CSC 230 – Project #4 (optional project)

Due: May. 1st, before midnight

Project Details:

In this assignment, you will play with an important data structure called *hash table*. In project 1, we have used with vectors. In project 2, we used dynamic arrays. In project 3, we used linked list. This project requires you implement hash table to fulfill the functionality provided by project 2 and project 3. You will first implement a *singlely linked list* class SLL in SLL.h file, and then implement HashTable class that is in hashTable.h file. Last, implement project4.cpp file. Please read the handout and Zybook before coding. A node.h file is provided. Do NOT change node.h file. The end of this handout has important guidelines to be used in testing.

Getting help

If you don't know where to start, if you don't understand testing, if you are lost, etc., please SEE SOMEONE IMMEDIATELY —an instructor, a tutor. Do not wait. A little in-person help can do the magic.

Your Implementation

This project includes three parts.

1. Implement the unfinished methods of SLL class. Modify the methods in SLL.cpp, do NOT change anything in node.h file. SLL must be class **TEMPLATE**. Use placeholder wherever you do not want to use fixed data type.
2. Implement HashTable class, which is class template. The default constructor uses 3 as the table size. The constructor with parameter uses the parameter value as the table size. Do not hard-code array size in your constructors. Let the user decide what array size will be.
3. Implement project4.cpp file, which creates a HashTable object with parameter 10007 for the constructor. The constructor of HashTable uses this prime number to build an internal SLL object array. Each element of the array is an SLL object. Of course, you can choose a different array size, but make sure the size is a prime number.
4. There are multiple input files are included in the jar file. Please note that we have new files whose data set sizes are larger than the files of the previous projects.

To compile the source code, you can type the following command:

g++ project4.cpp –o project4

The executable file will be *project4*. If your hash table implementation is right, when we execute file *project4*, the result looks like the following contents:

```
jli$ ./project4 40000-idr
The Number of Valid Insertation :26214
The Number of Valid Deletion :2795
The Number of Valid Retrieval :2867
Item numbers in the list :23419
Time elaspsed :0.102787

jli$ ./project4 60000-idr
The Number of Valid Insertation :39228
The Number of Valid Deletion :4262
The Number of Valid Retrieval :4386
Item numbers in the list :34966
Time elaspsed :0.150879

jli$ ./project4 250000-idr
The Number of Valid Insertation :156536
The Number of Valid Deletion :28338
The Number of Valid Retrieval :18613
Item numbers in the list :128198
Time elaspsed :0.692797

jli$ ./project4 500000-idr
The Number of Valid Insertation :312656
The Number of Valid Deletion :56010
The Number of Valid Retrieval :37360
Item numbers in the list :256646
Time elaspsed :1.88009
```

If you compare the above results with those of project 3, the Time values are significantly smaller than those of project3. This is the beauty of Hash Table. If it is properly designed, it can work really fast.

Hints:

1. Your program should work with all the provided input files, including the 750000-idr.
2. Because the SSN values are evenly distributed in the input files, the hash function in your imeplementation can be as simple as h(k) = k mod m, where m

is the table size.

3. Use **pointer** to access the elements of the array inside the HashTable object.

Grading:

1. The **correctness** of insert, delete, and retrieval numbers is 50 points.
2. The **reasonable time** needed to finish the processing is 50 points.
3. A submission that cannot be compiled successfully, the maximum grade is 30.
4. A submission using array, vector, or any other data structure other than HashTable class will get grade 0.

Suggestions!

Start working on the project **EARLY**! It takes a while to figure out how all the components work together. Do not wait until the last minute to start working.

Methodology on testing: Write and test one method at a time! Writing them all and then testing will waste your time. If you have not fully understood what is required, you will make the same mistakes many times. Good programmers write and test **incrementally**, gaining confidence gradually as each method is completed and tested.

Wrap up:

Put all your files, including all input files, your head files and cpp files into **project4.zip**. Submit this zip file to Canvas.