Programming Assignment #5

This assignment gives you a chance to test your knowledge of hash tables. Specifically, you will develop an open-addressing-based hash table class that uses both *linear probing* (with ascending indexes) and *double hashing* to resolve collisions. You will also be able to implement the two kinds of deletion policies that I demonstrated in class. The class will be templated and is named OAHashTable (for Open-Addressing Hash Table). It has a simple public interface as follows:

Method	Description
`OAHashTable(const OAHTConfig& Config)`	Constructor. *Config* - The configuration for the hash table. The OAHTConfig struct has these members: *InitialTableSize* - The number of slots in the table initially. *PrimaryHashFunc* - The hash function used in all cases. *SecondaryHashFunc* - The hash function used in double hashing. If it is 0, then linear probing will be used. *MaxLoadFactor* - The maximum "fullness" of the table. *GrowthFactor* - The factor by which the table grows. *DeletionPolicy* - How to deal with deletions (marking or packing) *FreeProc* - The method provided by the client that may need to be called when data in the table is removed.
`~OAHashTable();`	Destructor.
`void insert(const char *Key, const T& Data)`	Inserts a Key/Data pair into the table. Throws an exception if the data cannot be inserted. (E_DUPLICATE, E_NO_MEMORY)
`const T& find(const char *Key) const`	Finds the data by key and returns a reference to the constant data. Throws an exception if Key isn't found. (E_ITEM_NOT_FOUND)
`void remove(const char *Key)`	Removes a Key/Data pair by key. Throws an exception if the pair cannot be removed. (E_ITEM_NOT_FOUND)
`void clear();`	Removes all Key/Data pairs in the table, freeing the data if necessary. This does not free the hash table itself. If a FreeProc was provided, you must pass the data in each slot to that function and then set the slot as unoccupied.
`OAHTStats GetStats() const;`	Returns a struct that contains information on the status of the table for debugging and testing. The struct is defined in the header file.

Notes:

1) When an insertion will cause the maximum load factor to be *surpassed*, you must grow the table *before* inserting. A load factor of 1.0 means that you will grow the table only when an item is to be inserted into a full table.
2) The deletion policy affects how deleted items are handled. There are two possibilities: (**MARK**) Mark the slot as being in a deleted state; (**PACK**) After deleting an item in the table, compact the table by moving key/data pairs in the affected cluster using the algorithm demonstrated in class. Look at the output from the sample drivers for examples. Note that you can't reinsert clusters if you are using double-hashing. (You won't be asked to use double-hashing and packing.)
3) Don't forget that with double-hashing, the secondary hash function may return 0. The way you will guarantee a value between 0 and Tablesize - 1 is to call the secondary hash function with Tablesize – 1 and add 1 to the return value.
4) If the client provides a callback function for freeing data, you need to call this function and pass the data upon marking the slot available. The client may pass 0 (NULL), meaning that there is nothing to free. PLEASE CHECK THIS VALUE AND DON'T BLINDLY CALL IT.
5) There is a public method that exposes the internal state of the table (e.g. number of probes, expansions, etc.) This facilitates testing and debugging at the client level.
6) It is important that you track the number of probes (searches) and expansions (growing the table) throughout the lifetime of the table correctly. You should strive to match the output of the sample driver. Be sure to count **every** time you search for an item in the table. This includes inserts, deletes, and searches (find). Also, when you have a collision, the subsequent probes will move forward in the table (ascending indexes). Anytime you look for an item after hashing its key, this is a probe. The sample driver demonstrates this in detail. PLEASE LOOK AT THE DRIVER AS AN EXAMPLE.

Growing the Table
When the maximum *load factor* will be surpassed, you must expand the table. This means you need to allocate a new table and re-insert all of the existing key/data pairs from the old table into the new one as demonstrated in class. Since the table size has changed, each key will hash to a new index. When expanding the table, you will use the *GrowthFactor* that was supplied to the constructor. To keep the size of the hash table a prime number, use the included *GetClosestPrime* function to calculate the new table size. So in the *GrowTable* method, you would have code similar to this:

```
    double factor = std::ceil(TableSize_ * Config_.GrowthFactor_);  // Need to include <cmath>
unsigned new_size = GetClosestPrime(static_cast<unsigned>(factor)); // Get new prime size
```

Testing
As always, testing represents the largest portion of work and insufficient testing is a big reason why a program receives a poor grade. (My driver programs take longer to create than the implementation file itself.) A sample driver program for this assignment is available. You should use the driver program as an example and create additional code to thoroughly test all functionality with a variety of cases. (Don't forget stress testing.) There are actually two sample drivers posted for this assignment. Please test with both of them.

Remember that, due to the class being templated, you will include the implementation file at the bottom of the header as we have done in the past:

```
#include "OAHashTable.cpp"
```

What to submit
You must submit your header and implementation files (OAHashTable.h, OAHashTable.cpp) via the appropriate submission page.

Source Files	Description
OAHashTable.h	The header files. **No** implementation code allowed (except for the code already included.) The public interface must be exactly as described above.
OAHashTable.cpp	The implementation file. All implementation for OAHashTable code goes here. You must document the functions in detail with Doxygen tags in these files.

Make sure your name is on all documents.