CSCE 221 REPORT
PA # 3

First Name: Mitesh Last Name: Patel UIN: 124002210

User Name: patel221881 E-mail address: patel221881@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more in the Aggie Honor System Office http://aggiehonor.tamu.edu/

Type of sources				
People	TA	HRBB Peer Teacher		
Web pages (provide URL)	Listed below			
Printed material				
Other Sources	Data structures and Algorithms text book	Course webpage slides		

Resources(URL):
http://www.cplusplus.com/doc/tutorial/files/
http://www.asciitable.com
http://stackoverflow.com/questions/1819189/what-range-of-values-can-integer-types-store-in-c

I certify that I have listed all the sources that I used to develop the solutions/code to the submitted work.

"*On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.*"

Your Name (signature) Mitesh Patel Date 10-13-15

1. Program Description:

   (a) The program description is to implement a doubly linked list, a doubly linked list template, and to analyze the complexity of this implementation. We are also suppose to use these lists to create a program where we can search through a phonebook and find the correct name by the user input.

2. Purpose of the Assignment

   (a) The purpose of the assignment is to learn about data structures, and algorithms, how to use these data strucutres, and how to implement these data structures to solve complex problems.

3. Data structure description

   (a) By doing this programming assignment I have succeded in learning about linked lists, and doubly linkest lists.

   (b) The implementation of the data structures came mostly from the slides, and with that came the use of object oriented programming where we used many different classes to organize our program. In order for the phonebook to work correctly we had to implement many different functions within our main.cpp. Our Doubly linked list implementation included a constructor that defined header and trailer, a copy constructor, overloaded = operator, a function to return the pointer to trailer and first nodes, and inserting and removing functions, along with our insertOrderly function from part 2 of the assignment. Our DlistNode class contained a constructor that holds the data, previous nodes address, and next nodes address, we also defiend three functions to get the next node, and get the previous node, along with to get the current element in the node.

4. Algorithm description

   (a) I used many algorithms to solve the phonebook problem, one was to insert the record into a correct position in the list, and this algorithm basically created a new node, got the first node, found where to place it usng the overloaded < operator in our record class, and then insert before the node so that it can be sorted still. Our overloaded < operator in the record class just checked to see if the record was less than the record being compared to, our overload << operator was just to display the record easily. In my main, I used a while loop to read the elements from the file, this involved reading line by line, and then I created a function to insert the record to the right index of the vector since it had to be in the right place. For this I used the ASCII table to help me, and basically I created a string to hold the record last name that was passed, and then a int to holdt the first letters ascii value , and subtracted that from the ascii and that would be the index. Then I used a vector to insert the record into the list in order using the insertOrderly. I also use to display the list in the right order, it involved a simple for loop iterating through each element and couting the vector at that index. The big part of this phonebook problem was the actual search through the list to find what the user input wanted. For this I basically got what the index should be by using the above algorithm described. I then created a temporary list , and a node to make the first element in the vector hold the first in the node. I used a while loop to iterate through our list created above, and if the nodes data at the current position was equal to the last name then we got the last name and put it into a record, and we inserted that record in the temporary list. Then we continued to check for duplicate last names. Finally we return the result if we found the last name. If not because there are more than one last names with the same last name then we repeat this process and check for first names, and if there are same first names, we repeat this process again and check UIN. And after checking, UIN if still not found then that means its not in the phonebook.

   (b) Analyze algorithms:

      i. 7 Functions implemented from Part 1 and their complexity: insert_before() - creates a new node, makes it to where the next portion of the newNode should hold pointer to self, and then newNode's previous should be previous, then the previous nodes next should ne newNode, and then finally previous node should be newNode. Time complexity O(1). insert_after() - same as insert_before, except you just switch the previous to next and the next to previous. Time complexity O(1). delete_after() - inialize a node to next node, then make next node to to the inialized nodes next portion, then make next's previous partition be self reference. Time complexity O(1). delete_before() - same as delete_after() except switch the nexts to previous. Time complexity O(1). Copy constructor - inalize the list, create a new node, get the first node in list, then using while loop iterate through list and insert new element and go to the next node. Time complexity O(n). Assignment operator - delete the whole list, create two nodes one ot hold previous and one to hold new. Let new node be the first node in the list, and then iteratre through the list and then set new element, and go to next element (copy constructor). Time complexity O(n). Output operator - create a temporary node and iterate through list and cout the elements and make it go to the next elem using getNext(). Time complexity O(n).

ii. The overloaded less than operator compares the records to see which one should be placed before. InsertOrderly function inserts an object at the correct position assuming the link list is sorted. After the object is sorted the link list should remain sorted. This function utilizes the less than operator desribed above. The time complexity for this algorithm would be O(n) since it is iterating through element by element to compare.

5. Program Organizaiton and Description of classes

   (a) Class definitions are already in a header .h file, and their implementation are in a cpp file for part 1 of the assignment, however for template doubly linked list, we can only have one header with the definitions inside that file. Therefore, our part 2 of the assignment has a main.cpp, a Record.h, and a TemplateDoublyLinkedList.h . Record.h only have one definition and thats the overloaded <<, so no need to make a seperate cpp file for that.

   (b) List all the classes or interfaces you used and show relation between them

      i. Main class - in the main we used node class, a DlistNode class, a doublylinkedlist class, Dlistnode and doublylinkedlist related to each other because we neded to create nodes. Node was used in singly linked lists.

      ii. Classes used to implement data structures - for simplydoublylinkedlist we used a struct Node to implement the doubly linked list. for doublylinkedlist we used a class Dlistnode, and a class doublylinkedlist. Also used a record class along with these for part 2 of the assignment . Record class and doubly linked list related to each other because we needed to create a vector of doubly linked lists that hold a data type of record.

      iii. Exception classes - used a try catch block interface for exception handling

6. Instructions to compile and run.

   (a) Compiling and running:

      i. Directory: SimplyDoublyLinkedList:

         A. make clean

         B. make

         C. Run using: ./SimpleDoublyLinkedList

      ii. Directory: DoublyLinkedList:

         A. make clean

         B. make

         C. Run using: ./Main

      iii. Directory: TemplateDoublyLinkedList

         A. make clean

         B. make

         C. Run using: ./TemplateMain

      iv. Directory: Book

         A. g++ -std=c++11 *cpp

         B. Run using ./a.out

7. Input and Output Specifications

(a) Make sure when you input the file you input what its asking for. If it asks for last name input the following for example : Andrews —— You can also enter: andrews —– The program automatically capitalizes the first letter of the last name. The same goes for first name as well. However, the program asks that you enter the first letter of the name to be capitalized just for best results.

(b) The format of the data from the file is in the following

    i. List [0] means all the names with last name beginning with A in order. List [1] means all the names with the last name beginning with B in order... etc.

(c) The program gets a segmentation fault, when the user inputs a number when it asks you for the first name or last name. For example: if you enter 1andrews then you will get a segmentation fault. This is fixed by the case below.

(d) Cases of wrong input that are involved in this phone book application are cought with exceptions. The only case considered wrong input would be if the user inputs a name and the first letter happens to be a number. It also catches if the user enters any symbol for the first letter.

8. Logical Exceptions

    (a) The program crashed when you had invalid input as mentioned above but fixed with exceptions.

    (b) The program catches exceptions mentioned above.

9. C++ object oriented or generic programming features

    (a) Inheritance - we didnt have any inheritence between classes in our program. We had a friend class, but thats as it close as it gets to inhertiance. Our friend class was in DlistNode

    (b) Polymorphism - In cases of polymorphism, we did in fact use overloaded operators such as << to cout a record, and a < to compare records. We didnt have any over ridden functions. We also used pointers to base classes ->.

    (c) Templates - used templates for doublylinkedlist in order for it to work with the phonebook assignment. We wanted to create our doublylinkedlist generic so we could succesfully complete the phonebook application. If it was not, then it would not have worked.

10. Tests

    (a) Valid Cases - The valid cases are when the user enters a last name correctly, or a first name correctly, or a UIN correctly. If the there are three people with the same last name, then it will ask for the first name. If they all have the same first name, then it will ask for the UIN. As long as those inputs by the user are valid, then the program will out put correct results.

    (b) Invalid Cases - Invalid cases are when the user does not enter a correct last name, or if they fail to enter the name correctly meaning they had a digit in front of the last name or a symbol. If the program cannot simply find the last name, then it would mean there is no one by that name in the phonebook which is considered a valid case since the program would have executed fully.

    (c) Random cases - There arent really any random cases that I can think of, unless the program ends up entering the phone number instead of UIN, but it would still return valid output because it will be searching integers the same way it would search UINs. If you do happen to enter the phone number instead of UIN, then it will 100% for sure not find anything with that information in the phonebook since the program is expecting a UIN not a phone number.