# Programming Assignments #3

## Group project: Implementing your hash tables

**Due date: 11/4/2010 Friday 11:59PM**

**Problem (30 points)**

Implement dynamic hash tables using several different collision resolution techniques. In this project, you need to implement three collision resolution techniques, that is, (1) open addressing using **linear** and **quadratic probing,** and (2) **chaining with linked lists**.

Assuming N is the current size of the hash table, the basic hash function for key x is given by

**h(x) =(2x + 5) mod N**

The hash table should begin with size N = 8 and double in size when the load factor (i.e., fraction of hash table entries that are used) reaches 0.5. Also, the table should dynamically shrink when the load factor drops to 0.25, provided that it never gets smaller than N = 8. Recall that when the hash table size changes a re-hashing step is required to move the data from the old table to the new table.

Input

Your executable should assume a file named **input.txt**. The format of this file will be an operation followed by a key (except for PRT) on each line of the input file. The keys will be **distinct** integers. You should be able to handle three types of operations: insert, delete, print. Below is a sample input file:

```
INS  n   // n is an integer.
DEL  n   // n is an integer.
PRT      // print the contents of the hash table.
```

Output

Your executable should print to a file named **output.txt**. You should only print when you are given a PRT command. Every time you see a PRT command you should print the hash table contents of the three hash tables: linear probing, quadratic probing, and chaining in that order, as well as the hash table identifier (Linear, Quad, Chain) and the hash table size. (The format of the example files is provided below.) The three output sequences should be on their own lines in the output file. Numbers on each line should be separated by spaces. You should print the integers in your hash table as follows for different collision resolution techniques:

1. For open addressing, print the hash table array, starting at index 0 and going on until the last element in the array.
2. For chaining, print each linked list at an array location first, starting with index 0 and going on until the last element in the array.

Example Input/Output files

**input.txt:**
INS 412
INS 401
INS 819
INS 213
INS 1009
INS 323
DEL 213
PRT

**output.txt:**
Linear: 819 401 323 1009 412 N=16
Quad: 819 401 1009 323 412 N=16
Chain: 819 323 401 1009 412 N=16

Naming

You need to design and implement three hash table classes based on collision resolution techniques. Since this is not an OO design course. You are not required to use class inheritance. You must follow the following rules to name your classes and files.

- For linear probing-based hash table, the class name is **LinearHashTable**, and file names are **linearhashtable.h/cpp**
- For quadratic probing-based hash table, the class name is **QuadHashTable**, and file names are **quadhashtable.h/cpp**
- For chaining-based hash table, the class name is **ChainHashTable**, and file names are **chainhashtable.h/cpp**
- For test/driver program, the file name is **test.cpp**

Basic operations' prototype

All three hash tables will provide a same set of operations. You have freedom to add other functions if necessary. But, you must provide the basic operations as follows:

- void insert(int x), which insert the integer x into the hash table. It may require hash table expansion and thus re-hashing.

- void remove(int x), which delete the integer x from the hash table. It may require hash table downsizing and thus re-hashing.
- void print(ofstream& fout), which print out to the file the hash table identifier (Linear, Quad, or Chain), followed by the integers stored in the hash table, in the order of increasing index positions. In the end, you need to display the hash table size N.
- Constructor and destructor.

**Requirements:**

1. This is a group project and you can form a team of 2-3 members. You cannot take it as an individual project. It provides a great opportunity for teamwork.
2. You need to design and implement 3 hash table classes, each implementing one of the 3 collision resolution techniques.
3. Each hash table must be able to dynamically adjust its size according to insertions & deletions. You need to avoid memory leak problem for re-hashing.
4. You must have a **README** file that describes
    - your design
    - strengths/limitations
    - and more importantly, each team member's contributions. All team members must be a contributor in this project.

**Submission:**

- Deadline: **11/4/2010 Friday, 11:59PM**
- Make your files into a tar package, named hw3.tar, which includes **Makefile**, **all source files** (Do not include .o and executable files), and **README file**.

  tar –cvf hw3.tar linearhashtable.h linearhashtable.cpp quadhashtable.h quadhashtable.cpp chainhashtable.h chainhashtable.cpp test.cpp Makefile README

- Howto: (ONLY ONE team member is required to submit your team's project!)
  **/home/fac/testzhuy/CPSC250/SubmitHW hw3 hw3.tar**

**Grading:**

- You will receive zero if your program cannot be compiled and executed or if your submission does not include the required files.
- Each hash table implementation takes 10 points, totaling 30 points.
- For each hash table, you need to

- test scenarios where hash table dynamically expand and shrink. (7 points)
- test deletion and see if the item is correctly removed. (2 points)
- check memory leak (1 points)

**Test case:**

One test case for linear probing hash table is provided below. You have to develop your own test cases in order to test your program comprehensively.

**input.txt**

```
INS  8
INS  9
INS  16
PRN
INS  15
INS  50
INS  2
PRN
DEL  2
PRN
DEL  15
DEL  50
PRN
```

**output.txt (linear probing hash table's outputs based on the above input file)**

```
Linear: 8   9   16   N=8
Linear: 16  50  2  8  9  15  N=16
Linear: 16  50  8  9   15    N=16
Linear: 16  8   9    N=8
```