CSCE 221 Cover Page
Programming Assignment #5

First Name: Chase Last Name: McDermott UIN: 524004424

User Name: chasemcd1745 E-mail address: chasemcd1745@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more in the Aggie Honor System Office http://aggiehonor.tamu.edu/

Type of sources	
People	
Web pages (provide URL)	http://www.algolist.net/ Data_structures/Hash_table/Chaining
Printed material	*Programming Principles and Practices* -Bjarne Stroustrup
Other Sources	Dr. Leyk's slides

I certify that I have listed all the sources that I used to develop the solutions/code to the submitted work.

"*On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.*"

Your Name (signature) Chase McDermott Date 4/12/2017

- Description for PA5:
  - This assignment was meant to teach us about how to implement the Hash Table data structure. The assignment is: given a list of students with: name, email, UIN, and grade, (input.csv) parse this file using regular expressions, and store the data in a hash table using the chaining method (vector of lined lists) for collisions. Next, parse a file with even more students (roster.csv), look up their UIN in the hash table, and return their grade. Output the data to a file (output.csv), with all of the data from roster.csv, but with grades for students who were in input.csv.
- Data Structures and Algorithms in assignment:
  - Obviously the main data structure used in this assignment is a hash table. We use the hash table to store student data.
    - We use UIN as the key, and the grade is the data stored.
    - Our hash function is simply *UIN mod table_size*, with our table size being the size of the roster which is 100 students.
  - We also use linked lists to handle collisions.
    - We also use a vector in our hash table class to hold all of the linked lists.
  - Our main algorithm used is our search function which computes the hash function on the uin, looks up the linked list assigned to that hash key, and iterates through the linked list until it finds the desired UIN. This takes O(1) time complexity because on average, our linked lists are size .17 with no linked list having more than one element.
- Input and Output Data:
  - The input data is given in two files: input.csv, and roster.csv. Roster.csv contains information for all 100 students (name, email, UIN), and input.csv contains the same data along with a grade for a smaller number of students who submitted the assignment.
    - I assumed that all students have a UIN starting with 8 and having 8 digits after the 8. I used this in my regular expression to be more exact since all students in roster.csv have a UIN mathcing this, but this could easily be changed to simply 9 digits if needed.
  - The output data is simply the same as roster.csv, except with grades for students who submitted the assignment.
- Testing:
  - I tested my program with the given input data, and also added some students to the roster and to the input.csv along with grades to attempt to test all cases for my program. It works as expected.
- Compiling and Running my Program:
  - I created a Makefile for this program, so you simply run makefile, and then run the program with "./run-main".
- Standard Library Classes:
  - In my main.cpp I included:
    - iostream - for input and output.
    - fstream - for reading and writing to files.
    - string - for easily using strings.
    - regex - for parsing the files using regilar expressions.

- Hash Table Statistics:
  - These are all with a hash table size of 100 (corresponding to roster size).
    - Minimum linked list size: 0.
    - Maximum linked list size: 1.
    - Average linked list size: .17.
      - This is because with a hashtable size of 100 there are many linked lists with size of 0.
  - This does correspond to the expected running time for the hashing search algorithm because the expected running time for search is O(1), and my program computes a search in O(1) time as long as the size of the linked lists are arbitrarily small (in our case all non-empty linked lists are of size 1).

```
int Hashtable::search(int key)
{
        // searches for and returns a value with a matching key

    int newKey = hashFunc(key);
    // returns the grade of the student with the uin searched for
    return htable.at(newKey).search(key);
}


int Linkedlist::search(int uin)
{
        // searches for the node in the linked list who's uin
        matches inputted uin and returns the grade

    Node* curr = head;
    while (curr != nullptr) {
        if (curr->getUin() == uin) return curr->getGrade();
        else curr = curr->next;
    }

    return 0;
}
```

- Conclusion:
  - This was a very good and constructive assignment, that really taught me a lot about the hash table data structure. Implementing it really gave me a deep understanding of it.