EECS 560 Lab – Implementation of Hash Tables (Closed Hashing)

Prof.: Dr. Shontz, GTAs: Chiranjeevi Pippalla, Prashanthi Mallojula

100 Points

Due date:
11:59pm, Monday, 02/17/2020 for Tuesday labs
11:59pm, Wednesday, 02/19/2020 for Thursday labs

Purpose:

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

General Requirements:

In this lab, you are required to implement a hash table with Linear Probing and Quadratic Probing using an array of structures (not arrays in STL).

Implement a user credential storage system. Each record consists of a Username and Password. Specifications for each field are provided below. Password is considered the key to be used to compute the hash value, that is, the index of the hash table at which the record will be placed. You are to read in the records from a data.txt file. **There shouldn't be any duplicate records inserted into the hash table. That is, Username has to be unique, but Password can be duplicated.**

Username: String (of length not more than 6 and greater than 1). Only lower case letters are allowed.

Password: Alphanumeric. A combination of characters and numbers of length not more than 7 (at least 3 characters and at least 2 numbers). Only lower case letters are allowed. No special characters (for example : ., * , ? ,@ ,#, etc.) are allowed.

Hash function: The key to compute the hash value is the Password.

Hash value = (Sum of Ascii values of all characters + Sum of integers) % Table size
For example: Password = kuit560 and Table size = 11
Hash value = (107+117+105+116+5+6+0) % 11 = 456 % 11 = 5
Let say the Hash table name is password_hash.
Then the record is stored in password_hash[5].

Your data.txt file may look like either of the below mentioned examples:

Example 1:

json , kuit560
java, eng2020
python, kuit560
nodejs, cls334
angular, lang99
hadoop, bgdt564
spark, onl345

csharp, pwd3456

Or,

Example 2:

json:kuit560, java:eng2020, python:kuit560, nodejs:cls334, angular:lang99, hadoop:bgdt564, spark:onl345, csharp:pwd3456

Only the **password will be used as the key for calculating the hash value with the hash function**. The file you read the data from will be named **data.txt**. You may hard code the file name if you wish. After everything has been read from data.txt into the hash table (which is an Array of Structures), your program should ask the user to choose one of the options below:

Please choose one of the following commands:

1- Add User
2- Remove User
3- Forgot Password
4- Forgot Username
5- Print Users
6- Exit

Hashing details:

If there is no collision, the index should be calculated as per the hash function given above. Else, try resolving the collision using Linear Probing and Quadratic Probing as explained below:

Hashing with Linear Probing:

$f_i = i$,$\forall i$, $0 \leq i \leq k-1$, where x = summation of ASCII values and numbers

$h_0(x) = (h(x) + 0) \bmod table\_size = h(x) \bmod table\_size$,

$h_1(x) = (h(x) + f_1) \bmod table\_size = (h(x) + 1) \bmod table\_size$,
$h_2(x) = (h(x) + f_2) \bmod table\_size = (h(x) + 2) \bmod table\_size$,
...
$h_{k-1}(x) = (h(x) + f_{k-1}) \bmod table\_size = (h(x) + (k-1)) \bmod table\_size.$·

Hashing with Quadratic Probing:

$f_i = i^2$, $\forall i$, $0 \leq i \leq k-1$, where x = summation of ASCII values and numbers

$h_0(x) = (h(x) + 0^2) \bmod table\_size = h(x) \bmod table\_size$,

$h_1(x) = (h(x) + f_1)$ mod table_size = $(h(x) + 1^2)$ mod table_size,

$h_2(x) = (h(x) + f_2)$ mod table_size = $(h(x) + 2^2)$ mod table_size,

...

$h_{k-1}(x) = (h(x) + f_{k-1})$ mod table_size = $(h(x) + (k-1)^2)$ mod table_size.

Sometimes, it may happen that it is not possible to insert a particular record even when there are open buckets in the hash table. **For such cases, try to find the hash value until k = table_size** to insert the record. If there is no valid hash position found, then report failure for the insertion.

Rehash:

For closed hashing, rehashing is required when the Load factor is greater than 0.5.
Load factor = (number of elements in the hash table)/table_size

You need to rehash the hash table whenever the Load factor is greater than 0.5. The size of the new table should be the prime number closest to twice the size of the current table. Before inserting a record, you need to check if rehashing is required. If it is required, apply rehashing, that is, all the hash elements need to be hashed into new locations in the new hash table. This should be done sequentially. The new record should be rehashed whenever you reach the location in the old table where it would have been inserted. **The hash function and rehash function need to be implemented in the code but need not be displayed in the output operations.**

Operations:

The hash table should implement an appropriate constructor and destructor. The rest of the methods should be implemented as follows:

- **Add User(username, password)**: Should insert parameters into both of the hash tables (maintained by the Linear probing and Quadratic probing schemes) when it is not already present. Insertion must be done at the location which is obtained by the hashing function. If the location is full, then a new location will be calculated with Linear probing and Quadratic probing, respectively. Two different messages may be printed. The first message that will be printed will report on the success/failure of the insertion with Linear probing. (The message in this case will look like: "[user] has been successfully added at location[index x] using Linear probing") Similarly, the second one will report on the success/failure of the insertion with respect to Quadratic probing. (For example, "[user] has been inserted at location[index y] using Quadratic probing.") Altogether, there can be four different cases like success-success, success-failure, failure-success, and failure-failure, all of which should be handled by the two messages shown as output.

- **Remove User(username, password)**: Calculate the hash value based on password and then then remove the record from the hash table. The output should be either "[user] has been removed from the hash table." or "Record does not exist". There will be two cases of searching for an element before deletion:

1. A bucket that is always empty: Searching terminates. This means the user does not exist in the hash table. Hence, deletion is not possible.
2. A bucket that is emptied by deletion: Searching must continue. This means the user record with the same hash value was emptied and the required element needs to be found and deleted.

Hint: Try using an extra Boolean flag (true/false) to know if the bucket is emptied or not.

- **Forgot Password:** Prompt the user to enter the username. Find the record based on username. This should print the whole record**.**

  Output for Linear Probing

  username: password

  Output for Quadratic Probing

  username: password

- **Forgot Username:** Prompt the user to enter the password. Find the record based on the password. This should print the whole record.

  Output for Linear Probing

  Username: password

  Output for Quadratic Probing

  username: password

- Print Users: Should print out all the records of the hash table. This should print the whole record. Display the output of Linear probing and Quadratic Probing in the format given below: (Assume m is the table_size.)

  Output for Linear Probing

  Index[0]: username : password

  Index[1]: username : password

  …..

  Index[m-1]: username : password

  Output for Quadratic Probing

Index[0]: username : password

Index[1]: username : password

…..

Index[m-1]: username : password

- **Exit:** Should exit the program.

Expected Results:
data.txt records:
json:kuit560, java:eng2020, python:kuit560, nodejs:cls334, angular: lang99, hadoop:bgdt564, spark:onl345, csharp:pwd3456

The following outputs are just for illustrative purposes only.

The bucket size is 11. For the following examples, the bucket size is considered for illustrative purposes. You can consider table size m=11 as your initial table_size while implementing. Rehash whenever it is required (as mentioned in the rehash function details).

Please choose one of the following commands:

1- Add User
2- Remove User
3- Forgot Password
4- Forgot Username
5- Print Users
6- Exit

>Input: 1
Enter user details to be added:
>dotnet
>onl223

Linear Probing:
Record successfully inserted

Quadratic Probing:
Record successfully inserted

Please choose one of the following commands:

1- Add User
2- Remove User
3- Forgot Password
4- Forgot Username
5- Print Users
6- Exit

>5
Linear Probing:
0:spark:onl345
1:
2:nodejs:cls334
3:hadoop:bgdt564
4:
5:json:kuit560
6:python:kuit560
7:angular:lang99
8:csharp:pwd3456
9:dotnet:onl223
10:java:eng2020

Quadratic Probing:
0:spark:onl345
1:
2:nodejs:cls334
3:hadoop:bgdt564
4:dotnet:onl223
5:json:kuit560
6:python:kuit560
7:angular:lang99
8:csharp:pwd3456
9:
10:java:eng2020

Please choose one of the following commands:

1- Add User
2- Remove User
3- Forgot Password
4- Forgot Username
5- Print Users
6- Exit

>3
> Enter user name:
> spark

EECS 560 Lab – Implementation of Hash Tables (Closed Hashing)

Prof.: Dr. Shontz, GTAs: Chiranjeevi Pippalla, Prashanthi Mallojula

Linear Probing:

spark:onl345

Quadratic probing:

spark:onl345

Please choose one of the following commands:

1- Add User
2- Remove User
3- Forgot Password
4- Forgot Username
5- Print Users
6- Exit

>4
Enter Password:
> onl223

Linear Probing:

dotnet:onl223

Quadratic probing:

dotnet:onl223

Please choose one of the following commands:

1- Add User
2- Remove User
3- Forgot Password
4- Forgot Username
5- Print Users
6- Exit

>2
Enter user and password to be removed:
>java
>eng2020

EECS 560 Lab – Implementation of Hash Tables (Closed Hashing)

Prof.: Dr. Shontz, GTAs: Chiranjeevi Pippalla, Prashanthi Mallojula

Linear Probing:

Record removed

Quadratic probing:

Record removed

Please choose one of the following commands:

1- Add User
2- Remove User
3- Forgot Password
4- Forgot Username
5- Print Users
6- Exit

>5

Linear Probing:
0:spark:onl345
1:
2:nodejs:cls334
3:hadoop:bgdt564
4:
5:json:kuit560
6:python:kuit560
7:angular:lang99
8:csharp:pwd3456
9:dotnet:onl223
10:

Quadratic Probing:
0:spark:onl345
1:
2:nodejs:cls334
3:hadoop:bgdt564
4:dotnet:onl223
5:json:kuit560
6:python:kuit560
7:angular:lang99
8:csharp:pwd3456

9:
10:

Please choose one of the following commands:

1- Add User
2- Remove User
3- Forgot Password
4- Forgot Username
5- Print Users
6- Exit

>3
>enter use name:
> java

Linear Probing:

Record does not exist

Quadratic probing:

Record does not exist

Please choose one of the following commands:

1- Add User
2- Remove User
3- Forgot Password
4- Forgot Username
5- Print Users
6- Exit

>6
>Bye Bye!

Execution Instructions:
GDB/DDD can be used to debug your code. Make sure to use Valgrind to execute your code to check for any memory leaks/segmentation faults.

EECS 560 Lab – Implementation of Hash Tables (Closed Hashing)

Prof.: Dr. Shontz, GTAs: Chiranjeevi Pippalla, Prashanthi Mallojula

Grading rubric:

- Full grade: The program should execute without any issues with all the options executed and with no memory leaks.
- Points will be taken off for execution errors, such as memory leaks, segmentation/program abort issues and missing handling of invalid cases.
- Programs that are compiled but do not execute will earn in the range of 0 to 50% of the possible points. Your grade will be determined based on the program design and the options implemented in the code.

Submission instructions:

- All files, i.e., the source files and Makefile, should be zipped in a folder.
- Include a ReadMe.txt if your code requires any special instructions to run.
- The naming convention of the folder should be LastName_Lab3.zip (or .tar or .rar or .gz).
- Email it to your respective lab instructor:
  →Tuesday and Thursday (02:30 to 04:20 PM): chiranjeevi.pippalla@ku.edu (Chiru)
  →Tuesday (11:00AM to 12:50 PM): prashanthi.mallojula@ku.edu (Prashanthi)
  →Thursday (11:00AM to 12:50 PM): eecs560f2020@gmail.com (eecs560)
- Make sure the subject line is EECS 560 Lab3.
- Your program should compile and run on the **Linux machines** in **Eaton 1005D using g++.**