100 Points

Due date:
11:59pm, Sunday  09/15/2019 for Monday labs
11:59pm, Tuesday  09/17/2019 for Wednesday labs
11:59pm, Thursday 09/19/2019 for Friday labs

Purpose:
The purpose of this lab is to implement a hash table with open hashing in C++.

General Requirements:
For this lab, you are required to implement a hash table with separate chaining using a singly linked list (convert your previous doubly linked list to a singly linked list). You are to read in the elements from a data file. There shouldn’t be any duplicate elements inserted into the hash table.

Let the index be determined by the hash function (index = hashfunction(element, bucket_size)) where the hashfunction() is defined as:

Evaluate the ASCII value of the word/key using the formula total_ascii_val = ascii_val(stri[0]) + ascii_val(stri[1]) + ... + ascii_val(stri[str_len]), where str[i] indicates the letter stored in position i of the string. For example, take the word to be ‘cat’. The total_ascii_val for ‘cat’ will be computed as:
99 (for c) + 97 (for a) + 116 (for t) = 312.

Finally, the function should return the index of the array at which the respective linked list is located. The index will be computed according to the formula index = (total_ascii_val % bucket_size) where bucket_size will be passed as a parameter to the function. Here bucket_size is the total number of buckets, denoted by m in the upcoming formulae.

For example, the index for the word ‘cat’ will be 312% 5 = 2, where 5 is the bucket size. That means, the word ‘cat’ will be inserted in the linked list starting at array index 2.

The file from which you should read the keys/elements is called ‘data.txt’. You may hard code the file name if you wish. After building the structure, your program should ask the user to choose one of the options below:

Hash:
The hash table should implement an appropriate constructor and destructor. The rest of the methods should be implemented as follows:
- Insert(x): Should insert x into the hash table when it is not found. All the words/keys must be in lowercase. Insertion must be done at the beginning (towards the head) of the chain. Check the outputs given below to see the insertion sequences. The output will be “x was added successfully” when x is inserted; otherwise, the output will be “x was not added successfully”. Duplicate values are not allowed.
• Delete(x): Should remove x from the hash table. The output should be either “x has been deleted” or “x was not found”.

• Print(): Should print out all the elements of the hash table. Each chain must be separated by an arrow and end with an endl. It should output from bucket 0 to bucket m-1 in the format:

  Bucket 0: Element list
  Bucket 1: Element list
  ...
  Bucket m-1: Element list

• If there are n entries and m is the size of the array, ideally there would be n/m entries assigned to each index. This value lambda = n/m is called the load factor.

• Hashfunction(x, bucketsize) – Should compute the sum of the ASCII values of each letter of the word and then take the modulus with bucketsize. The description of the hash function is given on page 1. The initial bucket size will be such that the initial load factor, i.e., lambda, is around 75% - 80%. Suppose if you have four input elements, your bucket size should be 5 (which sets a ceiling for the load factor at 80%).

• Rehash(x) – Rehash the table when lambda > 1 and show the output “hash table is rehashed”. Rehash should hash all the elements of an existing hash table into a new hash table where the table size becomes the smallest prime number after 2*m, where m is the initial bucketsize.

• Find(x) – If the key x is found in the hash table, the output will be “x is found” along with the location where it was found; otherwise, the output will be “x is not found”.

Expected Results:
data.txt elements: snowhall, stronghall, wescoe, eaton

Take m=5 (where m is the initial bucketsize) and note that after rehash, the bucketsize will be the smallest prime number which comes after 2*m.
Please choose one of the following commands:
1- Insert  
2- Delete  
3- Find  
4- Print  
5- Exit

>1
Enter an element to be inserted:
>lawrence
‘lawrence’ is added to the hash table.

Please choose one of the following commands:
1- Insert  
2- Delete  
3- Find  
4- Print  
5- Exit

>1
Enter an element to be inserted:
>lawrence
‘lawrence’ was not added successfully.

Please choose one of the following commands:
1- Insert  
2- Delete  
3- Find  
4- Print  
5- Exit

>2
Enter an element to be removed:
>stronghall
‘stronghall’ is removed from the hash table.

Please choose one of the following commands:
1- Insert  
2- Delete  
3- Find  
4- Print  
5- Exit

>4
0: -> eaton
1: -> wescoe
2: -> snowhall
3:
4: -> lawrence

Please choose one of the following commands:
1- Insert
2- Delete
3- Find
4- Print
5- Exit

>3
Enter an element to be found:
>wescoe
‘wescoe’ is found at location 1.

Please choose one of the following commands:
1- Insert
2- Delete
3- Find
4- Print
5- Exit

>1
Enter an element to be inserted:
>jayhawk
‘jayhawk’ is added to the hash table.

Please choose one of the following commands:
1- Insert
2- Delete
3- Find
4- Print
5- Exit
>4

0: -> eaton -> stronghall
1: -> jayhawk -> wescoe
2: -> snowhall
3:
4:

*Note that at this point, lambda = 1. If any other element is added, lambda will be greater than 1 and the hash table will be rehashed.

------------------------------------------------------------
Please choose one of the following commands:
1- Insert
2- Delete
3- Find
4- Print
5- Exit
>1
Enter an element to be inserted:
>rockchalk
The hash table is rehashed.

------------------------------------------------------------
Please choose one of the following commands:
1- Insert
2- Delete
3- Find
4- Print
5- Exit
>4

0: -> rockchalk
1:
2: -> lawrence
3: -> jayhawk -> snowhall
4:
5:
6:
7: -> eaton
8: -> wescoe
9:
10:

------------------------------------------------------------
Please choose one of the following commands:
EECS 560 Lab – Implementation of Hash Table
Prof.: Dr. Shontz, GTA: Chiranjeevi Pippalla, Anubhav Ghosh

1- Insert
2- Delete
3- Find
4- Print
5- Exit
>5
Bye Bye!

Submission:
Follow the conventions below to facilitate grading:

Source Code
• Place all your source files (*.cpp, *.hpp) and input files in a single folder with no subfolders.
• Name your folder using the convention Lastname_LabX (e.g., Smith_Lab02).
• Include a functioning Makefile inside the folder. (The makefile should also include the clean command.)
• Verify that your code runs on the Linux lab machines.

Compressed File
• Compress the folder using .zip, .rar, or .tar.gz.
• Name your file using the convention Lastname_LabX (e.g., Smith_Lab02.zip).

Email
• Use Lastname_LabX (e.g., Smith_Lab02) as the subject line for your e-mail.
• Send your code to chiranjeevi.pippalla@ku.edu if you are in one of Chiranjeevi’s sections and to anubhav@ku.edu if you are in one of Anubhav’s sections.
• Anytime you have a question about the lab, put the word question in the subject of the email.