Project 3: Hash Table (in Java): You are to implement a hash table (an 1D array of ordered linked list) of B buckets).

- The bucket size, B, will be enter by user from console.
- The hash function for the hash table is the "Doit" function which was taught in class.
- The input to your program is a text file contains a list of triplets {<op data>} where op is a character either + or - or ?; where + means insert, - means delete, and ? means information retrieval and data is a character string.

****

Run your program twice, first with bucket size = 7 and next with bucket size 13.

Include in your hard copy *.pdf file:

- 1 page cover page
- source code
- outFile1 with bucket size = 7
- outFile2 with bucket size = 7
- outFile1 with bucket size = 13
- outFile2 with bucket size = 13

****

Language: Java

****

Project points: 10 pts

Due Date: Soft copy (*.zip) and hard copies (*.pdf):

+1 (11/10) 9/11/2021 Sunday before midnight
-0 (10/10) 9/15/2022 Thursday before midnight
-1 (9/10) for 1 day late: 9/16/2022 Friday before midnight
-2 (8/10) for 2 days late: 9/17/2022 Saturday before midnight
(-10/10) non-submission: 9/17/2022 Saturday after midnight

*** Name your soft copy and hard copy files using the naming convention as given in the project submission requirement discussed in a lecture and is posted in Google Classroom.

*** All on-line submission MUST include Soft copy (*.zip) and hard copy (*.pdf) in **the same email attachments** with correct email subject as stated in the email requirement; otherwise, your submission will be rejected.

****

I. Inputs:

a) inFile (args[0]): A text file contains a list of triplets {<op name}

For example,

```
+   Jesse
+   Ning
?   Ning
+   Asadbek
-   Asadbek
```

b) BucketSize // from args[1]; run your program 2 times, first time use 7 and second time use 13.

****

II. outputs:

a) outFile1 (args[2]): Print the final result of the hash table: from 0 to bucketSize-1, one linked list per text line.

For example (let B be the bucketSize):

HashTabel [0]: (dummy dummy'next's data) → (data next's data) → ….
HashTabel [1]: (dummy dummy'next's data) → (data next's data) → ….
:
:
HashTabel [B-1]: (dummy dummy'next's data) → (data next's data) → ….

b) outFile2 (args[3]): Print all intermediate outputs, for debugging and for you to see how your program is doing.

****
III. Data structure:
****
- listNode class
    - (string) data
    - (listNode) next
    methods:
    - constructor (data) //create a node with given data
- hashTable class
    - (char) op // either '+' or '- or '?'
    - (string) data
    - (int) bucketSize
    - (listNode) hashTable [] // 1D array of ordered linked list, size of bucketSize, dynamically allocated.
  method:
    - constructor (...) // dynamically allocates hashTable [], size of bucketSize,
        // for each hashTable [i] points to a dummy node ("dummy", null)
        // On your own! You should know how to do this.
    - (int) Doit (data) // a string hash function which returns an 'index' between 0 to bucketSize-1
        // The function given below.
    - informationProcessing (…) // see algorithm below.
    - (listNode) findSpot (…) // Given the index, the method searches thru hashTable[index] linked list to locate the
        //node match with data. See algorithm below.
    - hashInsert (…) // see algorithm below.
    - hashDelete (…) // see algorithm below.
    - hashRetrieval (…) // see algorithm below.
    - printList (index, outFile) // print the index and the
        // linked list of hashTable [index], use the format given in the above.
    - printHashTable (outFile) // output the entire hashTable, call printList (…), index from 0 to bucketSize -1.

****
IV. Main (…)
****
Step 1: inFile ←open input file using args [0]
  bucketSize ← args [1]
  outFile1, outFile2 ← open output files using args [2] and args [3]
Step 2: createHashTable (…) // use constructor
Step 3: informationProcessing (inFile, outFile2)
Step 4: printHashTable (outFile1)
Step 5: close all files

****
V. (int) Doit (data)
****
Step 0: long value ← 1
Step 1: i ← 0
Step 2: length ← use Java's string length function
Step 3: char oneCh ← use Java function to convert data[i] to a character
Step 4: value ← value * 32 + (int) oneCh
Step 5: i++
Step 6: repeat step 3 – step 4 while i < length
Step 7: return (value % bucketSize)

```
****************************************
VI.  informationProcessing (inFile, outFile2)
****************************************
Step 0: outFile2 ← "Enter informationProcessing method"
Step 1: op, data ← get from inFile
Step 2: outFile2 ← output op, data (with description stating what you are printing)
Step 3: index ← Doit (data)
        outFile2 ← print index (with description stating what you are printing)
Step 4: printList (index, outFile2) // with description stating which bucket you are printing before operation
Step 5: if op == '+'
                hashInsert (index, data, outFile2)
        else  if op == '-'
                hashDelete (index, data, outFile2)
        else  if op == '?'
                hashRetrieval (index, data, outFile2)
        else
                outFile2 ← "op is an unrecognize operation!"
Step 6: repeat step 1 to step 5 until inFile is empty.

****************************************
VII.  (listNode) findSpot (index, data)
****************************************
Step 1: Spot ← hashTable[index] // points to dummy node
Step 2: if Spot's next != null *and*  Spot's next data < data // use string comparison!!
                Spot ← Spot's next
Step 3: repeat Step 2 until condition failed  // either Spot's next is null or Spot's next data >= data
Step 4: return Spot

****************************************
VIII.  hashInsert (index, data, outFile2)
****************************************
Step 0: outFile2 ← print message: "*** enter hashInsert method. Performing hashInsert "
Step 1: Spot ← findSpot (index, data)
Step 2: if (Spot's next != null *and* Spot's next data == data)
                outFile2 ← print message: "*** Warning, data is already in the database!"
        else
                newNode ← get a listNode with data // Use listNode constructor
                newNode's next ←Spot's next
                Spot's next ← newNode
                outFile2 ← " After hashInsert operation … "
                printList (index, outFile2)
****************************************
IX.  hashDelete (index, data, outFile2)
****************************************
Step 0: outFile2 ← print message: "** Inside hashDelete method.  Performing hashDelete "
Step 1: Spot ← findSpot (index, data)
Step 2: if Spot's next == null or  Spot's next data != data)
                outFile2 ← print message: "*** Warning: data  is *not* in the database!"
        else
                temp ← Spot's next // the node to be deleted
                Spot's next ← temp's next
                temp's next ← null
                outFile2 ← " After hashDelete operation … "
                printList (index, outFile2)
```

```
****************************************
X.  hashRetrieval (index, data, outFile2)
****************************************
Step 0: outFile2 ← print message: "** Inside hashRetrieval. Performing hashRetrieval"
Step 1: Spot ← findSpot (index, data, outFile2)
Step 2: if Spot's next == null or  Spot's next data != data)
                outFile2 ← print message: "*** Warning, the record  is *not* in the database!"
        else
                outFile2 ← print message: "Yes, the record is in the database!"
```