Project: Student Gradebook

Write a program that allows the user create gradebook for students, that includes both a grade and a student’s ID number. The program should load in a list of students from a text file, then present the user with options to add a new student, delete a student from the list, change a student’s grade, and display all students. When the program closes, all changes to the list are saved onto the text file so it can be accessed again.

**Example of output**:

|  |  |
| --- | --- |
| Welcome to Student Gradebook  1. Insert New Student.  2. Delete an existing Student.  3. Change a student's grade.  4. Display a list of all students  5. Exit and save.  Please make your Selection: 4  --------------screen clears-------------------  Displaying List of students:  Student Name: Clark Kent  Student ID #: 789  Student Grade: C  Student Name: Barry Allen  Student ID #: 456  Student Grade: B  Student Name: Bruce Wayne  Student ID #: 123  Student Grade: A  Press any key to continue . . . | Welcome to Student Gradebook  1. Insert New Student.  2. Delete an existing Student.  3. Change a student's grade.  4. Display a list of all students  5. Exit and save.  Please make your Selection: 1  --------------screen clears-------------------  Insert student info :  Enter Student's First Name:  Arthur  Enter Student's Last Name:  Curry  Enter Student's ID:  963  Enter Student's Grade:  A  Press any key to continue . . . |

Each **Student** *struct* is held in an array called **Students**, which is a part of the **StudentList** *class*. The **StudentList** *class*, has several functions that can add to, remove, or modify items in the list.

The **Student** *struct* contains the following data members.

string firstName;

string lastName;

char grade;

int id;

The int id, data member will be used as the key with which items are sorted in escalating order within the list.

The **StudentList** *class* contains an *int* called **length**, the **Students** *array*, and a *const int* called **MAX\_ITEMS**, and the following functions used to manipulate an array of **Student** structs and their contents.

**StudentList();**

Constructor, initializes the **length** variable as 0

**~StudentList();**

Destructor, because this is an array based list, the destructor automatically deletes all data in memory when the class is destroyed. Does not need any special instructions.

**void insertStudent();**

First uses **isFull()** to make sure there is space for another item. Then uses **binarySearchMod()** to search the list and make sure there are no duplicate keys being created. If **binarySearchMod()** fails, the function proceeds to find a location for a new item using **binarySearchInsert()**, once an appropriate location for the new item is found, all items after this location index are moved down the list and the new item’s information is copied into the newly created space and **length** is incremented.

**void deleteStudent();**

Prompts user to enter the ID of a **Student** that is to be deleted from the list. Once the user enters a valid **ID** number, **binarySearchMod()** is called to find the matching index, once found, all items following this index are moved up to replace the slot that was to be deleted. Lastly, length is decremented by 1.

**void changeStudent();**

Prompts user to enter the ID of a **Student** whose **grade** is going to be changed. Once the user enters a valid **ID** number, **binarySearchMod()** is called to find the matching index, once found the **grade** variable in this index is updated.

**bool isEmpty();**

Checks to see if list is empty by checking if the **length** of the list is 0.

**bool isFull();**

Checks to see if the list is full by comparing the **length** of the list to the maximum amount of items the array can hold as denoted by **MAX\_ITEMS**.

**int binarySearchInsert();**

This function uses binary search to find a location for a new item to be added to the list. This function will search until it finds a **first** and **last** index that are adjacent, where the new item being added can fit between the **first** and **last** index. Except when the item is the greatest or smallest in the list this function always returns the **last** index so that the **insertStudent()**function can push the **last** and all subsequent indexes back by 1 place.

**int binarySearchMod();**

This function uses binary search to find an existing item in the list that is to be modified or deleted. Always returns **midpoint**.

**void displayList();**

Uses a while loop to read through all the variables in each index of the **Students** arrayand display them until the end of the list is reached.

**void loadFromFile();**

Creates a **dataFile** *fstream* object in order to open **StudentData.txt** for input. Checks for any errors in opening the text file. Creates a while loop that runs until the end of the file is found. With each pass a new index is accessed, data is copied from the text file to the index, and **length** is incremented.

**void saveToFile();**

Creates a **dataFile** *fstream* object in order to open **StudentData.txt** for output. Checks for any errors in opening the text file. Creates a do while loop that runs until the end of the list is found. Uses a while loop to move through the array and copies data from every **Student** index to the text file until the end of the list is reached.