CSCE 1030: Project 3
Due: 11:59 PM on Monday, December 2nd, 2024

PROGRAM DESCRIPTION

In this project, you have to write a C++ program to keep track of grades of students using structures and files.

You are provided a data file named **student.dat**. Open the file to view it. Keep a backup of this file all the time since you will be editing this file in the program and may lose the content.
The file has multiple rows—each row represents a student. The data items are in order: last name, first name including any middle names, student ID (8 digits long), number of tests the student took, and the scores in those tests. Note that each item is separated by commas (‘,’), even the last score has a comma after it. **Do not change this format.**

The teacher has given 5 tests to each student, but not all students took all 5 tests. The minimum test score is dropped.

You are also provided a .cpp file: **getNumber.cpp**. It has a single function **getNumber** which gets the number of student records in the data file. You will need to call this function to create your dynamic arrays.

PROGRAM REQUIREMENTS

1. Declare and initialize the following global parameters.
    - A global integer constant of type integer to store the number of tests given by the teacher and initialize it to 5.
    - An enumeration constant with values **Add, Remove, Display, Search, Results,** and **Quit**, and assign integer values 1, 2, 3, 4, 5, and 6 to the data items, respectively. They represent menu choice presented to the user.
    - A structure named **Student** with the following members.
        - Name of the student **(DO NOT USE SEPARATE VARIABLES FOR FIRST NAME AND LAST NAME)**
        - Student ID of the student
        - Number of tests taken by the student
        - An integer pointer for a dynamic array to store the test scores for the student
        - Average of the test scores
2. In your **main** function:
    - Display a menu for the user (SEE SAMPLE OUTPUT) that provides the user the following choices.
        - Add a new student record

    - Remove an existing student record
    - Display all records
    - Search for a particular student record using student ID
    - Export the results to a disk file
    - Quit the program
- Using a suitable message, ask the user to make the menu choice using an integer variable.
- Based on the value entered by the user for menu choice, design a switch-case block to implement the following requirements.
    - You must use the enumeration constants to set up your cases.
    - You must use a variable of your enumeration constant type for switching control.
    - If the user chooses to add a student record
        - Call the function **add_Student**
    - If the user chooses to remove a student record
        - Using a suitable message, ask the user the student ID of the student to remove.
        - Call the function **remove_Student** and pass the student ID entered by the user.
    - If the user chooses to display records
        - Call the function **display**
    - If the user chooses to search for a student
        - Using a suitable message, ask the user the student ID of the student to search for.
        - Call the function **search** and pass the student ID entered by the user.
    - If the user chooses to export results to a disk file.
        - Call the function **exportResults**.
    - If the user chooses to quit the program.
        - Exit the program with a suitable message.
    - If the user chooses anything else such as invalid number, character or string, then handle the error situation.
        - Notify the user of an incorrect choice.
        - Using a suitable loop, ask the user for the choice again.
        - Your program must keep on looping until the user enters the correct choice

3. Write a function named **add_Student**. Inside the function:
    - Using a suitable message, prompt the user for the first name, the last name, the student ID, the number of tests taken and the test scores of a student.
    - HINT: The test scores are stored in a dynamic array and the size of the array is the number of tests. Allocate memory accordingly.
    - Open the data file **student.dat** and write the data of the student to the file maintaining the given format.

- You must use the structure members to write the data to the file.
- This function will be called by the **main** function.

4. Write a function named **remove_Student.** It has a parameter for the student ID of the student to be removed.
    - Inside this function, call function **getNumber** to get the current number of students in the data file.
    - Create a dynamic array of the structure type **Student**. Use the number of students in the data file for the size of this dynamic array.
    - Open **student.dat** file for reading.
    - In a loop, read one line of the file at a time and store appropriate data in the dynamic array.
        - While reading check if the student ID being read from the file matches the student ID of the student to be removed.
        - If you find match, use a Boolean flag to store that you have found a match.
        - Copy the entire file to your dynamic array whether or not you find a match.
    - Close the file.
    - If you have found a match while reading:
        - Open **student.dat** file for writing.
        - Write the contents of the dynamic array into the file, only if the student ID does not match the student ID to be removed.
    - If you have not found a match while reading:
        - Notify the user with a suitable message to indicate the student does not exist.
    - Close the file.
    - This function will be called by the **main** function.

5. Write a function named **display**. Inside the function,
    - Open the data file for reading.
    - Using the **getNumber** function and a pointer of type Student, create a dynamic array to read the contents of the file into the dynamic array using a suitable loop.
    - Using a second loop, display the contents of the array in the following format:
        - Allocate 30 spaces for the entire name
        - Allocate 15 spaces for the student ID
        - Allocate 5 spaces each for the test score
        - You don't need to display the number of tests
    - Close the file
    - This function will be called by the **main** function.

6. Write a function named **search**. This function accepts the student ID of the student to search. Inside the function:
    - Open the data file for reading.

- Declare a pointer of type Student. You DO NOT need an array here.
- Using a suitable loop, read each line of the file and store the appropriate data into the appropriate members of the structure pointer.
- Check if the student ID being read from the file matches the student ID to search. If there is a match:
    - Set a Boolean flag to true to indicate match has been found.
    - Display the data corresponding data of the matched student using the following format:
        - Allocate 30 spaces for the entire name
        - Allocate 15 spaces for the student ID
        - Allocate 5 spaces each for the test score
        - You don't need to display the number of tests
- If the Boolean is false i.e. no match is found, display an appropriate message for the user,
- This function will be called by the **main** function.

7. Write a function named **exportResults**. Inside the function:
    - Open a data file named **averages.dat** for writing. This file will contain the averages of the test scores for each student along with the student ID.
    - Open **student.dat** for reading.
    - Create a dynamic array of type Students using a suitable pointer and calling the function **getNumber** to store the contents of the data file **student.dat.**
    - In a loop, read the contents of the file and store it in the dynamic array.
    - In a second loop, process each student at a time to compute the average as follows:
        - Compute the sum of the test scores.
        - Note that the minimum score is dropped. Call the function named **findMinimum (see Step 8)** to find the minimum score and subtract it from the total score.
        - Divide the sum by an appropriate integer to compute the average.
        - Write the student ID and the average to the **averages.dat** file. The average column should have one digit after the decimal point.
    - Close both files.
    - This function will be called by the **main** function.

8. Write a function named **findMinimum** that accepts an array of integers and the size of the array as parameters. This function computes the minimum out of the test scores and returns the score.
    - If a student has taken less than 5 tests, the minimum score is 0.
    - If a student has taken all 5 tests, the minimum score is the minimum of the 5 scores in the array.
    - This function will be called by the **exportResults** function.

9. Now that you've written and tested your program, make the following changes.
    - Create a file named **euidProject3_main.cpp** and copy the main function to this file.

- Create a file named **euidProject3_func.cpp** and copy all the functions (other than the main function) to this file.
- Create a file named **euidProject3_header.h** and copy all the global parameters, include directives, structure definition, function declarations to the header file. Make sure you include the header file in the .cpp files.
- You need to submit these three files to Canvas, where euid should be replaced by your EUID.
- Compile the files together (including **getNumber.cpp**) and make sure it works correctly.

10. You must follow all the good programming practices of file I/O as well as dynamic memory programming throughout your code.

11. Your program must have exactly these listed functions. If you miss a function, you won't get points for that. If write an extra function, you won't get points for that either. The functions are:
    - **getNumber**
    - **findMinimum**
    - **add_Student**
    - **remove_Student**
    - **display**
    - **search**
    - **exportResults**
    - **main**

12. Your program will be graded based largely on whether it works correctly on the CSE machines (e.g., cse01, cse02, …, cse06), so you should make sure that your program compiles and runs on a CSE machine.

DESIGN (ALGORITHM):

On a piece of paper (or word processor), write down the algorithm, or sequence of steps, that you will use to solve the problem. You may think of this as a "recipe" for someone else to follow. Continue to refine your "recipe" until it is clear and deterministically solves the problem. Be sure to include the steps for prompting for input, performing calculations, and displaying output.
You should attempt to solve the problem by hand first (using a calculator as needed) to work out what the answer should be for a few sets of inputs.
Type these steps and calculations into a document (i.e., Word, text, or PDF) that will be submitted along with your source code. Note that if you do any work by hand, images (such as pictures) may be used, but they must be clear and easily readable. This document shall contain both the algorithm and any supporting hand calculations you used in verifying your results.

Here are some sample outputs to help you write the code. The items in bold are entered by the user on the terminal as input data.

SAMPLE OUTPUT 1:

```
$ ./a.out
1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:20
Incorrect choice. Please enter again.

1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:e
Incorrect choice. Please enter again.

1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:asdf
Incorrect choice. Please enter again.

1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:6
Bye!!!
```

SAMPLE OUTPUT 2:

```
$ ./a.out
1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:3
            Smith,John Stevens       12456214   99   98   96   92   91
                 Johnson,Chris       11058975   84   83   78   91
                     abcd,abcd       11114444  100  100  100   98

1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:4
Enter ID of student to search:11110000
Student does not exist.

1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:4
Enter ID of student to search:11058975

Johnson,Chris       11058975   84    83    78    91

1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:6
Bye!!!
```

SAMPLE OUTPUT 3:

```
$ ./a.out

1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:1
Enter last name of the student:newlast
Enter first name of the student:newfirst
Enter student ID:12121212
How many tests did this student take?4
Enter score #1:100
Enter score #2:85
Enter score #3:87
Enter score #4:94

1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:6
Bye!!!

$ cat student.dat
Smith,John Stevens,12456214,5,99,98,96,92,91,
Johnson,Chris,11058975,4,84,83,78,91,
abcd,abcd,11114444,4,100,100,100,98,
newlast,newfirst,12121212,4,100,85,87,94,
```

SAMPLE OUTPUT 4:

```
$ ./a.out
1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:2
Enter ID of student to remove:10101010
Student does not exist.

1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:2
Enter ID of student to remove:11114444

1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:6
Bye!!!

$ cat student.dat
Smith,John Stevens,12456214,5,99,98,96,92,91,
Johnson,Chris,11058975,4,84,83,78,91,
newlast,newfirst,12121212,4,100,85,87,94,
```

SAMPLE OUTPUT 5:

```
$ ./a.out
1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:5
Results exported to file.

1.Add
2.Remove
3.Display
4.Search
5.Results
6.Quit
Enter choice:6
Bye!!!

$ cat averages.dat
12456214        96.2
11058975        84.0
12121212        91.5
```

TESTING:

Test your program to check that it operates as desired with a variety of inputs. Then, compare the answers your code gives with the ones you get from hand calculations.

SUBMISSION:

- Your program will be graded based largely upon whether it works correctly on the CSE machines, so you should make sure your program compiles and runs on the CSE machines.

- Your program will also be graded based upon your program style. This means that you should use comments (as directed), meaningful variable names, and a consistent indentation style as recommended in the textbook and in class.

- We will be using an electronic homework submission on Canvas to make sure that all students hand in their programming projects on time. You will submit four items -- (1) the two .cpp code files, (2) the header file, and (3) the algorithm design document -- to the **Project 3** dropbox on Canvas by the due date and time.

- Projects are meant to be problem-solving exercises and are designed to help you practice your coding on larger projects with various pieces of functionality.

- If you choose to work alone, the coding should be primarily your sole work.

- If you choose to work in a group, you are allowed to get assistance from group members when working on these assignments. However, each student is required to report the name(s) of the students they worked with on the assignment. The maximum size of the group is 4 students, you can of course form a smaller group.

- Cheating for these assignments is defined as copying from a fellow student outside your working group, or copying from the web, or any other sources outside the class. You should not copy someone else's code or let a classmate examine your code if you have not identified as working in a group for your homework.

- As a safety precaution, do not edit your program (using vim or nano) after you have submitted your program where you might accidentally re-save the program, causing the timestamp on your file to be later than the due date. If you want to look (or work on it) after submitting, make a copy of your submission and work on that copy. Should there be any issues with your submission, this timestamp on your code on the CSE machines will be used to validate when the program is completed.