CMPT 225 Assignment 3: Trees

In this assignment you are to implement (in C++) a pointer-based binary search tree to store and retrieve customer data.

Start by downloading the assignment files. This zipfile contains a makefile, a test script and ground truths, and stubs for all of the .h and .cpp files you need.

- **Please do not create any additional .h or .cpp files.**
- **You can add to the .h files, but do not modify any of the provided method prototypes.**
- **You can modify customer_app.cpp for testing, but your code must work with the original customer_app.cpp.**

Customer Class

Customer.h, Customer.cpp

The customer class represents a customer of some fictional enterprise. The customer class should have the following attributes:

- Last name (*name*) - stored as a string (for which you will need to include the C++ string class: #include < string > using namespace std;))
- First initial (*initial*) - stored as a single character (i.e. type char)
- Account balance (*account*) - the amount currently owed by the customer

And these methods:

- Default constructor
- Constructor(string, char, int) where the parameters are the values for last name, first initial and balance
- Getters for each attribute
- Setter for the balance (not required for the other attributes)
- Overloaded operators for the comparison operators (<, <=, >, >=, ==, !=). This will allow you to compare two contacts in the same way that base type comparisons are made. The comparison should be made alphabetically, by last name, and then initial. For example Smith J is greater than Smith H, and Smith J is less than Smythe H. Note that the string class also overloads the comparison operators so you can compare two strings in this way.
- Overloaded operator for the cout << operator to allow you to output contacts. The output must be this:

```
Customer c("Doe", 'J', 3344);
cout << c;
```

should print:

```
Doe, J. (3344)
```

It is very important that your output looks exactly like this. The auto-grading script will require this output to function properly.

You can find some basic instruction on how to overload the comparison operators and the cout << operator in this lab activity.

Binary Search Tree

BSTree.h, BSTree.cpp

Implement a binary search tree that has the following methods, all methods should preserve the binary search tree property.

- Default constructor - creates an empty tree
- Copy constructor - creates a copy of the given tree
- Destructor - frees dynamic memory allocated by the tree
- bool insert(string, char, int) - creates and inserts a new customer (with the data shown in the parameters) in the tree, in a new tree node.
- bool remove(string, char) - deletes the first node with a matching name and initial from the tree. Returns true if the deletion was successful (that is, if the customer was found). Note that two customers are equal if they have the same name and initial, regardless of the account balance.
- bool search(string, char) - searches the tree for the given value, returning true if the customer is found
- vector<Customer> rangeSearch(string, char, string , char) - returns a vector of Customers where the customer names (initial and last name) are in the range specified in the parameters. For example rangeSearch("Dobbs", 'A', "Fogg", D) returns all customers whose names are between Dobbs A and Fogg D.
- void inOrderPrint() - prints the contents of the tree in sorted order. The output must be one customer per line, no other information. E.g. if the tree contains two customers, it must print:

```
Doe, J. (3344)
Doe, K. (2344)
```

I expect that you will need to also implement a number of helper methods (which should be private).

Node Class

Node.h, Node.cpp

As part of the tree implementation you should implement a Node class. Each node should contain a Customer object, pointers to left and right children and (optionally) the parent.

Application

customer_app.cpp (**complete**, no need to modify)

I have provided an application to maintain a collection of customers. It uses your binary search tree class as the collection's container. It allows users to do the following:

- Insert new customers
- Delete customers
- Search for customer
- Print the entire customer collection
- Print a range of customers
- Read a file of customers into the tree
- Quit

Note that this application will not function correctly until you have correctly implemented all the required methods in the Customer and BSTree classes.

Sample Files

Two sample files of customers are included in the zip file. I'd suggest using *small_names.txt* for your preliminary testing as it is small enough for you to work out exactly what is going on with it. You can use the larger one *big_names.txt* for further testing.

Automated Testing

*test_driver.cpp, test.py, *.gt*

A test driver *test_driver.cpp* is provided. It takes a command-line argument that specifies which test case to run. The test script *test.py* will run all of the test cases.

Please assert that the first two test cases pass before moving on to the others. The first case checks the Customer printing, and the second checks the inOrderPrint.

Manual Testing and Debugging

- You can run an individual test case in the test driver by supplying a command-line argument

  ```
  ./test_driver 4
  ```

  You could then compare this output to the ground truth file 4.gt.
- You can examine operations interactively using the application customer_app:

  ```
  ./customer_app
  ```

- You should make changes to all files to print out extra debug information (e.g. insert cout statements all over BSTree.cpp). **Make sure to turn these off before submission and automated testing.** The automated testing wants exactly the output in the .gt files, nothing else. One quick way to accomplish this is to define a constant, e.g. DEBUG, at the top of a BSTree.cpp, and only print the debug output if this is true.

  ```
  #define DEBUG 1
  ...
  ```

```
if (DEBUG) cout << "Current state of BST is " << bst.inOrderPrint();
```

Assessment and Submission

Submission

You should submit your assignment online to the CourSys submission server. You should submit the following:

- Modified *BSTree.cpp*
- Modified *Node.cpp*
- Modified *Customer.cpp*
- Modified *BSTree.h*
- Modified *Node.h*
- Modified *Customer.h*

Please read the documentation on the submission site for further information. The assignment is due at 11:59pm on July 31.

Assessment

The submitted files must build with the provided makefile and customer_app.cpp in order to receive marks. The assignment is worth 10% and marks are allocated to the assignment as follows:

- Correctness 6% (provided and held-out test cases)
- Memory management 1% (valgrind --error-exitcode=1 -q --leak-check=full ./test_driver 2)
- Coding style 3% (use of functions and loops, code indentation and spacing, comments and variable naming)
