Project Description

The focus of this project is upon using data structures to hold, operate on, and process information. The main goal of this project is to create three separate hash tables, each with their own secondary index in the form of a linked list. Each hash table holds a separate tuple with their own schemes that store unique data. In addition, each hash table uses its own unique hashing function and collision strategy to store data.

The project is divided into four sections. The first section is focused on reading in input and organizing output. This is accomplished by opening an input file that contains the names of three other files, as well as a list of queries. Before handling these queries, the three other files are opened and read, giving the starting entries for all three hash tables. Then the queries in the original input file are processed using regular expressions to find tuples and parse data. This data is used in the other three parts of the project, all of which are the three different hash tables used for each file and scheme: age, disability, and geography.

The main educational goal of this project is to familiarize students with advanced data structures, namely hash tables. Using C++ standard data structures such as map or list are prohibited, directing the student to instead create their own and discover the methods by which a hash table works. In addition, the project familiarizes the student with the use of regular expressions and the regex_search function to parse input.

Major Functionality Piece 3 - Sena Moon

For part 3, I created a source file called age.cpp and a header file named age.h and used main.cpp to create age table and age linked list objects and call age table and age linked list functions. To store age.csv data into my age table, I used a 2d vector of struct. My hash table uses modulo hash function and chaining probing as collision strategy; therefore, I set my table size as static const int and resized my 2d vector to the

variable tableSize to avoid table size changing. When storing tuples into my 2d vector, I used my INSERT function which is defined to take whatever entry is passed into the function, parse it through string stream, use getline to take each tuple into the temporary 1d vector in each attribute's index, push back the ageStruct object into ageTableVector2d at the index of the key, which is calculated through my hash function. I used the age_holder string that Part 1 passed to me in main.cpp to store data into my 2d vector. And for linked list, I used a struct to hold an object and pointer and created a class that contains head pointer and functions.

I declared my INSERT, UPDATE, SELECT, DELETE, DISPLAY, WRITE functions in age.h file and defined them in age.cpp file for my hash table. And I declared and defined addNode, SELECT, and DELETE functions for my linked list. All of my functions are called underneath the regex match if statements for each function in the main. INSERT function does what I explained in the previous paragraph.

My addNode function is a simple function to create the linked list using my ageStruct's object. My UPDATE function takes the entry, parse it, using getline it gets values for temporary geoid, under 5, under 18, over 65 and goes through the 2d vector where my hash table is stored, and looks for the key value that equals to the index given by the query. And then it updates the values in my table to the values that query gave. Then my function prints updated values to the user. Also, if the geoid in my table doesn't match the query's geoid, then my function gives the user the prompt saying it failed to update the query to the table. For DISPLAY, I gave a user prompt that indicates that current output is the display of age table, and formatted the attributes to a user-friendly view. Then, I walked through the vector and found each tuple and set it to the same format as the attributes names. For WRITE function, I created a csv file named out_Age.csv and input the 1st and 2nd lines as same as the age.csv test files. Then, I walked through the vector and input the tuples to the outFile as the same format as other test files.

My SELECT and DELETE functions are both defined and declared in hash table and

linked list. Both functions use the same set up to get geoid from the passed entries. Then, when geoid is not “*”, then both walk through the hash table vector and find the entry and display it to the user for SELECT and deletes and displays what’s been deleted for DELETE to the user. And when geoid is “*”, both functions refer to the linked list and finds entry at the pointer and displays/deletes the entry for the user.