CSCE 2110 – Fall 2016

Program 1: Hash Tables

Due: Friday, October 14th (10/14/16) before midnight (11:59 PM)

This assignment will cover Hash Tables that implement chaining to handle collisions. With this assignment, there is some skeleton-code given that will compile. However, you are *not required* to use it. You are welcomed to do your own implementation and use any/all of the code given to you. The code presented to you is to promote a good starting point or pseudo-code to help you move forward on your implementation of a Chaining Hash Table.

Your program will:

1. Implement a Hash Table that solves collisions using: (10 Points)
    a. Chaining (linked lists) (10 Points)

2. Perform proper:
    a. Hash Insert(), (10 Points)
    b. Hash Delete(), (10 Points)
    c. and Hash LookUp(). (10 Points)
        i. Will Print the ID, GPA, and Major to the screen. (10 Points)

3. Provide an improved Hash Function, other than what was provided to you. (10 Points)
    a. You will be required to cite the resource(s) you use to solve this problem.

4. Provide two views of the Hash Table: (10 Points)
    a. A detailed, per Bucket per Node view.
    b. A high-level, Hash Table view

5. Be readable and professional. (10 Points)
    a. Both code and output.

6. Implement a multi-file structure. (10 Points)
    a. Requires the use of a makefile.
    b. Requires the use of a clean function in your make file.

What is required:

- Submitted before the due date.
  - No late assignments will be graded.
  - No Make up is provided.

- C++ Program.
  - Otherwise, will result in a 0.

- Must compile (with no warnings!) on the CSE linux machines.
  - Warnings will be treated as incomplete, thus resulting in a 0.
    - It DOES NOT count if it compiles on your PC. It MUST compile on the CSE machines.

- Must have more than one file.
- Will use a make file to compile the program.
  - Make file must have a "clean" operation.

- Your code MUST be professional and readable:
  - Such as:
    - Comments.
    - Indentation.
    - Proper naming of variables/functions.
    - Implementation of a naming convention.

- Must show that a collision is handled by chaining.
  - Don't make your Hash Table BIGGER than the data you plan to implement.

What is allowed:

- Use of the following libraries:
  - cout
  - endl
  - Strings
  - Linked Lists
  - Iostream

- You may make any alteration that you see fit to the code provided.
- Discussing Hash Functions. (NO CODE SHARING!)
- Getting help on compile errors/warnings.

What is NOT allowed: (Will result in a 0)

- Using any method other than chaining to solve collisions.
- Using any library not authorized above.
- Use of any code that you did not author.

- Use of any code not provided to you by the TAs, Peer Mentors, or myself.
- Use of any g++ flag (other than –c).
- Using a Hash table larger than 20.
- Failure to cite any external (out of class) resources used to complete this assignment.

Here is what your program should be able to do:

((Example Output Provided Later))

1. Insert at least 3 Students into the Hash Table.
    a. Print to the screen any changes made to the Hash Table.
        i. 1 detailed view of each node in each bucket.
        Similar to HashTable::printBucketValues()
        ii. 1 high-level view of the structure of the Hash Table.
        Similar to HashTable::printHistogram()
    b. This should show that your submitted hash function meets the requirements of a good hash function.
        i. I.E.: Doesn't group similar terms near one-another.
        ii. Don't use the function provided; research and submit your own.

2. Remove at least 1 Student from the Hash Table
    a. Print to the screen any changes made to the Hash Table.
        i. 1 detailed view of each node in each bucket.
        Similar to HashTable::printBucketValues()
        ii. 1 high-level view of the structure of the Hash Table.
        Similar to HashTable::printHistogram()

3. Add a total of 25 (include the 2 left from above) students.
    a. Print to the screen any changes made to the Hash Table.
        i. 1 detailed view of each node in each bucket.
        Similar to HashTable::printBucketValues()
        ii. 1 high-level view of the structure of the Hash Table.
        Similar to HashTable::printHistogram()
    b. Your hash table should NOT have 25 or more slots available. Otherwise you fail the ability to show me that your Hash Table handles collisions with chaining…

4. Perform a LookUp on at least 1 student of your choice and put their values on screen.
    a. The ID (int)
    b. Their GPA (double)
    c. Their Major (string)

Example Output

((Here I had already placed 3 values into the Hash Table and then Printed the changes to the screen.))

```
Hash Table Bucket Values:
Bucket 1:
{ }
Bucket 2:
{ 1 }
Bucket 3:
{ 2 }
Bucket 4:
{ 3 }
Bucket 5:
{ }
Bucket 6:
{ }
Bucket 7:
{ }
Bucket 8:
{ }
Bucket 9:
{ }
Bucket 10:
{ }
Bucket 11:
{ }
Bucket 12:
{ }
Bucket 13:
{ }
Bucket 14:
{ }
Bucket 15:
{ }
Bucket 16:
{ }
Bucket 17:
{ }
Bucket 18:
{ }
Bucket 19:
{ }
Bucket 20:
{ }

Hash Table Contains 3 Nodes total
1:
2:       O
3:       O
4:       O
5:
6:
7:
8:
9:
10:
11:
12:
13:
14:
15:
16:
17:
18:
19:
20:
```

((I then was able to perform a Hash Delete on HashKey 1! Then printed the results))

```
Hash Table Bucket Values:
Bucket 1:
{ }
Bucket 2:
{ }
Bucket 3:
{ 2 }
Bucket 4:
{ 3 }
Bucket 5:
{ }
Bucket 6:
{ }
Bucket 7:
{ }
Bucket 8:
{ }
Bucket 9:
{ }
Bucket 10:
{ }
Bucket 11:
{ }
Bucket 12:
{ }
Bucket 13:
{ }
Bucket 14:
{ }
Bucket 15:
{ }
Bucket 16:
{ }
Bucket 17:
{ }
Bucket 18:
{ }
Bucket 19:
{ }
Bucket 20:
{ }


Hash Table Contains 2 Nodes total
1:
2:
3:       O
4:       O
5:
6:
7:
8:
9:
10:
11:
12:
13:
14:
15:
16:
17:
18:
19:
20:
```

((I then put in a total of 25 values and printed the results to the screen))

```
Hash Table Bucket Values:
Bucket 1:
```

```
{ 20 }
Bucket 2:
{ 21 }
Bucket 3:
{ 2, 22 }
Bucket 4:
{ 3, 23 }
Bucket 5:
{ 4, 24 }
Bucket 6:
{ 5, 25 }
Bucket 7:
{ 6, 26 }
Bucket 8:
{ 7 }
Bucket 9:
{ 8 }
Bucket 10:
{ 9 }
Bucket 11:
{ 10 }
Bucket 12:
{ 11 }
Bucket 13:
{ 12 }
Bucket 14:
{ 13 }
Bucket 15:
{ 14 }
Bucket 16:
{ 15 }
Bucket 17:
{ 16 }
Bucket 18:
{ 17 }
Bucket 19:
{ 18 }
Bucket 20:
{ 19 }


Hash Table Contains 25 Nodes total
1:       O
2:       O
3:       O O
4:       O O
5:       O O
6:       O O
7:       O O
8:       O
9:       O
10:      O
11:      O
12:      O
13:      O
14:      O
15:      O
16:      O
17:      O
18:      O
19:      O
20:      O
```

((Here I performed a LookUp on value 19 and returned the results.))

```
Looking up Student 19…
Student 19 FOUND! Printing out Student 19 Data:
   ID: 19
  GPA: 4.0
Major: CS
```