Project 2: Hash Table (in C++): You are to implement a hash table (an 1D array of ordered linked list) of B buckets, fThe bucket size, B, will be given via argv[2] in command line. The hash function for the hash table is the "Doit" function given in the lecture note and have discuss in the lecture. The input to your program is a text file contains a list of triplets {<op firstName lastName >} where op is either + or - or ?; + means insert, - means delete, and ? means information retrieval; firstName and lastName are character strings. lastName in the triplet is the key passes to the hash function to get the bucket index from the hash function for information storage and retrieval.

****

Add the following into your #include:

```
#include<string>
using namespace std;
```

****

What your program will perform:

1. Read the input triplet: op firstName lastName
2. if op is +, get the index from Doit(lastName), then, go to hashTable[index] to perform insertion process
   If op is -, get the index from Doit(lastName), then, go to hashTable[index] to perform deletion process
   If op is ?, get the index from Doit(lastName), then, go to hashTable[index] to perform information retrieval process
3. output the results to outFiles.
4. Run your program twice, first with bucket size = 29 and next with bucket size 43.
5. Include in your hard copy *.pdf file:
   - 1 page cover page
   - source code
   - outFile1 with bucket size = 29
   - outFile2 with bucket size = 29
   - outFile1 with bucket size = 43
   - outFile2 with bucket size = 43

****

Language: C++

****

Project points: 10 pts

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

-0 2/20/2021 Saturday before midnight
-1 for 1 day late: 2/21/2021 Sunday before midnight
-2 for 2 days late: 2/22/2021 Monday before midnight
-10/10: 2/22/2021 Monday 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: There will be two inputs to the program:

a) inFile (use argv[1]): A text file contains a list of triplets {<op firstName lastName.}

For example,

```
+       Longcheng       Ochilov
+       Sweyaksha       Webster
-       Longcheng       Ochilov
+       Pengdwende      Cesa
?       David   Chowdhury
+       Kushal  Zheng
+       Mohammed        Aucacama
-       Sweyaksha       Kim
:
```

b) BucketSize (use argv[2]): Run your program twice, first with bucket size = 29 and next with bucket size 43,

****

II. outputs: There will be two output files:

a) outFile1 (use argv [3]): Print the final result of the hash table: B ordered linked lists, one linked list per text line.

For example (let B be the bucketSize):

HashTabel [0]: (dummyFirst dummyLast next's firstName) → (firstName lastName next's firstName) → ….
HashTabel [1]: (dummyFirst dummyLast next's firstName) → (firstName lastName next's firstName) → ….
:
:
HashTabel [B-1]: (dummyFirst dummyLast next's firstName) → (firstName lastName next's firstName) → …..

b) outFile2 (use argv [4]): Print all intermediate outputs, to help you debugging!

****

III. Data structure:

****

- listNode class
    friend of hashTable
    - (string) firstName
    - (string) lastName
    - (listNode *) next

    methods:
    - constructor (firstName, lastName) //create a node with given data
    - printNode (node) // use the format:
      (this node's firstName, this node's lastName, next node's firstName) →
      // see example given in the above.
- hashTable class
    - (char) op // either '+' or '- or '?'
    - (int) bucketSize // via argv[2]
    - (listNode *) hashTable [bucketSize]

    method:
    - createHashTable (...) // The method dynamically allocates hashTable [], size of bucketSize,
      //where each entry point to a dummy node: ("dummyfirst", "dummylast", null)
      // On your own! You should know how to do this.
    - (int) Doit (lastName) // Given the lastName, the method returns the 'index' between 0 to bucketSize-1
      // The function can be found in the lecture note.
    - informationProcessing (inFile, outFile2) // see algorithm below.
    - (listNode *) findSpot (index, firstName, lastName) // search thru hashTable[index] linked list to locate the record with firstName and lastName. See algorithm below.
    - hashInsert (…) // see algorithm below.
    - hashDelete (…) // see algorithm below.
    - hashRetrieval (…) // see algorithm below.
    - printList (index, outFile) // print 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 argv[1]
         bucketSize ← argv[2]
        outFile1, outFile2 ← open output files using argv[3] and argv[4]
Step 2:  createHashTable (…)
Step 3: informationProcessing (inFile, outFile2)
Step 4: printHashTable (outFile1)
Step 5: close all files
```

```
*****************************************
VI.  informationProcessing (inFile, outFile2)
*****************************************
Step 1: op, firstName, lastName ← get from inFile
Step 2: outFile2 ← print op, firstName, lastName (with description)
Step 3: index ← Doit (lastName)
         outFile2 ← print index (with description)
Step 4: printList (index, outFile2)
Step 5: if op == '+'
                hashInsert (index, firstName, lastName outFile2)
        else  if op == '-'
                hashDelete (l index, firstName, lastName outFile2)
        else  if op == '?'
                hashRetrieval (index, firstName, lastName outFile2)
Step 6: repeat step 1 to step 5 until inFile is empty.
```

```
*****************************************
VII.  hashInsert (index, firstName, lastName, outFile2)
*****************************************
Step 0: outFile2 ← print message: "***Performing hashInsert on firstName, lastName "
Step 1: Spot ← findSpot (index, firstName, lastName)
Step 2: if (Spot's next != null *and* Spot’s next’s lastName == lastName *and*  Spot’s next’s firstName == firstName)
                outFile2 ← print message: "*** Warning, the record is already in the database!"
        else
                newNode ← get a listNode with firstName, lastName // Use listNode constructor
                newNode’s next ←Spot’s next
                Spot’s next ← newNode
                printList (index, outFile2)
```

```
*****************************************
V.  (listNode *) findSpot (index, firstName, lastName)
*****************************************
Step 1: Spot ← hashTable[index]
Step 2: if Spot’s next != null *and*  Spot’s next’s lastName < lastName // string comparison!!
                Spot ← Spot’s next
Step 3: repeat Step 2 until condition failed
Step 4: if Spot’s next != null *and*  Spot’s next’s lastName == lastName *and* Spot’s next’s firstName < firstName
                Spot ← Spot’s next
Step 5: repeat step 4 until condition failed
Step 6: return Spot
```

****

VIII. hashDelete (index, firstName, lastName, outFile2)

****

```
Step 0: outFile2 ← print message: "*** Performing hashDelete on firstName, lastName "

Step 1: Spot ← findSpot (index, firstName, lastName)

Step 2: if (Spot's next != null *and* Spot’s next’s lastName == lastName *and*  Spot’s next’s firstName == firstName)

                Junk ← Spot’s next
                Spot’s next ← Spot's next next
                Junk's next ← null
                Free junk
                printList (index, outFile2)
        else
                 outFile2 ← print message: "*** Warning, the record  is *not* in the database!"
```

****

VII. hashRetrieval (index, firstName, lastName, outFile2)

****

```
Step 0: outFile2 ← print message: "*** Performing hashRetrieval on firstName, lastName "

Step 1: Spot ← findSpot (index, firstName, lastName)

Step 2: if (Spot's next != null *and* Spot’s next’s lastName == lastName *and*  Spot’s next’s firstName == firstName)

                outFile2 ← print message: "Yes, the record is in the database!"

        else
                outFile2 ← print message: "No, the record is not in the database!"
```