Program 2: File Parser

CSCE 2110 – Fall 2016

Due: Friday, November 4th (**11/04/16**)
before midnight (**by 11:59 PM**)

Introduction

In this assignment, you will be adding a file parser functionality to your previous programming assignment: Hash Tables. (Refer to Program 1: Hash Tables). A solution to the previous programming assignment is accessible via blackboard.

To start, you are required to use the hash function that I have given to you *(%hash_length*). DO NOT implement any other hash function.

```
int HashTable::HashFunction(int hashKey)
{
    return (hashKey) % hashLength;
}
```

Second, you should increase the size of your hash table to be **500**.

```
hashTableLength = 500;
```

Third, you should be aware that the test files I have provided to you have a set amount of students. However, you should not use this number to determine when to stop reading from the file. Your program should **read until the end of the file** (Not just to 1000 inputs).

For this assignment, you are allowed to use any existing library, such that you DO NOT use "using namespace std." In other words, do not use the ***entire*** standard template library for your implementation. You can, instead, include the specific libraries you use in your headers or within the code itself. However, be aware that part of the submission requirement is that your program will compile and execute (***WITHOUT WARNINGS!!***) on the CSE Linux Machines.

For example: if you intend on using fscanf, you can reference via

std::fscanf(),
#include <stdio.h>
or any mixture there-of.

<u>Your program will (rubric):</u>

Out of 100%

1. Implement a Hash Table that solves collisions using chaining.

2. (10 %) Read from an input file (.txt) data structure (provided by myself).
   a. Name of the file will be "input.txt". 🡐 ***can be hard-coded***
   b. Format is a comma-separated-value (CSV): ID,GPA,Major
      i. 10941050,3.88,CS

3. (10 %) Perform a Hash Insert() on ALL inputs.

4. (10 %) Parse the file until the END of it.

NOTE: I have provided to you 2 different types of input files. In one, the ID numbers count UP from 1000. In the other, I randomly assigned an 8 digit number to 1000 students. When grading, I will have more than 1000 students.

5. (10 %) Use the correct Hash Function (as above).

6. (10 %) Provide a **detailed**, per Bucket per Node view.
   a. Your Bucket view must now show: ID Number, GPA, and Major.
      i. 10941050, 3.88, CS 🡐 ***With spaces… (must be readable output!)***

7. (10 %) Provide a **high-level**, Hash Table view.

8. (10 %) Be readable and professional.
   a. *Both code and output*.

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

10. (10 %) Should compile into an executable file that has your euid in it. Such that if your EUID is abc1234, your program executable should be named abc1234.
    a. Use the g++ flag –o:
    b. g++ <files> -o abc1234

All points are graded on a binary rubric.

Either you:

1. Meet the requirements fully, **(Receiving credit)**
2. Or do not. **(Receive no credit)**

<u>List of what is required:</u>

- Submitted <u>**before**</u> the due date.
  - No late assignments will be graded.
  - No Make up is provided.
  - Check Blackboard after you submit to ensure that it is there and what you wanted to be there.

- 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 ***<u>DOES NOT</u>*** count if it compiles on your PC. It MUST compile on the CSE machines.

- Must have more than one file.
  - Otherwise will result in points lost (see above).

- Will use a make file to compile the program.
  - Make file must have a "clean" operation.
  - Otherwise will result in points lost (see above).

- Your code MUST be professional and readable:
  - Such as:
    - Comments.
    - Indentation.
    - Proper naming of variables/functions.
    - Implementation of a naming convention.
  - Otherwise will result in points lost (see above).

- Must show that a collision is handled by chaining.
  - **<u>Don't make your Hash Table BIGGER than the data you plan to implement.</u>**
    - *Keep the size under 1000.*
  - Otherwise will result in points lost (see above).

<u>((Future changes to this list can be found in course announcements on Blackboard))</u>

<u>List of what is allowed:</u>

- Use of the following libraries:
    - cout
    - endl
    - Strings
    - Linked Lists
    - Iostream
    - Fscanf
    - Stdio
    - math
- 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. **(NO CODE SHARING!)**

((Future changes to this list can be found in course announcements on Blackboard))

<u>What is NOT allowed:</u>

(Will result in a 0)

- Using any method other than chaining to solve collisions.
- Using any library not authorized above.
    - Be sure to have explicit permission from myself (the instructor) via email or announcement on BB.
- 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 size larger than the expected input (we have to test for chaining).
- Failure to cite any external (out of class) resources used to complete this assignment.

((Future changes to this list can be found in course announcements on Blackboard))

Here is what your program should be able to do:

1. **Read** from a file and insert **all** of its data into your hash table.

2. **Print** to the screen:
   a. 1 detailed view of each node in each bucket.
      i. Similar to HashTable::printBucketValues(), but the following format:
         <ID Number>, <GPA>, <Major>

   b. 1 high-level view of the structure of the Hash Table.
      Similar to HashTable::printHistogram()

Example Output

[[Note: #, #.##, XX should translate to a given ID Number, GPA, and Major Abbreviation]]

```
abc1234@cse01:~/2110/Program 2$ ./abc1234

Opening file: input.txt
Students found and inserted successfully.

Printing Detailed View (Bucket per Bucket) of the Hash Table with inserted
values…:

Hash Table Bucket Values:
Bucket 1:
{
       #, #.##, XX
       #, #.##, XX
}
Bucket 2:
{
       #, #.##, XX
       #, #.##, XX
}
                              .
                              .
                <REDACTED FOR PRESENTATION PURPOSES>
                              .
                              .
Bucket 499:
{
       #, #.##, XX
       #, #.##, XX
}
Bucket 500:
{
       #, #.##, XX
       #, #.##, XX
}

Printing High-Level View of the Hash Table:
Hash Table Contains 1000 Nodes total
1:       O O
2:       O O
                              .
                              .
                <REDACTED FOR PRESENTATION PURPOSES>
                              .
                              .

19:       O O
20:       O O
```