CS 3345, Programming Project 2

Write a program to simulate an Open Hash Table (1) employing Quadratic Probing, and (2) employing Double Hashing .

The input data specifies the type of probing, the size of the table and includes data to insert, delete, and to find. The table size will not be larger than 16381, and will always be prime. The load factor, λ will not exceed 0.5 in the Quadratic Probing Table and will not exceed 0.9 in the Double Hashing Probing Table..

Create the following classes:

```
class Node {
      private String key; // the key of the element
      private long value; // the data stored
      Node(String ky, long val); // constructor
      . . .
}

class HashTable {
      HashTable(int size);                          // constructor
      boolean insert(String key, long val);         // attempt to insert a record. Return false if
                                                    // the key is already present in the table
      boolean delete(String key);                   // attempt to delete a record. Return false if
                                                    // the key isn't present in the table
      long search(String key);                      // attempt to find a record. Return the value
                                                    // or -1 if the key is not found
      void clearTable();                            // empty the hash table

      int  size();                                  // returns the number of records in the table
      . . .
}
```

The insert, delete, and search functions will employ a function:
`static int hash(String key, int tableSize);` The function will return the int:

$$\left[\left[\sum_{i=0}^{n-1}\left(key.charAt(i)\times 31^{n-i-1}\ mod\ 2^{32}\right)\right]\ mod\ 2^{32}\right]\ mod\ tableSize$$

where there are n characters in the key. So if the key is ABCD the hash() function will return

$[((65\times 31^3\ mod\ 2^{32})+(66\times 31^2\ mod\ 2^{32})+(67\times 31^1\ mod\ 2^{32})+(68\times 31^0\ mod\ 2^{32}))\ mod\ 2^{32}]\ mod\ tableSize$

With tableSize = 23 this equals 20.

1. Use Horner's rule to simplify the computation.

2. Use unsigned int arithmetic to automatically implement $mod\ 2^{32}$.
Java doesn't support unsigned types or unsigned arithmetic, but signed and unsigned integer arithmetic result in the same bit patterns. However the '%' operator isn't the same as the mathematical mod operator. It will interpret a 32 bit int as a signed value and will give a signed result.

For example the unsigned value 4294967283_{10} is encoded in a 32 bit int as $FFFFFFF3_{16}$ but appears to Java's print function and its % operator as -13_{10} and $-13\%7\ yields\ -6$. The correct

result for 4294967283 mod 7 is 5.

Therefore:
3. Use `Integer.remainderUnsigned(sum, tableSize)` to perform the `mod tableSize` function. The result will be an int in $[0, tableSize)$.

Write a fast version of the hash() function and thoroughly test it before proceeding to use it. See the 'H' command below. The function computes the table index where the first probe should take place.

Input Data:
The input data will be read from a text file named **Hash_in.txt**. The data will begin with a single character 'Q' or 'D' to designate the type of probing to employ, Quadratic, or Double. The next line will contain one integer M giving the size of the table, $M \leq 16381$, M prime. If Double Hashing is to be used, the third line will contain an integer R, as required by the Double Hashing algorithm. It will be prime and in [3,101] and will always be less than M. If Quadratic Probing is to be used, the third line will contain the number zero.

NOTE THE 4th LINE HAS BEEN ELIMINATED FROM THE SPEC'

Each of the following lines will have one of the following formats, where "k" is a sequence of up to 20 upper and/or lower case alphabetical characters, and "v" is an integer in $[1, 2^{63} - 1]$:

```
I k v       // insert record with key k and value v.
            // Print "Key k inserted" or "Key k already exists"
J k v       // Insert record with key k and value v and print nothing
D k         // delete record with key k.
            // Print "Key k deleted" or "Key k doesn't exist"
F k         // delete record with key k and print nothing  // NOTE THE CHANGE FROM COMMAND 'E'
S k         // search for key k and print "Key k found, record = v" or "Key k doesn't exist"
T k         // search for key k and print nothing
P           // print "Number of records in table = #####"
C           // delete all hash table entries.
Q           // print the following six integers in the order given below,
            // space separated on a line:
            // total number of successful inserts,
            // total number of probes on all successful inserts,
            // total number of successful searches,
            // total number of probes on successful searches,
            // total number of unsuccessful searches,
            // total number of probes on unsuccessful searches,
            // all of these quantities should be measured since the program began,
            // or since the last C command was read
H k         // print the string k, then a space, and then hash(k,tablesize) on a line
E           // End of input file
```

If an insert causes the table or overflow, or an insert is impossible because the table is already half full when using Quadratic Probing, print "Table Overflow" and exit the program.

Sample Input	Output for Sample Input
Q	Key Salvage inserted
23	Key Garbage inserted
0	Key refuse inserted
I Salvage 123456	Number of records in table = 3
I Garbage 3456789012	Key waste inserted
I refuse 4567890123	Key scrap inserted
P	Key drivel inserted
I waste 5678901234	Key refuse already exists
I scrap 6789012345	Key refuse already exists
I drivel 7890123456	Key Junk doesn't exist
I refuse 5567890123	Key key doesn't exist
I refuse 6567890123	Key Salvage deleted
S Junk	Key trash doesn't exist
S key	Key scraps inserted
D Salvage	Key obsolete inserted
D trash	Key deprecated inserted
I scraps 89012345678	Key trash doesn't exist
I obsolete 9012345678	Key Salvage doesn't exist
I deprecated 0123456789	Key Salvage inserted
S trash	Key Junk doesn't exist
S Salvage	Key refuse found, record = 4567890123
I Salvage 1234567899	Key Salvage found, record = 1234567899
S Junk	Number of records in table = 10
S refuse	10 12 2 2 5 9
S Salvage	ABCD 20
P	abcdefghijklmnopqrst 9
Q	
H ABCD	
H abcdefghijklmnopqrst	
E	

With linear probing and the same sequence of operations above, the Q command prints 10 12 2 2 5 7, and with double hashing and R = 7, the Q command prints 10 12 2 2 5 8

RULES FOR PROGRAMMING AND SUBMISSION:

(1) YOUR SUBMISSION MUST BE YOUR OWN WORK. CITE ANY ELEMENTS OF CODE THAT YOU COPY FROM ELSEWHERE. THIS IS NOT A GROUP PROJECT!
(2) Write your program as one source file in Java or C++ and do not include the "package" construct in your Java source.
(3) Name your source file as $N_1N_2F_1F_2$P2.java where your given name begins with the characters N_1N_2 and your family name begins with the characters F_1F_2. For example my name is Ivor Page, so my source file will be called IVPAP2.java. Note that in all but the "java" extension, the characters are upper case.
(4) Test your program with the data files provided on eLearning for this project.
(5) Your program must not output any prompts. Its output must exactly match the format of the column, **Output for Sample Input**, above
(6) Do not use any Java Collection Classes except Strings, an array of Strings if you use String.split(), and the array of your HashTable class.
(7) Your program must compile and run on one of the compilers in the man Computing Lab ECSS 2.103/104.
(8) Your program must read from a text file with name **Hash_in.txt** and output to System.out (cout for C++).
(9) Use good style and layout. Comment your code well.
(10) Submit your ONE source code file to the eLearning Assignment drop box for this assignment. Don't submit a compressed file.
(11) **There will be a 1% penalty for each minute of lateness. After 60 late minutes, a grade of zero will be recorded.**

Send any questions/corrections to ivor@utdallas.edu.

Class layout for project
Here is the basic Java class layout that should apply to project 2 - all classes in one source file.

```
import java.util.*;
import java.io.*;   // for reading from a named file

class IVPAP2 { // for my project 2
   // static functions go here
   . . .
   public static void main(String [] args) throws IOException {
       //the "throws" statement is for reading from a named file
        Scanner sc = new Scanner(new File(Hash_in.txt));
        . . .
   }
} // end of IVPAP2

class Node {
   . . .
}

class HashTable {
   . . .
}
// the three class definitions can be in any order.
```