Hash Tables Project 6 220 points

Due Monday December 15, 11:00-1:00
in the Engineering College Library, PL 2600

With Thanks and Credit to: John Edgar, Simon Fraser University, who devised the original version of this assignment.

All students must register this project with our grader Kaveh Ahmadi by December 2. Otherwise, 20 points will be deducted from the final submission.

Kaveh's email: kavehahmadi60@yahoo.com

Assignment

In this assignment you are to implement a hash table for use in a simple spell checker. The spell checker should be set up with a GUI for the interface. The hash table MUST be built as specified in this document.

There is good opportunity for extra credit (see below). Partial credit will be awarded to non-working projects. The assignment will be worth 220 points and be allocated as follows:

Correctness and completeness - 180
Program Style - 20
Human Engineering of the GUI - 20

As for Program Style, you must follow the posted spacing and naming guidelines.

Submission

A printed version of you project must be submitted to the grader, Kaveh, who will be in the Engineering College Library from 11:00-1:00 on Monday December 15. If you need to make other arrangements, contact Kaveh's via the email address posted above.

Projects submiited later that day will be charged a 20 point penalty. Projects submitted up to 2 days late will be charged a 40 point penalty. Later submissions will not be accepted.

Email submissions are NOT allowed. As in previous projects see the posted instructions for

Printed Submission of Projects.pdf

You submission must include a printout of

(a) the text of the document to be checked given as input
(b) a copy of the input highlighting or marking any words that do not appear in the dictionary.
(c) the number of collisions in the hash table and the length of the longest chain
(d) a "ReadMe file" explicitly stating any grounds for extra credit.

There should be multiple sample runs to demonstrate all required functionality.

Pair Submissions

The project can be done in pairs, and we encourage this. Pair projects will be evaluated at a slightly higher standard, at about a 20 points level. For example, a submission worth 270 points would be counted as 250 points if submitted by a pair. Note here that any project can be awarded considerable extra credit.

Hash Table Class

Implement a hash table to store strings (i.e. objects of the Java String class). The Hash Table must be implemented as a data type with the class name "HashTable". The HashTable class will have the following methods:

HashTable ( )	Constructor for a hash table of default size: 1000
HashTable (int n) -	Constructor for a hash table of size n.
void Insert (String S)	Inserts S into the hash table.
boolean Contains (String S)	Return true if S is in the table, false otherwise
int NumEntries ( ) -	Returns the number of strings stored in the table

Hash Function

The insertion of a word requires the computation of an array index

array index = HashValue **mod** array size

The insert function must thus generate a hash value from a given word w. The hash value should be calculated by assigning the values 1 to 26 to the letters **a** to **z** (regardless of letter case) and calculating a numerical value based on the letters in the word as follows:

Group the letters of w in chunks of 4. For each four letters, calculate

$ch_1 * 31^3 + ch_2 * 31^2 + ch_3 * 31^1 + ch_4 * 31^0$

and add the total value of each chunk to get the hash value. For the actual index of the word, compute

HashValue **mod** array size

For example the string "cat" has the hash value 3124:

$3*31^2 + 1*31^1 + 20*31^0 = 2,934$

and the string "table" has the hash value 745,810, calculated as follows

$20*31^3 + 1*31^2 + 2*31^1 + 12*31^0 = 596,855$
$5*31^3 = 148,955$
$596,855 + 148,955 = 745,810$

For a hash table of size 101, "cat" maps into the index

2934 mod 101 = 5

and "table" maps into the index

745810 mod 101 = 26.

Note: In Java, there is a built-in hashCode function which can be applied to a string object S

```
S.hashCode()
```

This built in function computes the hash code for a string S as

```
S[0]*31^(n-1) + S[1]*31^(n-2) + ... + S[n-1]*31^(n-n)
```
$= ch_1*31^{(n-1)} + ch_2*31^{(n-2)} + ... + ch_n$

where `S[i]` is the `i`th character of the string, `n` is the length of the string, and ^ indicates exponentiation. This function may **not** be used.

Insertion and Collisions

Your insert function must deal with collisions (where two different strings have the same hash value). You **must** use **linear probing** to deal with collisions. You should test your insertion method carefully. In particular you should ensure that insertion still works correctly when the hash table is nearly full, and does not result in an error.

For one test, consider setting your hash table's size to be smaller than the size of your word list (see below) and attempting to insert the entire word list into the hash table. This may cause errors that you might not encounter with smaller test data sets. When you get your hash function working correctly, and your hash table is an appropriate size you should be able to successfully enter all the words in the word list (see below) into it.

Let N be the number of strings to be stored in the table. Choose an array size that will have a load factor around 1/3. That is, make the size of the array larger than 3*N. You **must** choose a modulus that is prime (i.e. choose an actual array size that is a prime number just over 3*N) and have your program **automatically** choose this prime number. Automatically means that your program needs to use the size of the word file as a starting point in your search for a prime; do not hard-code a value that happens to work for this data set.

Spell Checker

Use your hash table to implement a simple spell checker. The spell checker should load a dictionary into a hash table, and then test each word in a given document to see if it is contained in the dictionary. For the dictionary you must use the list of frequently used words given in the MS Word file "`Dictionary.doc`" posted on the website. Use MS Word to convert this document into a .txt file called "Dictionary1.txt". Store this file in the project for access by your program in the project.

The document to be spell-checked should also be a .txt file, called "MyDocument.txt", and should also be placed in the project for access by your program

Your spell checker should thus be implemented as a program that reads in the dictionary file and the document file and then checks the document for possible spelling errors.

Extra Credit

Extra credit may not be counted if it is not explicitly stated in the readme document. *It is your resposibility to document in the Readme file what you think counts as extra credit.* It is not our job so somehow deduce what you did for extra credit.

There are many options for extra credit.

a. Use excellent program style (see the posted programming guidelines). Pay special attention to the use of indentation, comments, and naming.
b. Develop a good grahical user interface.
c. Present good documentation of the project.
d. Augment your spell checker so that it tries to change one letter in an incorrect word to find some valid word to suggest.
e. Check a misspelled word against a list of common misspellings.
f. Use a special dictionary (there are some online).
g. Show that your hash table works correctly when the table is almost full or totally full.

h. Allow the user of the GUI to enter his or her own text to be checked.
i. Other options are also fine, just try to be reasonable.

There is no extra credit for the use of pointers, e.g. to achieve chaining of collisions.

Notes

The following incomplete code fragments may be helpful. The first reads a dictionary into a hash table:

```
File F = new File("Dictionary1.txt");
Scanner dictFile = new Scanner(F);
String line;
while (dictFile.hasNextLine()) {
   line = dictFile.nextLine();
   // insert the words in line into the hash table
}
```

For reading a document into an array of single words, consider:

```
public static String readFile(String fileName) {
   // Reads the text file fileName into a single string of words
   File F = new File(docName);
   Scanner docFile = new Scanner(F);
   String result, line;

   result = "";
   while (docFile.hasNextLine()) {
      line = docFile.nextLine();
      result += line + "\n";
   }
   return result;
}

public static String[] getWords(String str){
   // Breaks str into an array of words
   // Splits str on the basis of blanks
   // Returns an array of words
}
```

For testing if an integer is prime:

```
boolean isPrime(int n) {
   if (n%2==0)          //check if n is a multiple of 2
      return false;
   else
      for(int i=3; i*i<=n; i+=2) {
         if(n%i==0)
```

```
              return false;
        }
     return true;
}
```