HOMEWORK 8

100 points

DUE DATE: December 8th 11:59pm.

Warning: For any homework assignment that contains a programming segment please read the following very carefully.

Your code must compile and run on Black server. If your code does not work on Black server as submitted the grade for that problem is 0. Always test your code on Black, even if it is incomplete, make sure to get your code to compile and run on our Servers.

THIS HOMEWORK CONTAINS 2 PROBLEMS.

With your requirements document you are given:

1. BSTree.h
2. BSTree.cpp
3. HashTable.h
4. HashTable.cpp
5. main.cpp
6. make file
7. test10.cpp (Test your implementation of the hash table ADT, discover mistakes, correct them and execute your test plan again)
8. incomplete login.cpp (For Problem 2)

Problem 1: Hash Table ADT Problem

In this homework assignment you will implement the Hash Table ADT using an array of binary search trees representation.

Data Items:

The data items in a hash table are of generic type DataType. Each data item has a key of the generic type KeyType that uniquely identifies the data item. Data items usually include additional data. Type DataType must provide a function called *getKey* that returns a data item's key and a static method called *hash* that returns an unsigned int and receives a const reference to a KeyType as a parameter.

Data Structure:

This hash table ADT is an array of binary search trees. The placement of the data items in particular binary search tree is determined by the index calculated using the Data Type's static method named *hash*.
The placement within a particular binary search tree is determined by the chronological order in which the data items are inserted into the list-the earliest insertion takes places at the root of the binary search tree, the most recent as a leaf of the binary search tree. The ordering within a particular binary search tree is not a function of the data contained in the hash table data items. You interact with each binary tree by using the standard binary search tree operations.

Operations (Methods):

`HashTable (int initTableSize)` : Constructor, creates the empty hash table

`HashTable(const HashTable& other)` : Copy constructor. Initializes the hash table to be equivalent to the HashTable object parameter *other*.

`HashTable& operator= (const HashTable& other)` : Overloaded assignment operator. Sets the hash table to be equivalent to the `other` HashTable object parameter and returns a reference to this object.

`~HashTable()` : Destructor. Deallocates (frees) the memory used to store a hash table.

`void insert(const DataType& newDataItem)` : Inserts `newDataItem` into the appropriate binary search tree. If a data item with the same key as `newDataItem` already exists in the binary search tree, then updates that data item with the `newDataItem`. Otherwise, it inserts it in the binary search tree.

`bool remove (const KeyType& deleteKey)` : Searches the hash table for the data item with the key `deleteKey`. If the data item is found, then removes the data item and returns `true`. Otherwise returns `false`.

`bool retrieve( const KeyType& searchKey, DataType& returnItem) const` : Searches the hash table for the data item with key `searchKey`. If the data item is found, then copies the data item to `returnItem` and returns `true`. Otherwise, returns `false` with `returnItem` undefined.

`void clear ()` : Removes all data items in the hash table.

`Bool isEmpty() const`: Returns `true` if the hash table is empty. Otherwise, returns `false`.

`void showStructure() const` : Outputs the data items in the hash table. If the hash table is empty, outputs "Empty hash table". Note that this operation is intended for testing/debugging purposes only. It only supports data items with key values that are one of C++'s predefined data types (int, char, and so forth) or other data structures that have overridden `ostream operator<<.`

Implementation Notes:

You can implement a hash table in many ways. We have chosen to implement the hash table using chaining to resolve collisions. The binary search tree ADT provides a simple way of dealing with a chain of data items and is an opportunity to use one of your ADTs to implement another ADT.
Step 1: Implement the operations in the Hash Table ADT using an array of binary search trees to store the hash table data items. You need to store the number of hash table slots(`tableSize`) and the actual hash table itself(`dataTable`).
You will be working on HashTable.cpp. Base your implementation on the file HashTable.h. (`BSTree.h,` and `BSTree.cpp` are already completed for you).

Please note that HashTable.h contains a private field named as dataTable
`BSTree<DataType, KeyType>* dataTable;`
dataTable is an array (C array, not a pointer).

Save your implementation of the Hash Table ADT in the file HashTable.cpp. Be sure to document your code as you did on previous homework assignments.

Your main.cpp reads account numbers and balances for a set of accounts. It then tries retrieving records using the account numbers as the keys.
When using main.cpp simply enter 5 account id and balance pairs for example:

```
Enter account information (acct_num balance) for 5 accounts:
6274 415.56
2843 9217.23
4892 51462.56
8837 27.25
1892 918.26

Enter account number (<EOF> to end): 4892
4892 51462.6
```

if you enter the same account number again, it updates the info:

```
Enter account information (acct_num balance) for 5 accounts:
6274 415.56
2843 9217.23
4892 51462.56
8837 27.26
2843 1476.50
update 2843

Enter account number (<EOF> to end): 2843
2843 1476.5
```

We have also provided **test10.cpp** (another driver). This driver is for you to test your own code, it helps you discover your implementation mistakes and correct them. See below on how your tester works, after entering few value. Output below only shows the testing for insert… You can test your other functions as shown in the menu below.

```
Commands:
  H   : Help (displays this message)
  +x  : Insert (or update) data item with key x
  -x  : Remove the data element with the key x
  ?x  : Retrieve the data element with the key x
  E   : Empty table?
  C   : Clear the table
  Q   : Quit the test program
0:
1:
2:
3:
4:
5:
6:

Command: +6274
Inserted data item with key (6274) and value (1)
0:
1: 6274
2:
3:
4:
5:
6:

Command: +2843
Inserted data item with key (2843) and value (2)
```

```
Command: +1892
Inserted data item with key (1892) and value (5)
0:
1: 6274 8837
2: 1892
3:
4:
5: 4892
6: 2843

Command: +9523
Inserted data item with key (9523) and value (6)
0:
1: 6274 8837 9523
2: 1892
3:
4:
5: 4892
6: 2843

Command:
```

You can still use the same make file, simply when you are ready rename your test10.cpp as main.cpp (make sure to place the main.cpp we gave you to some other directory so you don't have 2 drivers under one folder.)

Problem 2: Hash Table ADT Exercise

One possible use for a hash table is to store computer user login usernames and passwords. Your program should load user name and password sets from the file *password.dat* and insert them into the hash table until end of file is reached on password.dat.
There is one user name/password set (separated by a tab) per line as shown in following example (password.dat)

```
jack    broken.crown
jill    tumblin'down
mary    contrary
bopeep  sheep!lost
cole    merry-soul
simon   no!pieman
```

Your program should present a login prompt, read one user name, present a password prompt, read the password, and then print either "Authentication successful" or "Authentication failure" as shown in sample run below:

```
0:
1: jack mary
2:
3: bopeep cole jill
4:
5:
6: simon
7:
Login: jack
Password: broken.crown
Authentication successful
Login: bopeep
Password: tumblingdown
Authentication failure
Login: jill
Password: sheep!lost
Authentication failure
Login:
```

The authentication loop is to be repeated until the end of input data – EOF- is reached on the console input stream (`cin`).
An incomplete login.cpp is provided for you. Work on your program to ensure that it will read in the user names and passwords from *password.dat* and then allow the user to try authenticating usernames and passwords as shown as long as the user enters more data.

Again, you can still use the same make file, simply when you are ready rename your login.cpp as main.cpp (make sure to place the old main.cpp to some other directory so you don't have 2 drivers under one folder.)

Homework 8 Deliverables:

The following files must be submitted via Handin no later than December 8th 2016 by 11:59pm

1. **HashTable.cpp**
2. **Login.cpp**

DOCUMENTING YOUR CODE WITH PRE AND POST CONDITIONS
BOTH DOCUMENTATION STYLES ARE ACCEPTABLE.PICK ONE!

EXAMPLE 1 Documentation is done as a header to a function.

```
109     /**
110      * @pre   : Heap must remain as max heap
111      * @param : x
112      * @post  :Adds an item to a max heap
113      * @return: None
114      */
115     void insert(const T &x) {...18 lines };
133
134
```

EXAMPLE 2: Documentation is done during listing of public methods in a class.

```
30      void Dequeue(ItemType& item);
31      // Function: Removes front item from the queue and returns it in item.
32      // Post: If (queue is empty) EmptyQueue exception is thrown
33      //       and item is undefined
34      //       else front element has been removed from queue and
35      //       item is a copy of removed element.
```