Mason Bone, Christopher Martinez, Summer Hodges, Alex Daughters
CSCE 2110
Project 3

Emergency Storage

4 December 2019

Overview

For this project, there were 4 main parts; one to read in information from user-designated files, one to execute functions for geography, one to execute functions for age, and one to execute functions for disability. For the geography, age, and disability we created a type file with get/set functions corresponding to whatever variable were necessary to store and then a collection file to execute the functions to insert, update, select, delete, display, and write data to a table. This is supposed to represent a system that stores information in the case of an emergency.

Part 1: Mason Bone

For part one of the project, my job was to read in the initial tables and store them. As well as get all the users queries and call the corresponding functions. I did this in two files with one class. The files are called tableInput.cpp and tableInput.h, and the class is called tableInput.

This class has a lot of private vectors used to store initial information. The first three private data stored are the objects to all the table functions. The next two are a vector of strings named tableNames that stores the names of the tables and another vector of strings named tableFiles that contains all the file names for the initial tables. And the final three are matrices that contain the data used for the functions.

There are 5 functions in this class. The first of which is called initialTable. This function first get the users input file and opens an instream. It then gets the names of the files and the names of the tables and stores them in the already declared vectors. It then makes sure the right information goes to the right table. It dies this by using a for loop with if statements and regex. And finally it reads in the functions that will be called later in the program.

The next three functions are called start1, start2, and start3. Each has a parameter of an int value to know which table is needed. These three functions get the file names for the tables and open in streams. They then read in and splice the information and store it in their table

vectors. And finally they call the necessary insert functions for the tables to make sure all information is correctly stored in the proper tables.

The final function is called tableFuncs that stores and splices the functions to be called later in the program. It then sorts through and call the appropriate functions for the correct tables.

Part 2 (Geography): Christopher Martinez

For the geography section of the code, I created 4 files. The file called “geogNode.h” with a corresponding cpp file defines the get and set functions. The information we needed to store in the type file was a string for the key(replan_id) read in from the user-designated files, a string called replan_id, a string called geo_name, a string called geo_stusab, a string called geo_state, a string called geo_county, a string called geo_geoid, a string called population. These strings essentially separate the different sections of the data that needs to be stored. I also created collection files called “geogHash.h” with a corresponding cpp file to define the functions. Also included in this header file is the respective initiator constructor and destructor.

In geogHash.h I declared the functions to insert, update, select, delete, display, and write data to a table. Under private, we have a dynamic array called “table” of type class geogNode and a vector of vectors of type geogNode called “geogList” that holds information in a linked list.

The first hash function defined searches through the bucket using the key. This function employs the mid-square hashing method and returns a number smaller than the size of the table in order to save the data in one of the indices of the array.

The second hash function defined employs the modulos against a number smaller than that of the table size in order to save the data in one of the indices of the array.

For the insert function, which is a void, the parameters are strings for the key, geoName, geoSub, geoState , geoCounty, geoId, geoLevel, population, and name of a table. This function calls the modulos and mid-square hashing functions.

For the update function, which is a void, the parameters are strings for the key, key, geoName, geoSub, geoState , geoCounty, geoId, geoLevel, population, and name of a table. This function updates information for something already in the table.

For the select function, which is a void, the parameters are strings for the key, key, geoName, geoSub, geoState , geoCounty, geoId, geoLevel, population, and name of a table. Based on a user query, the table is searched for a user-designated spot in the table. The search informs the user if the entry has been found or not found.

For the delete function, which is a void, the parameters are a string for the key, key, geoName, geoSub, geoState , geoCounty, geoId, geoLevel, population, and name of a table. A loop searches through the table to locate a designated value in the table and then sets it to NULL when it is found.

For the display function, has no parameters and is a void function. This function outputs all the information in the table using a for loop that repeats until it reaches the end of the table size.

For the write function, has no parameters and is a void function. This function outputs and stores all of the information in the table to a user-designated file name using a for loop that repeats until it reaches the end of the table size.

Part 3 (Age): Summer Hodges

For the age section of the code, I created 4 files. The type file is called “ageNode.h” with a corresponding cpp file to define the get and set functions. The information we needed to store in the type file was a string for the key read in from the user-designated files, a string for the number of people under 5-years-old, a string for the number of people under 18-years-old, and a string for the number of people over 65-years-old. These strings for the numbers are later translated into integers. The collection file is called “ageHash.h” with a corresponding cpp file to define the functions.

In ageHash.cpp we defined the functions to insert, update, select, delete, display, and write data to a table. Under private, we have a dynamic array called “table” of type class ageNode and a vector of vectors of strings called “ageLinkedList” that holds information in a linked list.

The first function defined is the hashing function that returns an integer and the parameter is the key. This function generates and returns a number smaller than the size of the table.

For the insert function, the parameters are a string for the key, a string for people under 5, a string for people under 18, a string for people under 65, and a string for the name of a table and is a void function. This function inserts the values for key, under 5, under 18, and over 65 into the table. This is done using modulus hashing and quadratic probing.

For the update function, the parameters are a string for the key, a string for people under 5, a string for people under 18, a string for people under 65, and a string for the name of a table and is a void function. This function updates information for something already in the table.

For the select function, the parameters are a string for the key, a string for people under 5, a string for people under 18, a string for people under 65, and a string for the name of a table and returns a vector of type ageNode. Based on a user query, the table is searched for a user-designated spot in the table. The search informs the user if the entry has been found or not found.

For the delete function, the parameters are a string for the key, a string for people under 5, a string for people under 18, a string for people under 65, and a string for the name of a table

and is a void function. A loop searches through the table to locate a designated value in the table and then sets it to NULL when it is found. Therefore, it is deleted.

For the display function, has no parameters and is a void function. This function outputs all the information in the table using a for loop that repeats until it reaches the end of the table size.

For the write function, has no parameters and is a void function. This function outputs and stores all of the information in the table to a user-designated file name using a for loop that repeats until it reaches the end of the table size.

Part 4 (Disability): Alex Daughters

For the disability section of the project we had disaHash.cpp/.h and disaNode.cpp/.h files. In disaNode.h I have defined a string for the key called geo_name, integer called hearing_disability, integer called vision_disability, integer called cognition_disability, integer called ambulatory_disability, integer called self_care_disability, integer called independent_living_disability in the private section along with the get and set function for all of these in the public. The collection file for these is disaHash.h and its corresponding .cpp.

In disaHash.cpp we defined the functions to insert, update, select, delete, display, and write data to a table. Under private, we have a dynamic array called “table” of type class disaNode and a vector of vectors of strings called disaLinkedList that holds information in a linked list.

The first function defined is the hashing function that returns an integer and the parameter is the key. This function generates and returns a number smaller than the size of the table.

For the insert function, the parameters are a string for the key, a string for hearing_disability, a string for vision_disability, a string for cognition_disability, a string for ambulatory_disability, a string for self_care_disability, a string for independent_living_disability, and a string for the name of a table and is a void function. This function inserts the values for key and all for the variables listed above into the table. This is done using multiplicative string hashing and linear probing.

For the update function, the parameters are a string for the key, a string for hearing_disability, a string for vision_disability, a string for cognition_disability, a string for ambulatory_disability, a string for self_care_disability, a string for independent_living_disability, and a string for the name of a table and is a void function. This function updates information for something already in the table.

For the select function, the parameters are a string for the key, a string for hearing_disability, a string for vision_disability, a string for cognition_disability, a string for ambulatory_disability, a string for self_care_disability, a string for independent_living_disability, and a string for the name of a table and returns a vector of type disaNode. Based on a user

query, the table is searched for a user-designated spot in the table. The search informs the user if the entry has been found or not found.

For the delete function, the parameters are a string for the key, a string for hearing_disability, a string for vision_disability, a string for cognition_disability, a string for ambulatory_disability, a string for self_care_disability, a string for independent_living_disability, and a string for the name of a table and is a void function. A loop searches through the table to locate a designated value in the table and then sets it to NULL when it is found. Therefore, it is deleted.

For the display function, has no parameters and is a void function. This function outputs all the information in the table using a for loop that repeats until it reaches the end of the table size.

For the write function, has no parameters and is a void function. This function outputs and stores all of the information in the table to a user-designated file name using a for loop that repeats until it reaches the end of the table size.