CMPT 135 – FIC 202301 - Assignment 3

Due Date and Time: Sunday 2 April 2023 at 11:55 PM

Instructor: Dr. Yonas T. Weldeselassie (Ph.D.)

Introduction

In this assignment we will design a school management system in order to handle the typical operations of a school administrative system. We will design the school management system using object oriented programming paradigm that will also borrow ideas from relational databases in order to manage memory efficiently. The purpose of the assignment is to apply everything we have learned in CMPT 135 into action in one complete project.

Restrictions

In this assignment, you are **NOT** allowed to use any C++ STL library. You will be given the necessary include directives and namespaces in the starter code file provided together with this assignment; and you are **NOT** allowed to add any include directive or namespace to the project. You are also required to use only the programming methodologies discussed in the course; don't copy and paste some unnecessary complicated C++ code fragments from the Internet. Last but not least, you are **NOT** allowed to change any class design, member variable, or function signature from the code provided to you. You are also **NOT** allowed to add or remove any member or friend function provided for the classes in the starter code file.

IDE

This assignment C++ project was developed and tested under Microsoft Visual C++ 2010 Express. Your submission will also be tested under the same IDE. Thus you are strongly advised to use the same IDE for your work. Of course you may use any other IDE. However please make sure you don't have heap corruption or index out of bound errors which are very well caught in Microsoft Visual C++ 2010 Express but not as good in other IDEs. The test program will also use a fixed seed for the random number generator so that you will generate the same random numbers as in the sample run output and get the same output on the same IDE.

Please note that the program output may not fit on your console output window and you may not be able to see the whole output in your console output window. In this case, I advise you either you use break points (do research about this) or insert system pause statements in the test main program in order to see partial outputs at a time.

Asserting Conditions

In this assignment you will be asked to assert conditions in different parts of the project. In order to be able to do so, you are provided the `cassert` include directive. Therefore you are required to use the assert statement in order to test the validity of a specified condition and if the condition is not satisfied then the assert statement should halt (stop) execution your program.

Submission

You are required to submit one source code file (that is .cpp file) that must contain all your class declarations and definitions and the test main program. You must submit through Moodle before the due date and time. No email submission will be accepted.

Submission Due Date and Time

The due date and time to submit your work through Moodle is **Sunday 2 April 2023 at 11:55PM**.

Problem Statement: The School Management System (Database)

Our aim is to design a school management system as a database system where the system will enable us to

- Add a new course to the system,
- Remove a course from the system,
- Search for a specific course in the system;
- Register a new student to the system,
- Search for a specific student in the system,
- Remove a student from the system;
- Enroll a student to a course,
- Withdraw a student from a course,
- Assign a student a letter grade for a course,
- Compute a student's GPA,
- Compute the top student in the system;
- Print the information stored in the system, and
- Etc.

In order to guide you through the design process, we will build the system step by step as described below.

Step 1

You are given the declaration of a **`SmarterArray`** class template in the Starter Code text file. You are also given the definition of its default constructor and overloaded output stream operator functions. Define all the remaining member functions. The class declaration is heavily commented to help you understand what you need to do.

It is your responsibility to test the correctness of your class definition. To do so, I advise you to write a small test program that tests each member and friend functions of the class before you continue to the next section.

Step 2

You are given the declaration of a **`Map`** class template in the Starter Code text file. You are also given the definition of its default constructor and overloaded output stream operator functions. Define all the remaining member functions. The class declaration is heavily commented to help you understand what you need to do.

It is your responsibility to test the correctness of your class definition. To do so, I advise you to write a small test program that tests each member and friend functions of the class before you continue to the next section.

Step 3

You are given the declaration of a **`Course`** class in the Starter Code text file. You are also given the definition of its default constructor and overloaded output stream operator functions. Define all the remaining member functions. Some comments are provided inside the class declaration to help you understand what you need to do; although it should be fairly straight forward to understand the functionality of each member function and thus straightforward to implement.

It is your responsibility to test the correctness of your class definition. To do so, I advise you to write a small test program that tests each member and friend functions of the class before you continue to the next section.

Step 4

You are given the declaration of a **`Date`** struct in the Starter Code text file. There is nothing to add to it. Go to the next step.

Step 5

You are given the declaration of a **`Student`** class in the Starter Code text file. You are also given the definition of its default constructor and overloaded output stream operator functions. Define all the remaining member functions. Some comments are provided inside the class declaration to help you understand what you need to do; although it should be fairly straight forward to understand the functionality of each member function and thus straightforward to implement.

It is your responsibility to test the correctness of your class definition. To do so, I advise you to write a small test program that tests each member and friend functions of the class before you continue to the next section.

Step 6

You are given a typedef for **`Map<int, char>`** as follows:

```
typedef Map<int, char> StudentMap;
```

There is nothing to add to it. Go to the next step.

Step 7

Now that we have the main building blocks for our school management system, we will design the system as described below.

The system will be designed as a class with the following three member variables:

- **`SmarterArray<Student> studentList;`**
  - This will store the students registered in the school.
- **`SmarterArray<Course> courseList;`**
  - This will store the courses offered in the school.
- **`SmarterArray<StudentMap> studentMapList;`**
  - This will store for each student the courses the student is enrolled in and the letter grades for the courses. See below for details.

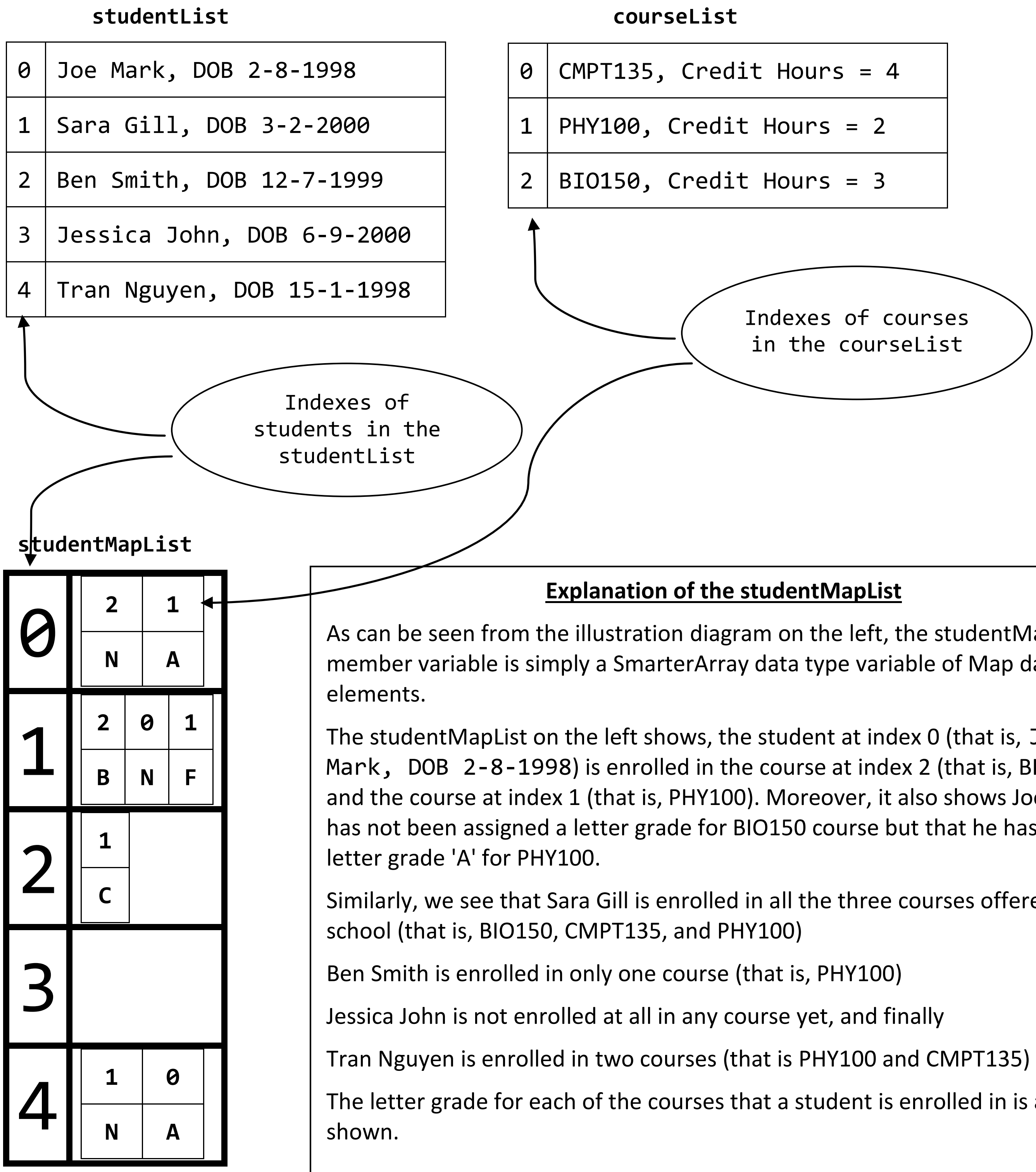
The **`studentMapList`** member variable stores a Map of a **`StudentMap`** type (where the **`StudentMap`** is a typedef described in step 6 above) for each student. Thus the size of the **`studentMapList`** is always the same as the size of **`studentList`**.

The map for each student will store a key-value pair for each of the courses a student is enrolled in, where

- The key is an integer index of a course in the **`courseList`** member variable, and
- The value is the letter grade of the student for the course.

Whenever a student is enrolled in a course, a letter grade of **'N'** is assigned automatically. The letter grade will then be updated whenever the student is assigned a letter grade for the course.

A diagram representation of the school management system is shown below.


studentList
0 Joe Mark, DOB 2-8-1998
1 Sara Gill, DOB 3-2-2000
2 Ben Smith, DOB 12-7-1999
3 Jessica John, DOB 6-9-2000
4 Tran Nguyen, DOB 15-1-1998
courseList
0 CMPT135, Credit Hours = 4
1 PHY100, Credit Hours = 2
2 BIO150, Credit Hours = 3
Indexes of courses in the courseList
Indexes of students in the studentList
studentMapList
0
2 1
N A
1
2 0 1
B N F
2
1
C
3
4
1 0
N A

Explanation of the studentMapList

As can be seen from the illustration diagram on the left, the studentMapList member variable is simply a SmarterArray data type variable of Map data type elements.

The studentMapList on the left shows, the student at index 0 (that is, `Joe Mark, DOB 2-8-1998`) is enrolled in the course at index 2 (that is, BIO150) and the course at index 1 (that is, PHY100). Moreover, it also shows Joe Mark has not been assigned a letter grade for BIO150 course but that he has got the letter grade 'A' for PHY100.

Similarly, we see that Sara Gill is enrolled in all the three courses offered in the school (that is, BIO150, CMPT135, and PHY100)

Ben Smith is enrolled in only one course (that is, PHY100)

Jessica John is not enrolled at all in any course yet, and finally

Tran Nguyen is enrolled in two courses (that is PHY100 and CMPT135)

The letter grade for each of the courses that a student is enrolled in is also shown.

Observe that by having the students and courses stored separately and only having a map for each student will save us a lot of computer memory especially when you have a large number of students and courses in the School Management System. This is the basic idea behind relational databases.

The declaration of **`SchoolManagementSystem`** class is shown below. For the sake of completeness, it is also provided in the Starter Code text file.

```
class SchoolManagementSystem
{
private:
	SmarterArray<Student> studentList; //A SmarterArray to store the students in the school
	SmarterArray<Course> courseList; //A SmarterArray to store the courses in the school
	SmarterArray<StudentMap> studentMapList; //A SmarterArray to store the students' maps
public:
	SchoolManagementSystem();

	int getNumberOfRegisteredStudents() const;
	int getNumberOfCoursesOffered() const;
	int findStudent(const string &firstName, const string &lastName) const;
	Student getStudent(const int &studentIndex) const;
	StudentMap getStudentMap(const int &studentIndex) const;
	int findCourse(const string &courseName) const;
	Course getCourse(const int &courseIndex) const;
	double getStudentGPA(const int &studentIndex) const;
	int getTopStudentIndex() const;

	bool registerStudent(const Student &s);
	bool enrolStudent(const int &studentIndex, const int &courseIndex);
	bool assignLetterGrade(const int &studentIndex, const int &courseIndex, const char &letterGrade);
	bool offerCourse(const Course &course);
	void removeStudent(const int &studentIndex);
	bool withdrawStudent(const int &studentIndex, const int &courseIndex);
	void removeCourse(const int &courseIndex);

	static Student generateRandomStudent();
	static char generateRandomLetterGrade();

	friend ostream& operator << (ostream &, const SchoolManagementSystem &);
};

SchoolManagementSystem::SchoolManagementSystem()
{}

ostream& operator << (ostream &out, const SchoolManagementSystem &sms)
{
	out << endl << "Students List" << endl;
	if (sms.studentList.getSize() == 0)
		out << "No student has been registered yet." << endl;
	for (int studentIndex = 0; studentIndex < sms.studentList.getSize(); studentIndex++)
		out << "Student at index " << studentIndex << ": " << sms.studentList[studentIndex] << endl;

	out << endl << "Courses List" << endl;
	if (sms.courseList.getSize() == 0)
		out << "No course has been offered yet." << endl;
	for (int courseIndex = 0; courseIndex < sms.courseList.getSize(); courseIndex++)
		out << "Course at index " << courseIndex << ": " << sms.courseList[courseIndex] << endl;

	out << endl << "Students Map" << endl;
	if (sms.studentMapList.getSize() == 0)
		out << "No student is enrolled in any course yet." << endl;
	for (int studentIndex = 0; studentIndex < sms.studentMapList.getSize(); studentIndex++)
	{
		out << "Student at index " << studentIndex << endl; out << sms.studentMapList[studentIndex];
		out << "GPA = " << sms.getStudentGPA(studentIndex) << endl << endl;
	}
	return out;
}
```

```
Student SchoolManagementSystem::generateRandomStudent()
{
	string fn, ln;
	Date dob;
	fn = rand() % 26 + 65;
	for (int i = 0; i < 9; i++)
		fn += char(rand() % 26 + 97);
	ln = rand() % 26 + 65;
	for (int i = 0; i < 9; i++)
		ln += char(rand() % 26 + 97);
	dob.y = 1998 + rand() % 5;
	dob.m = 1 + rand() % 12;
	dob.d = 1 + rand() % 30;
	return Student(fn, ln, dob);
}
char SchoolManagementSystem::generateRandomLetterGrade()
{
	int g = rand() % 11;
	if (g == 0)
		return 'A';
	else if (g <= 2)
		return'B';
	else if (g <= 5)
		return'C';
	else if (g <= 7)
		return 'D';
	else
		return 'F';
}
```

The description of the member functions of the theSchoolManagementSystemclass is given below.

- `SchoolManagementSystem();`
  - ➢ Implemented for you. That is there is no need for any code in this member function because all the member variables will be initialized to default values automatically.
- `int getNumberOfRegisteredStudents() const;`
  - ➢ Returns the number of students registered in the school.
- `int getNumberOfCoursesOffered() const;`
  - ➢ Returns the number of courses offered in the school.
- `int findStudent(const string &firstName, const string &lastName) const;`
  - ➢ Returns the index of the first student whose first and last names match the arguments. If no such student exists, then it returns -1.
- `Student getStudent(const int &studentIndex) const;`
  - ➢ Asserts the studentIndex argument and then returns the student object at the given index argument.
- `StudentMap getStudentMap(const int &studentIndex) const;`
  - ➢ Asserts the studentIndex argument and then returns the StudentMap at the given index argument.
- `int findCourse(const string &courseName) const;`
  - ➢ Returns the course index whose name matches the course name argument. If no course name matches the argument, then it returns -1.
- `Course getCourse(const int &courseIndex) const;`
  - ➢ Asserts the courseIndex argument and then returns the course object at the courseIndex argument.
- `double getStudentGPA(const int &studentIndex) const;`
  - ➢ Asserts the studentIndex argument and then returns the student's GPA calculated from the courses the student is enrolled in and whose letter grades are not 'N'. Those courses with letter grade 'N' should not be included in the GPA computation (they must be skipped).

- If a student is not enrolled in any course or if all the letter grades of a student are 'N', then this function must return 0.0 GPA.
- In order to compute a student's GPA, we need to compute the total credit hours and total credit points for the student as follows:
  - The total credit hours is computed by adding the credit hour of each of the courses the student is enrolled in and for which a letter grade different from 'N' is already assigned.
  - The total credit points is computed by adding credit hours * credit points of the letter grade for each of the course the student is enrolled in and for which a letter grade different from 'N' is already assigned. The credit point for the letter grade 'A' is 4.0, for the letter grade 'B' is 3.0, for the letter grade 'C' is 2.0, for the letter grade 'D' is 1.0, and for the letter grade 'F' is 0.0.
  - Then GPA = total credit points divided by total credit hours.

- `int getTopStudentIndex() const;`
  - Asserts that there is at least one student registered in the system and then returns the studentIndex whose GPA is the maximum. If two or more students are found with equal maximum GPAs then this function must return the smallest studentIndex among the equally top students' indexes.
- `bool registerStudent(const Student& s);`
  - It checks if a student that is equal to (==) the Student argument already exists in the system. If yes, it does nothing and returns false. Otherwise, it appends the student argument to the studentList SmarterArray member variable, appends an empty StudentMap object to the studentMapList member variable (i.e. a map for the newly registered student), and finally returns true.
- `bool enrolStudent(const int &studentIndex, const int &courseIndex);`
  - Asserts both the studentIndex and courseIndex arguments. It then checks if the student at the studentIndex argument is already enrolled in the course at the courseIndex argument. If yes, then it does nothing and returns false. Otherwise, it appends the courseIndex argument and a letter grade of 'N' pair to the student's map and returns true.
- `bool assignLetterGrade(const int &studentIndex, const int &courseIndex, const char &letterGrade);`
  - Asserts all the studentIndex, courseIndex and letterGrade arguments. LetterGrade is asserted to see it is equal to one of the characters 'A', 'B', 'C', 'D', or 'F'. Then it checks if the student at the studentIndex argument is enrolled in the course at the courseIndex argument. If no, then it does nothing and returns false. If yes, it assigns the student's course matching the course at the courseIndex argument the letterGrade argument and return true.
- `bool offerCourse(const Course &course);`
  - Checks if a course that is equal to (==) to the Course argument is already offered in the school. If yes, it does nothing and returns false. Otherwise it appends the course argument to the courseList SmarterArray member variable and return true.
- `void removeStudent(const int &studentIndex);`
  - Asserts the studentIndex argument and then removes the element at the studentIndex argument from the studentList and the element at the studentIndex argument from studentMapList.
- `bool withdrawStudent(const int &studentIndex, const int &courseIndex);`
  - Asserts both the studentIndex and the courseIndex arguments. Then it checks if the student at the studentIndex argument is enrolled in the course at the courseIndex argument. If no, then it does nothing and returns false. If yes, it removes the key-value pair element of the student's map that matches the course at the courseIndex argument and returns true. Please note that this function must withdraw a student from a course irrespective of the fact that the student has been assigned a letter grade for the course or not.
- `void removeCourse(const int &courseIndex);`
  - It asserts the courseIndex.
  - Withdraws every student from the course at the courseIndex argument.
  - Removes the element of the courseList SmarterArray member variable at the courseIndex argument.

- ➢ Decrement (that is decrease by 1) each key value greater than the courseIndex argument from all the students' maps. Why? Remember that when an element of an array is removed (deleted), the indexes of those elements of the array that come after (behind) the element being deleted are going to change (decrease by 1) because the elements are going to be shifted by one step to the left side. This means that after this function removes a course from the courseList SmarterArray, then the courses that are stored after (behind) the course just removed will all have their indexes decreased by 1. Since the map for each student stores the course index in the key array, therefore we need to decrement (that is decrease by 1) each key value greater than the courseIndex argument from all the students' maps.

- `static Student generateRandomStudent();`
  - ➢ Implemented for you.
- `static char generateRandomLetterGrade();`
  - ➢ Implemented for you.
- `friend ostream& operator << (ostream &, const SchoolManagementSystem &);`
  - ➢ Implemented for you.

Implement the SchoolManagementSystem class as described above.

Step 8

You are provided a test program in the Starter Code text file. Use the provided test program to test your work. The output of a sample run of the test program is also provided in a separate file named Sample Run Output text file for your reference. For those of you working on the same IDE, your output should be identical to the outputs provided; while for those of you with different IDE then you need trace your outputs manually to make sure your program is working without any semantic errors.

Please note that the provided test program registers very few students and offers very few courses in order to make the sample output size manageable. Surely your work will be tested with more students and more courses and more operations such as assigning letter grades, withdrawals, course removals, student removals, etc. So I advice you test your work with more data and more operations.

Submitting Your Work

You are required to submit your source code file (that is .cpp file) through Moodle that contains

- The declaration and definition of the SmarterArray class template,
- The declaration and definition of the Map class template,
- The declaration and definition of the Course class,
- The declaration of the Date struct,
- The declaration and definition of the Student class,
- The given typedef (typedef Map<int, char> StudentMap;),
- The declaration and definition of the SchoolManagementSystem class, and
- The test program.