CENG213 Data Structures
Fall 2008
Homework 4:
Hash Table Performance Analysis

Due: 12/01/2008 23:59

General Rules:

- You have 12 days total of late submission for all the homeworks. You can use them in any/all of the homeworks but be sure that your total late submissions are less than the limit.
- There is no teaming up. The homework has to be done individually. Be careful when discussing the homework. Sharing much information can also be accepted as cheating by looking at the similarities between submitted works.
- In case of cheating, all involved get zero.
- Your codes should be written in C++, and will be controlled using g++ on "inek" machines. Be sure that your homework runs on "inek" machines.
- Proper use of indentation is encouraged. You can use auto-indentations of the editors, homeworks without indentation will be penalized by 5%
- Be careful about input/output specifications (do not print any unnecessary characters, white spaces.)
- Submit your homework electronically through https://cow.ceng.metu.edu.tr

Problem Description:

You are faced with the problem of creating a reference index for some books with good computational performance in look-ups. The reference index will consist of the word (key), and the first and last places in the book the word is referenced in, and total number of references to that word. You will implement the index as a hashtable. Your problem is to create the hash table for sample books using different hash functions and algorithms, and analyze their performances.

Input/Output Description:

Your book input will be a plain text file. You are going to read it line by line, and fetch each word in order. Words are separated with whitespaces. You are required to trim the words from any **preceding** or **trailing** special character(s) in this list: `. , : ; ? ! ' " ( )`. If after trimming you only end up with the empty string, you will skip it.

Your position information consists of the the line number (lines start from 1, all lines count) and the index of the word from start of the line (1st word, 2nd word, 3rd word, so on; skipped words do not count).

In your hash table structure, you are going to store the first occurance position, the last occurance position, and number of occurances of every word. You are going to use open addressing for handling collisions.

You are going to use linear and quadratic probing, and 2 different hash functions. Hence,you are required to build 4 different hash tables for your book to analyze and compare their results. (linear probing with the first hash function, linear probing with the second hash function, quadratic probing with the first hash function, quadratic probing with the second hash function).

After completing the hash tables of the book, you are going to calculate the average number of probes needed to get a key and print that statistic for all your 4 hash tables. You are going to assume that each word is equally likely to be queried when calculating the average number of probes. You will query every word in your hash table to calculate the statistic, do not use any formulas.

Remember that resolution errors of division operation's results will accumulate on a **double.** In order to prevent this, sum up number of probes and number of keys as **integer**s, and do the division only at the last step in your homework as a single operation; don't divide and add values continuously.

You will be given some parameters (A, B, C, D, E) for your hash functions from the input. For meaning of these values, see the Hash Functions part.

In addition, you will be given a few words to query and you will look-up and output their information (from any one of the hash tables).

I/O Specs:

```
        ./hw4 <book_filename> <table_size> <A> <B> <C> <D> <E> <query1> <query2>
<query3> .... <query n>
```

sample_book.txt:

```
        Alice's Adventures in
         Wonderland

        Lewis Carroll


        This eBook was designed and published by Planet PDF. For more free

        eBooks visit our Web site at http://www.planetpdf.com/. To hear

        about our latest releases subscribe to the Planet PDF Newsletter.


        Page 2

        Alice's Adventures in Wonderland


        CHAPTER I: Down the Rabbit-Hole

        Alice was beginning to get very tired of sitting by her

        sister on the bank, and of having nothing to do: once or

        twice she had peeped into the book her sister was reading,

        but it had no pictures or conversations in it, 'and what is

        the use of a book,' thought Alice 'without pictures or

        conversation?'
        So she was considering in her own mind (as well as she

        could, for the hot day made her feel very sleepy and

        stupid), whether the pleasure of making a daisy-chain

        would be worth the trouble of getting up and picking the

        daisies, when suddenly a White Rabbit with pink eyes ran

        close by her.
```

Output:

```
8523 unique words
Linear Probing, Hash#1: 2.52435
Quadratic Probing, Hash#1: 1.893431
Linear Probing, Hash#2: 1.457452
Quadratic Probing, Hash#2: 3.232453
<query-word-1> 12:3 433:9 24
<query-word-2> 42:6 42:6 1
<query-word-3> 79:2 180:1 80
....
<query-word-n> 3:1 76:4 9
```

- The table size given will always be equal to or greater than number of unique words in the book.
- 12:3 means that word is seen on 12th line 3rd position
- Write a HashTable class that takes at least 3 arguments in its constructor: probing type, hash function type, table size

Hash functions:

- Cast each character of the string into an unsigned int before using in calculations
- Type of value is unsigned int
- The final "bitwise-and 0xFFFFFFFF" operation is to keep the result within 32 bits in 64 bit systems (on 64 bit systems, an int is 64 bits wide, this will affect your results if not handled properly)
- x << y means left-shift bits of x by y, x >> y means right-shift bits of x by y ; both of these operators exist in C/C++.

Function#1:

```
1. value <- 0
2. for i <- 1...length
3.     value = [ value << A + value >> B + (string[i] mod C) ] & 0xFFFFFFFF
```

- You can test with: A=1..3 B=1..2 C=prime number between 7 - 100

Function#2:

```
1. value <- 1
2. for i <- 1...length
3.     value = [ value * (string[i] mod D) + E] & 0xFFFFFFFF
```

- You can test with: D=prime number between 7 – 100, E=anything