CS 202 – Assignment #8

Purpose: Learn linked lists operations like insertion, deletion, merging, sorting etc.
Points: 100

Why we need Linked Lists in C++?
A linked list is a linear dynamic data structure to store data items. Linked lists are a good choice, if the size is unknown. You can insert the elements at any positions like start (head), end (tail) or at any position you need and also dynamically increasing the size of linked list. Linked lists are powerful, especially when you want to add or remove the elements in any position other than head or tail. Linked lists can be more memory-efficient than arrays, especially when dealing with data structures of varying sizes. Each node in a linked list only stores the necessary data and a pointer to the next node, rather than requiring contiguous memory allocation. Certain operations, such as splitting, merging, or reversing a list, can be performed efficiently on linked lists compared to arrays.

Assignment:
In this assignment you will learn to use singly linked lists. You have to code the implementation file that has multiple functions. This assignment includes various operations on student data, such as reading data from a file, inserting sample data, sorting the data based on different criteria, taking user inputs, and updating the data file.

The main() function serves as the entry point for the program. The program reads student data from a file named "studentsData.txt" and inserts it into a linked list using the insertStudentFromFile() function of the CalculateGrade class. The program also includes some hard-coded sample student data, which is inserted into the linked list using the insertStudentFromSample() function of the CalculateGrade class.

The program presents a menu to the user, offering the following options:
1. Sort data from file
2. Sort sample data
3. Take 3 user inputs and sort
4. Insert a student to "studentsData.txt"
Based on the user's choice, the program calls the corresponding function in the CalculateGrade class to perform the requested operation.

If the user selects the third option (Take 3 user inputs and sort), the program prompts the user to enter the details of 3 students, including name, age, midterm1 score, midterm2 score, and final exam score.
The entered student data is then inserted into the linked list using the insertStudentFromSample() function.

If the user selects the fourth option (Insert a student to "studentsData.txt"), the program calls the updateStudentToFile() function of the CalculateGrade class to update the student data in the file.

After the data is entered, the program sorts the data based on the user's selected sorting criteria.
For the sorting options (1, 2, and 3), the program prompts the user to select the sorting criteria:
1. Sort by age
2. Sort by grade (A to F)
3. Sort by grade (low to high) with age as the secondary priority (youngest age at the top)
The program then calls the appropriate sorting function in the CalculateGrade class based on the user's selection.

After the user's chosen operation is completed, the program calls the computeAndPrintGrades() function of the CalculateGrade class to compute and print the grades for all the students. The main purpose of this program is to demonstrate the sorting and management of student data, with the ability to read data from a file, insert new data, and sort the data based on various criteria. The program utilizes the CalculateGrade class, to handle the underlying data structures and algorithms for these operations.

Student UML:

Student
+ name: string
+ age: int
+ midterm1: int
+ midterm2: int
+ finalExam: int
+ grade: char
+ next: Student *

struct Student:

The **Student** class is a data structure that holds information about a student, including their name, age, midterm exam scores, final exam score, and their letter grade. It also includes a pointer to the next **Student** object, which allows the class to be used as a node in a linked list.

Member Variables Descriptions:

+string name: A string that stores the name of the student
+int age: An integer that stores the age of the student
+int midterm1: An integer that stores the score of the first midterm exam for the student
+int midterm2: An integer that stores the score of the second midterm exam for the student
+int finalExam: An integer that stores the score of the final exam for the student
+char grade: A character that stores the letter grade of the student, which is determined by their exam scores.
+Student *next: A pointer to a Student object that represents the next student in a linked list

Global functions

```
bool sortByAge(const Student* a, const Student* b);
```

This function is a comparison function used to sort the students in ascending order based on their age. It takes two Student* pointers as arguments and returns true if the age of the first student (a) is greater than the age of the second student (b). This will result in the students being sorted in descending order of age.

```
bool sortByGrade(const Student* a, const Student* b);
```

This function is a comparison function used to sort the students in ascending order based on their grade. It takes two Student* pointers as arguments and returns true if the grade of the first student (a) is less than the grade of the second student (b). This will result in the students being sorted in ascending order of grade.

CalculateGrade UML:

CalculateGrade
-headFromFile: Student *
-tailFromFile: Student *
-headFromSample: Student *
-tailFromSample: Student *
-insertStudent(head: Student* &, tail: Student* &, newStudent: Student * &): void
+CalculateGrade()
+ calculateGrade(student: Student *): void
+ insertStudentFromFile(name: std::string, age: int, midterm1: int, midterm2: int, finalExam: int): void
+ insertStudentFromSample(name: std::string, age: int, midterm1: int, midterm2: int, finalExam: int): void
+ printStudents(): void
+ sortDataFromFile(sortCriteria: std::string): void
+ sortDataFromSample(sortCriteria: std::string): void
+ updateStudentToFile(): void
+ computeAndPrintGrades(): void

Member Variables Descriptions:

-Student *headFromFile: A pointer to the head of the linked list of students read from a file
-Student *tailFromFile: AA pointer to the tail of the linked list of students read from a file
-Student *headFromSample: A pointer to the head of the linked list of students added from a sample
-Student *tailFromSample: A pointer to the tail of the linked list of students added from a sample

Member Functions Descriptions:

`CalculateGrade();`
The constructor initializes the member variables headFromFile, tailFromFile, headFromSample, and tailFromSample to nullptr. This indicates that the linked lists for the file-based and sample-based students are initially empty.

```
void calculateGrade(Student* student);
```
This function calculates the letter grade for a given Student object based on their exam scores. It first calculates the total score as the average of the student's midterm 1 score, midterm 2 score, and final exam score.

It then sets the grade member variable of the Student object based on the following criteria:

If the total score is greater than or equal to 90, the grade is set to 'A'.

If the total score is greater than or equal to 80 and less than 90, the grade is set to 'B'.

If the total score is greater than or equal to 70 and less than 80, the grade is set to 'C'.

If the total score is less than 70, the grade is set to 'F'.

```
void insertStudent(Student*&head, Student*&tail, Student*newStudent);
```
This function is responsible for inserting a new Student object into a linked list.

It takes three parameters:

head: a reference to the pointer pointing to the head of the linked list.

tail: a reference to the pointer pointing to the tail of the linked list.

newStudent: the Student object to be inserted.

If the linked list is empty (i.e., head is nullptr), it sets both head and tail to the newStudent object. If the linked list is not empty, it appends the newStudent object to the end of the list by setting the next pointer of the current tail to the newStudent object and then updating the tail pointer to point to the newStudent object. Finally, it sets the next pointer of the newStudent object to nullptr to indicate that it is the new tail of the list.

```
void insertStudentFromFile(std::string name, int age, int midterm1, int midterm2, int finalExam);
```

This function is responsible for creating a new Student object with the given parameters and inserting it into the linked list associated with the file-based students (headFromFile and tailFromFile).

It first creates a new Student object with the provided name, age, midterm1, midterm2, and finalExam values, and sets the grade member to '0' and the next pointer to nullptr. It then calls the calculateGrade function to compute the letter grade for the new Student object. Finally, it calls the insertStudent function, passing the headFromFile, tailFromFile, and the new Student object as arguments, to insert the new Student object into the file-based linked list.

```
void insertStudentFromSample(std::string name, int age, int midterm1, int midterm2, int finalExam);
```

This function is responsible for creating a new Student object with the given parameters and inserting it into the linked list associated with the sample-based students (headFromSample and tailFromSample).

It first creates a new Student object with the provided name, age, midterm1, midterm2, and finalExam values, and sets the grade member to '0' and the next pointer to nullptr. It then calls the calculateGrade function to compute the letter grade for the new Student object. Finally, it calls the insertStudent function, passing the headFromSample, tailFromSample, and the new Student object as arguments, to insert the new Student object into the sample-based linked list.

Note: The insertStudent function is a helper function that handles the actual insertion of the Student objects into the linked lists.

```
void printStudents();
```
This function is responsible for printing the details of all the students from two different sources- the file and the sample data. It first prints a header "Students from File:" and then iterates through the linked list of students from the file (headFromFile). For each student, it prints the details: student number, name, age, midterm 1 score, midterm 2 score, final exam score, total score, and grade.

After printing the students from the file, it prints a header "Students from Sample:" and then iterates through the linked list of students from the sample data (headFromSample). For each student from the sample data, it prints the same details as it did for the students from the file.

Note: It prints on console only.

```
void sortDataFromFile(std::string sortCriteria);
```
This function is responsible for sorting the students from the file based on a specified criterion and writing the sorted data to a file named "sortedData.txt". It creates a vector of Student* pointers (students) and populates it with the students from the headFromFile linked list.

Depending on the sortCriteria parameter, it sorts the students vector using the appropriate comparison function. If sortCriteria is "age", it sorts the students in descending order of age using the sortByAge function. If sortCriteria is "grade", it sorts the students in ascending order of grade using the sortByGrade function. If sortCriteria is neither "age" nor "grade", it sorts the students first by grade in descending order, then by age in ascending order, and finally by name in ascending order.

After sorting the students, it opens a file named "sortedData.txt" and writes the details of each student to the file, including the student number, name, age, midterm 1 score, midterm 2 score, final exam score, total score, and grade. Finally, it prints a message indicating that the result has been written to the "sortedData.txt" file.

```
void sortDataFromSample(std::string sortCriteria);
```
This function is responsible for sorting the student data from a sample and writing the sorted data to a file named "sortedSampleData.txt". It first creates a vector of Student* pointers called students and populates it with the student data from the headFromSample pointer.

The function then sorts the students vector based on the provided sortCriteria. If sortCriteria is "age", it sorts the vector in ascending order of the students' ages using the sortByAge comparison function. If sortCriteria is "grade", it sorts the vector in descending order of the students' grades using the sortByGrade comparison function. If sortCriteria is neither "age" nor "grade", it sorts the vector in descending order of the students' grades, and in ascending order of the students' ages, and then in ascending order of the students' names.

After sorting the students, it opens a file named "sortedSampleData.txt" and writes the details of each student to the file, including the student number, name, age, midterm 1 score, midterm 2 score, final exam score, total score, and grade. Finally, it prints a message indicating that the result has been written to the "sortedSampleData.txt" file.

```
void computeAndPrintGrades();
```
This function is responsible for computing and printing the grades for all students in the linked list starting from the headFromFile pointer only.

It creates an output file named "studentsData_OutputFile.txt" and writes the student data to the file. The function iterates through the linked list starting from the headFromFile pointer and writes the student data to the output file. After processing all the students, the function closes the output file.

Note: Writes the data to the file only.

```
void updateStudentToFile();
```

This function is responsible for updating the student data in the "studentsData.txt" file. This function serves as a way for the user to add new student data to the existing data stored in the "studentsData.txt" file. It first prompts the user to enter the following student details like Name, Age, Midterm1 score, Midterm2 score, Final Exam score. After the user has entered the student details, the function calls the insertStudentFromFile() function to insert the new student data into the linked list.

The function then opens the "studentsData.txt" file and writes the student details to the file. After writing the student details to the file, the function closes the file. Finally, the function prints a message to the console indicating that the student details have been updated in the "studentsData.txt" file.

Note: The new data must be appended to the existing/old studentsData.txt file, not overwritten.

```
~CalculateGrade();
```

The purpose of this destructor function is to ensure that the memory used by the CalculateGrade object is properly cleaned up when the object is destroyed. This is important to prevent memory leaks, which can lead to increased memory usage and potential program crashes.

The function iterates through the two linked lists, one at a time, and deletes each node using the delete operator. This process ensures that all the dynamically allocated memory used by the Student objects in the linked lists is properly released, preventing any memory leaks. Overall, this destructor function is a crucial part of the CalculateGrade class, as it ensures that the resources used by the class are properly cleaned up when the object is no longer needed.

Note: Use two while loops, one for each linked list- headFromFile, headFromSample.

Provided files:

CalculateGrade.h
main.cpp
studentsData.txt
make

CS202_Assignment8.pdf
sample_output_Ast8.pdf

You have to complete calculateGradeImp.cpp file only. The function prototypes are in header file. Do not make any changes to the header file as well as main.cpp file.

Files to Upload/Submit:

CalculateGradeImp.cpp

To compile: make

To run and check memory leaks:

```
valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --show-leak-kinds=all --track-origins=yes --track-fds=yes ./main
```

If you want save the output/report of a Valgrind to a text file, then use the following:

```
valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --show-leak-kinds=all –track-origins=yes ./main
```

Sample_Output:

Please check the sample output, which is provided in a separate file. Your program should match the sample output. There are 10 test cases for this assignment. You're going to run the code 10 times, and the code generate output files and print details on console based on user selection and inputs. The output on console is from printStudents(). This function prints the data in the order listed in file and also prints the sample data that is hardcoded in main(). Please check the sample output for more details.

Note: I know the sample output provided may seem quite lengthy and intimidating at first glance. However, I want to assure you that the majority of the output is very similar in structure and content. The reason the total number of pages appears to be so high is that the sample covers a wide range of examples and edge cases to ensure we're thoroughly testing the application. Don't let the sample size intimidate you – think of it as a valuable resource to help you become a better problem-solver and programmer.

Menu:
1. Sort data from file
2. Sort sample data
3. Take 3 user inputs and sort
4. Insert a student to studentsData.txt

3. Take 3 user inputs and sort (Use the following data for this option)

Name	Age	midterm1	midterm2	finalExam
Linda	33	98	97	100
John	30	25	50	45
Tom	25	75	85	89

4. Insert a student to studentsData.txt (Use the following data for this option)

Name	Age	midterm1	midterm2	finalExam
Simon	35	100	100	100