Middle East Technical University Department of Computer Engineering

CENG 213

Data Structures

Fall '2012-2013

Uncle Barney

Due date: 6 January 2013, Sunday, 23:55

1 Objective

This homework is intended to help you gain the practice of implementing a hash table, along with its accompanying hash function. You will be Barney Stinson's best nerd friend, and help him out with his problem.

2 Story

Barney goes into a crowded bar. From the beginning till the end, he is surrounded by a number of mates, who leave their phone numbers for him to call. With an old-fashioned approach, he simply takes an address book with him to save the names & numbers. At the end of the night, he runs into a girl, who says "I am Brenda from downtown. Call me." and leaves. Here is the somewhat-censored call details:

- Hey Brenda, just tell me where you are so ...
- I am Brigitte, you idiot.
* HANG *

He soon realizes the problem. He was too drunk to properly associate the names with phone numbers on the same page, and he called the wrong person. He decides to be more careful about it, nevertheless it still takes too much time to find a person among others. His life would have been much easier if each page included only one person.

Then, Barney comes to you to design him an address book. The address book you design will be of a dynamic size which he will define each time. However, he is simply asking more than a standard book, in which each page has the names starting with the associated letter. He needs a more-complex function to map people to pages(analogous to hash table's buckets), and to access them instantly given the person's details. The distribution of people in the book's pages should be as uniform as possible.

For evaluation, he will take an address book of your design with him on any given night, over the course of m days. Each time, a random n number of people will give him addresses, and k people (again, random) will want him to call at the end of the night. The quality of your address book will be determined by the distribution of people to the pages, as well as average lookup time.

3 Specifications

- You will be implementing a hash table class, named as AddressBook, all by yourself. The bare header file, "AddressBook.h", is given below. You are free to add any private variables/functions to it. However, your hash table should support at least the given functions.

```
class AddressBook{
private:
    // ... members, methods
public:
    // Constructors & Destructor, be careful about memory leaks.

    AddressBook(); // Default table size is 151 buckets.
    // numberOfBuckets is a prime number.
    AddressBook(int numberOfBuckets);
    ~AddressBook();

    // Hash a person to a non-negative integer ( return hash value, not
    // bucket number). Use hashValue(mod numberOfBuckets) as bucket
    // number.
    // Return -1 on invalid parameters.
    int hashPerson(Person& person);

    // Put a person to the hash table, along with the phone number.
    // Do nothing on invalid parameters.
    // Invalid phoneNumber: ""
    void addEntry(Person& person, string phoneNumber);

    // Remove the entry from the address book.
    // Do nothing on invalid parameters.
    void removeEntry(Person& person);

    // Get the total # of people in the given bucket. Return -1 on
    // invalid bucket number.
    int getNumberOfPeopleInBucket(int bucket);

    // Get phone number of a person.
    // Important: To add an extra flavor to your homework, your code
    // will be evaluated by how much time it takes to access a phone
    // number on the average. You will ge a small portion of your grade
    // depending on the average access time of your code.
    // For invalid parameters, return empty string.
    string getPhoneNumber(Person& person);

};
```

- A person's key in Barney's address book will be represented with the Person class. "Person.h" is given below. It should not be modified in any case.

```
#ifndef HW3_Person_h
#define HW3_Person_h
#include <String>
using namespace std;

class Person{
private:
    string name; // Invalid: ""
    int age; // Invalid: age < 18
    string homeTown; // Invalid: ""
public:
    // Constructors and destructor
    Person();
    Person(string name, int age, string homeTown);
    Person(const Person& person);
    ~Person();

    // Getters and setters
    string getName();
    void setName(string name);
    int getAge();
    void setAge(int age);
    string getHomeTown();
    void setHomeTown(string homeTown);
};
#endif
```

- You should try to find a hash function that distributes the people as uniformly as possible to the address book's pages (hash table's buckets). The ultimate (though, unreachable) goal is to create a hashing that aims at most 1 people per bucket. You will be graded on how close you are to this uniformity. You are strongly encouraged to use all three fields when designing your hash function.
- In your final package, you will submit your implementation in "AddressBook.cpp", and will use separate chaining as collision resolution strategy. People will be "chained" to co-exist in the same bucket.
- Sample datasets will be given to you, each including name, age and hometown for every person, as well as their phone numbers and people to call. Test datasets will resemble the given datasets, however, they will also include new people. So, you simply can not design a perfect hash function that assumes all keys and values are known a priori. During testing, after inserting all values, we'll request a number of people's phone numbers from address book.
- Test datasets will have the same nature as the sample datasets (based on which you will tune your hash function). Please note that people with same name, age or hometown may exist in a dataset. However, not all three of them will be the identical for any two people in the same dataset. By maximizing the performance on the given datasets, you'll have a very good chance to have a good grade. You should not worry about the rest.

- If you have a high number of collisions, the time it takes for your code to fetch a person's phone number increases. Try to reduce the number of collisions as much as possible for the given datasets.

  Your responsibility in this context includes providing the correct number of people in a bucket, and of course, returning phone numbers correctly.

- Last but not least, you will analyze the success of your implementation by evaluating it for the given datasets, and submit a report including the analysis. Please read section 5 for details.

- Dynamic resizing/rehashing is not allowed. Your work will be inspected to ensure fairness.

- It is crucial that you provide right numbers both in "getNumberOfPeopleInBucket()" function and your report, as they will be verified by our test setup.

- %90 of your grade will be based on the correctness of your solution, while the remaining %10 will relate to the performance of your code.

4 Dataset Properties

- You will be provided a number of datasets (Night1.txt - ... - NightM.txt), each file representing a new address book used in a night out with Barney. The file format is given below:

```
167 // Number of pages in the address book.
--Mates--
Alice // Name of first entry (person).
30 // Age of first entry.
New York // Hometown of first entry.
0905552321 // Phone number of first entry.
...
...
Belle // Name of Nth entry.
35 // Age of Nth entry.
Oslo // Hometown of Nth entry.
0119120121  // Phone number of Nth entry.
--Calls--
Mary Jane // Name, Call 1.
25 // Age, Call 1.
Istanbul // Hometown, Call 1.
...
...
Hannah // Name, Call K.
32 // Age, Call K.
Varsova // Hometown, Call K.
```

- You will test your code yourselves by reading the input and performing the required operations in your main function. For each entry in "Mates" list, you will call "AddEntry(...)" function after forming the "Person" object with required parameters. Similarly, each entry in "Calls" list requires a call of "getPhoneNumber(...)" function. Please test all of your functions to verify that they work correctly. The time performance test will only take "getPhoneNumber()" calls into account.

- To avoid discrimination, datasets include male names as well as female ones. Uncle Barney is a womanizer, yet your solution should be generic.

5 Report

- You should create a new hash table with the specified size in each dataset, and hash all entries in "Mates" list. Then, you will make the specified calls. Your report should include your table's performance on each dataset.
- The report you will submit should have the following format:
  1. Time Performance: Report average access time for all sample datasets (Only take "getPhoneNumber()" calls into account). To measure time, form a loop and do all access calls successively. Make comments on why certain datasets's performance are different than others. If you have made any optimizations in your code, you can mention them here. To measure time, you can use "clock()" function in "time.h".
  2. Hash Function Quality: Count average number of collisions for all given datasets in "getPhoneNumber()" calls. Explain the nature of the data, and how you improved your hash function.
  3. Bucket Statistics: List average number of entries in non-empty buckets (along with total number of non-empty buckets) for each dataset. Compare them with the ideal values. Comment on the results.

6 Notes

- Please follow the newsgroup regularly, to be aware of any clarifications/updates in the homework.
- Needless to say, you should use C++.
- Your codes will be graded using black-box technique. We will have a "test.cpp" that does extensive checks on all of the functions you implemented.
- We will test your codes on departmental machines using "g++". Please make sure to run tests on ineks.
- You have a total of 7 days for late submission. You can spend this time on a single homework, or distribute the days to all. If you exceed the limit, a penalty of $\mathbf{5 * days * days}$ will be applied.

7 Submission

Submission will be done via COW. You should upload a single zipped file called "hw3.zip", including AddressBook.h, AddressBook.cpp, Person.h, Person.cpp and other files you created, as well as the report. Please do not include a main function in any of your files.

8 Grading

- This homework will be graded out of 100. It will make up %10 of your final grade.
- If your code is working correctly, you get a minimum score of 90. You get the rest 10 points based on the quality of your hash function, and performance of your overall code.

9 Cheating Policy

We have zero tolerance policy for cheating. People involved in cheating will be punished according to the university regulations.