Project 5: HashTable
Due: - 5/6/2009

Objective
The main objective of this project is for the student to gain additional experience working with the HashTables.

Problem
The C++ STL (Standard Template Library) does not support hash-based containers. However, most compilers like GCC or Visual C++ have their own implementations. Here's the problem: what happens if you want to create cross-platform software that uses a hashtable? Whenever there are platform dependent features that have been added to software then porting the program to another platform will require "recoding" all the dependent sections.

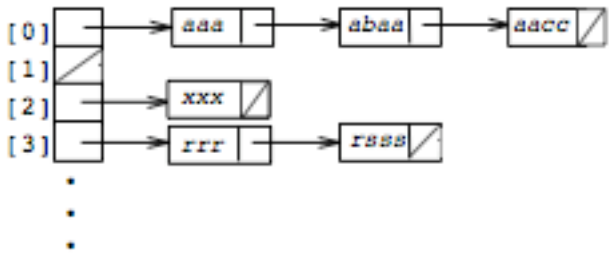
In this project, you will code your very own "independent" hashtable code that uses the "chaining" method, which was discussed in your textbook and in class.

Let's pretend that your software project boss has given you the assignment to come up the code to implement the hashtable in order to keep the company's software as platform independent as possible.

In this program, you will be using an array (or vector) of linked lists to implement a hash table. To process the linked lists, you will use the List STL class.

Background
A *hash table* is a data structure in which the location of an item is determined directly as a function of the item rather than by a sequence of trial-and-error comparisons and is used when fast searching is needed. Ideally, search time is O(1); i.e., it is constant — independent of the number of items to be searched. Here we will use the method known as *chaining*, in which the hash table is implemented using an array (or vector) of linked lists.

When an item is to be inserted into the table, a *hashing function h* is applied to the item to determine where it is to be placed in the table; for example, a common one is *h*(*item*) = *item* % *table-size* where the table size is usually taken to be a prime number in order to scatter ("hash") the items throughout the table. For non-numeric items, numeric codes— e.g., ASCII — are used. The linked list at location h(*item*) is searched for the item. If it is not found, the item is inserted into the linked list at this location.



Requirements

1. Write a hashtable class that uses chaining to handle collision.
2. Test the resulting implementation thoroughly.
3. Compare your code to the GCC – "hash_set" class and the STL set class.
    a. Do some basic "timing" studies comparing your class to each of the other classes – your class should be better than set and close to hash_set.
    b. Create a short report explaining your testing and results
    (Note: For this project, the testing and report are extremely important, because both help prove that your hashtable class will do the job.)
4. Add your class implementation to wordQuery.cpp to give it fast searching capabilities.

Implementation Details

The folder "/gc/cs212/project5" on the moses file server contains an example of GCC "hash_set". Please refer to this program as you are write and test your hash class. Use a similar test to verify that your hash class' performance is comparable to the performance of GCC's hash_set.

Create three files: hashT.cpp, hashT.h, and wordQuery.cpp

hashT.cpp and hashT.h – hashtable class (should have similar capabilities as hash_set).

wordQuery.cpp – this program reads in lines from a textfile, parses out the words, and stores the words in your hashtable. Next, the program enters a loop – asking the user for a word and then reporting back either "present" or "not present" in document.

Turn-in (into a Dropbox folder)

1. Code files: hashT.cpp, hashT.h, and wordQuery.cpp.
2. Report: explaining your design and "proof" that it is close to being as efficient as map_set.