx.pdf

CS131-7_ LastName-FirstName2015mmddProject5versionN _2ofx.pdf

Etc.

Be aware that Titanium has upload limits.

Due Date:

This project is assigned Tuesday, November 17, 2015 (11/17/2015).

The project deadline is Monday December 7th at 1pm (12/7/2015)

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, December 14, 2015 (12/14/2015) at 1pm for no more than 50% of the possible grade points. For this project and all other projects in this course nothing will be accepted after December 14th, 2015 including restitution.

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 along with the final result of your efforts.

Restitution

There is no restitution for this project. Please test your program that you hand in adequately so that restitution is not necessary.

Previous projects were allowed restitution if they were not adequately tested and if required stories did not work. The necessity of testing has been a major component of this class. But this lesson should have been learned on previous projects in this class. Therefore, untested code which does not work will heavily count against your grade for this project.

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
- 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 ones 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

- \The Variable and Object names are indicative of their functions within the program and the external objects they represent, rather than being cryptic or arbitrary.