degree.h

```
//File:
degree.h
#pragma once

enum class degreeProgramEnum { SECURITY=0, NETWORK=1, SOFTWARE=2, NONE=3 };
```

In here pragma once line is used to avoid the compilation issues. It also reduce the compilation time. When in another file if the degree.h is included (#inclde “degree.h”) once compiler will only read this file once. If another file has the same inclusion compiler no need to read this because it is already read.

If we do not include that compiler will find every time it sees the inclusion of degree.h file.

enum class is user data type. In here we assign each for numbers (levels). In here we assign each values. That means degreeProgramEnum data type can have different values. If we assign SECURITY it measn this has 0 as value.

roster.h and student.h

```
#pragma once
```

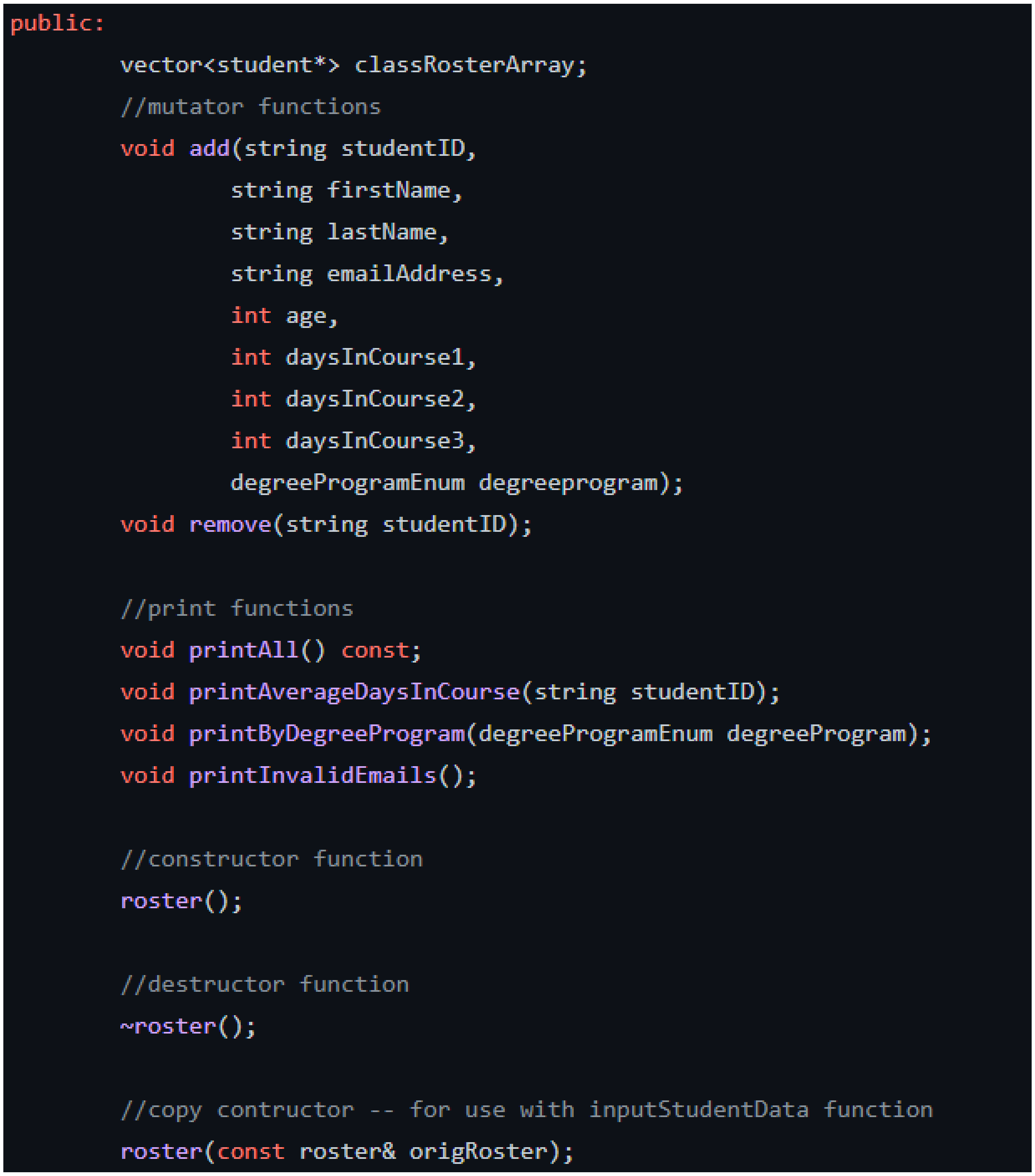
This line is for only compile this file when initial inclusion is seen. And after that don does not compile this even included in another file.

Class is and concept of the Object-Oriented Programming. We see things as objects. In here roster is seen as a object. For implementation wee need to given the structure of this object. For that we use class. In contrast Class is the blueprint for a object.

Before create the object we should define the what are the attributes and methods(what object can do) that we defined. In cpp standard we define class structure in the headrer file. And the real implementation in the cpp file. Both should have the asme name except the extension.

(roster.h , roster.cpp)

Inside the file we need to wrap the all attrubutes and method inside the class roster {}.



```
public:
        vector<student*> classRosterArray;
        //mutator functions
        void add(string studentID,
                string firstName,
                string lastName,
                string emailAddress,
                int age,
                int daysInCourse1,
                int daysInCourse2,
                int daysInCourse3,
                degreeProgramEnum degreeprogram);
        void remove(string studentID);

        //print functions
        void printAll() const;
        void printAverageDaysInCourse(string studentID);
        void printByDegreeProgram(degreeProgramEnum degreeProgram);
        void printInvalidEmails();

        //constructor function
        roster();

        //destructor function
        ~roster();

        //copy contructor -- for use with inputStudentData function
        roster(const roster& origRoster);
```

Inside the class there are access modifiers. In here public is used. That means all the attributes and the methods can be used from (access) from another class.

In here roster class has one public attribute called classRosterArray which is a vector that keeps objects of student class.

// Methods

In here there are two mutator function (Function that change the state) add and remove. Add function will not return anything since it is declared as void. It takes several arguments such as studentID, firstName,….

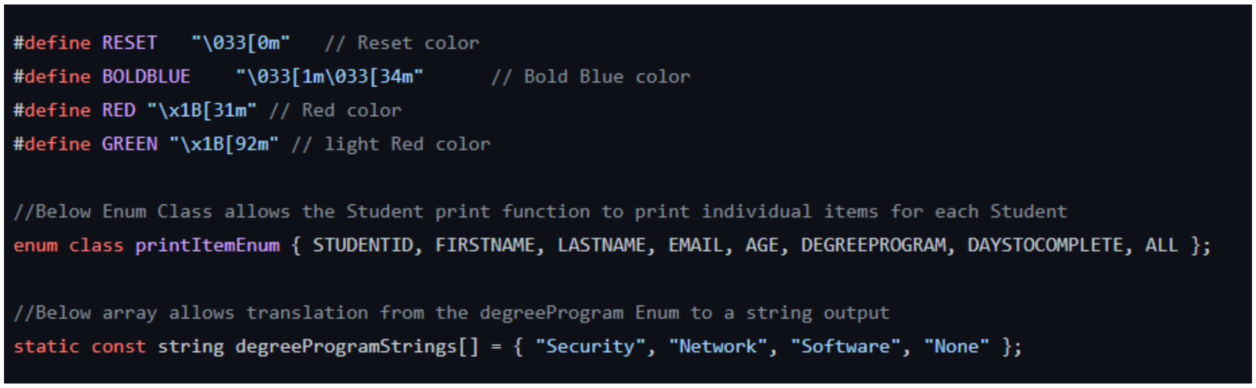
Remove function is also void. And will take one argument.

The behaviour of the function is in the roster.cpp file.

// print functions in roster.h

These functions does not change the state of the object. It will print the current available information in different way. In here printAll() member function is defined as const. That means those functions which are denied permission to change the values of the data members of their class.

Student.h



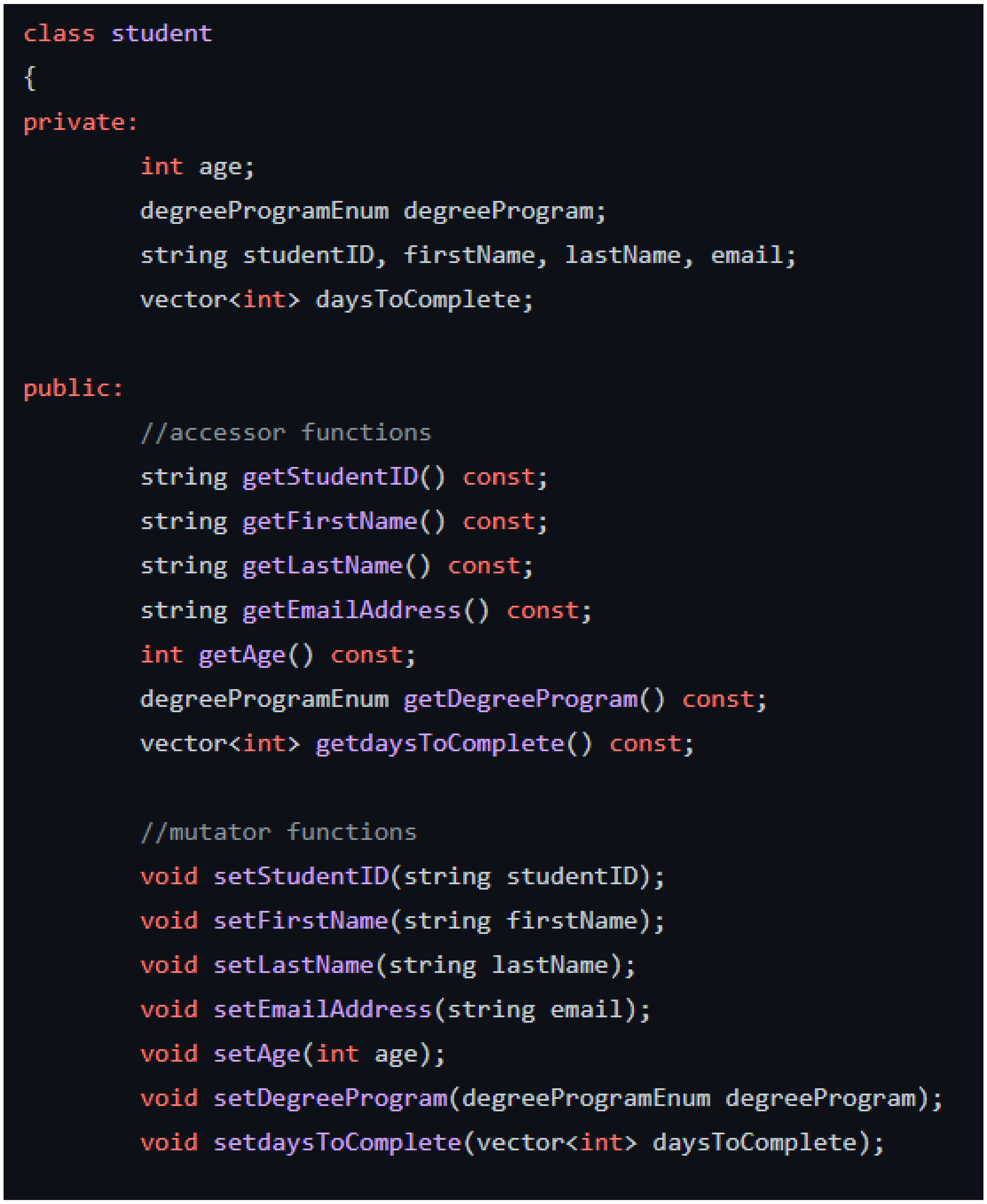
```
#define RESET   "\033[0m"   // Reset color
#define BOLDBLUE    "\033[1m\033[34m"      // Bold Blue color
#define RED "\x1B[31m" // Red color
#define GREEN "\x1B[92m" // light Red color

//Below Enum Class allows the Student print function to print individual items for each Student
enum class printItemEnum { STUDENTID, FIRSTNAME, LASTNAME, EMAIL, AGE, DEGREEPROGRAM, DAYSTOCOMPLETE, ALL };

//Below array allows translation from the degreeProgram Enum to a string output
static const string degreeProgramStrings[] = { "Security", "Network", "Software", "None" };
```

In the student header file same implementation details is applied for the OOP. But in here there are some specific values define before the class definition. These values specific only for the student.h file and uses of it.

In the above lines there are macros. (That somputed in the preprossesing time). There are four defined macros in the file. #define <NAME_OF_THE_MACRO> VALUE. In here values are given in string data type. These strings are kind of special strings. These are called ANSI escaping characters or Unicode. These will print the colors in to the console. In here RESET in assign to “\033[0m” . When we use cout and print RESET it will clear the color of the console. Like wise there are many colors and spaces can be printed to the console(terminal).

After that there is printItemEnum data type which enum class. In here there are different level but not assign values. When we does not assign it will assign from 0 to en. (STUDENTID = 0, FIRSTNAME = 1 ,…., ALL =7)

Below that line array is initialized and it is static string type which hold "Security", "Network", "Software", "None" strings. Since it is static it can be accessed from anywhere and not destroy like in local variables.



```
class student
{
private:
        int age;
        degreeProgramEnum degreeProgram;
        string studentID, firstName, lastName, email;
        vector<int> daysToComplete;

public:
        //accessor functions
        string getStudentID() const;
        string getFirstName() const;
        string getLastName() const;
        string getEmailAddress() const;
        int getAge() const;
        degreeProgramEnum getDegreeProgram() const;
        vector<int> getdaysToComplete() const;

        //mutator functions
        void setStudentID(string studentID);
        void setFirstName(string firstName);
        void setLastName(string lastName);
        void setEmailAddress(string email);
        void setAge(int age);
        void setDegreeProgram(degreeProgramEnum degreeProgram);
        void setdaysToComplete(vector<int> daysToComplete);
```

Like in the above roster header file these also have attributes and methods. In this class student class has private variable age , degreeProhram , studentID, firstName, lastName, email and int vector called daysToComplete. These are defined as private. It means these variable cannot be access from the another class or subclass.

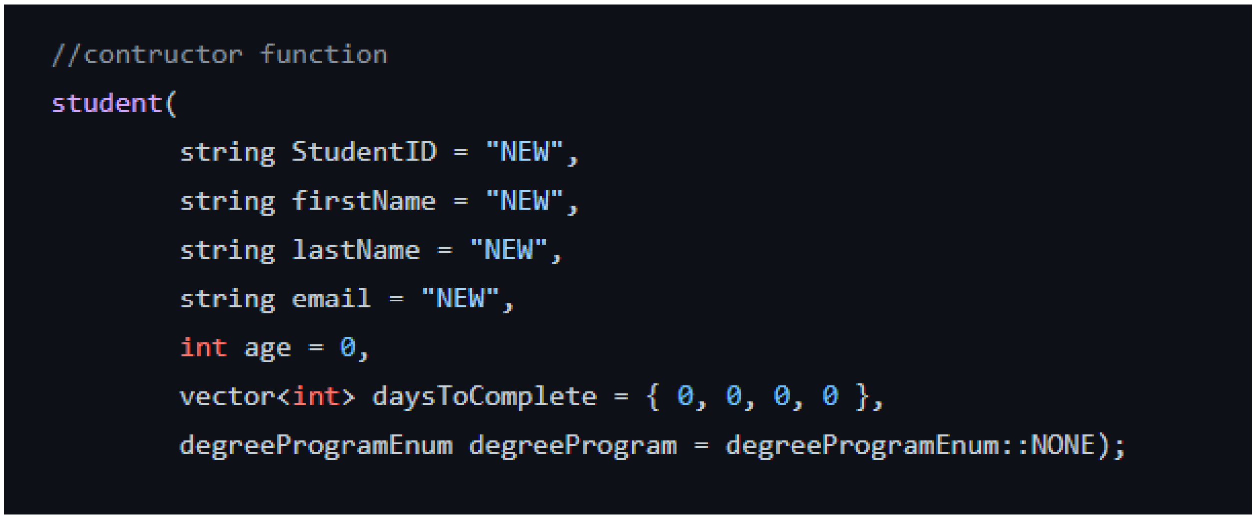
// accessor function

But let's say we need to change or get the value of the age. These can not be done like in the example.

Student.age

Because age is private attribute. For do that we need a public function to change or read that varables. These functions are called accessor functions. All of these functions will be used to read the private variables. Therefore it should denied the changes happen to the attributes. Therefore these functions are defined as const.

In the other hand mutators are the functions that will change the state. These function are used to change the values of attributes of the class.



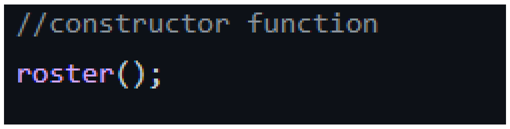
```
//contructor function
student(
        string StudentID = "NEW",
        string firstName = "NEW",
        string lastName = "NEW",
        string email = "NEW",
        int age = 0,
        vector<int> daysToComplete = { 0, 0, 0, 0 },
        degreeProgramEnum degreeProgram = degreeProgramEnum::NONE);
```

This is the same as the contructor. But this is required arguments. And as default values it assign some values to the arguments I the user did not given any argument.

Also this has const print function to print some data.

//Constructor

This the function that creates an object from this defined class. This will only call when the creation of the object. In standard we use this to initialize the attributes (if have). If do not define the values when this functions is called it will auto assign values such that if int,double , as 0 and 0.0 , string as empty and address as null, etc.

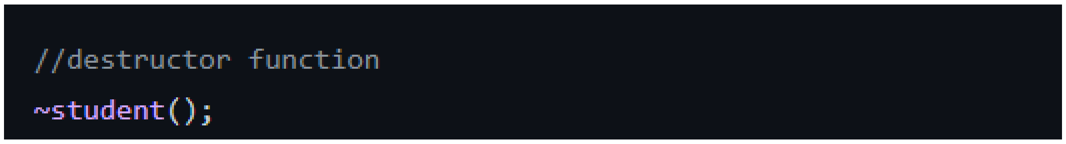


```
//constructor function
roster();
```

// deconstructor

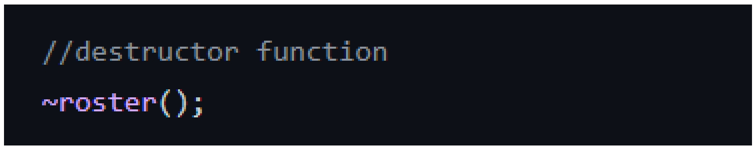
This is automatically called when the object is going to detroy. We can specify what should be done before object is destroy.

Student class



```
//destructor function
~student();
```

Roster class



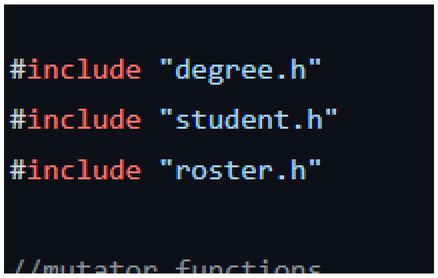
```
//destructor function
~roster();
```

// Copy constructor

A **copy constructor** is a member function that initializes an object using another object of the same class

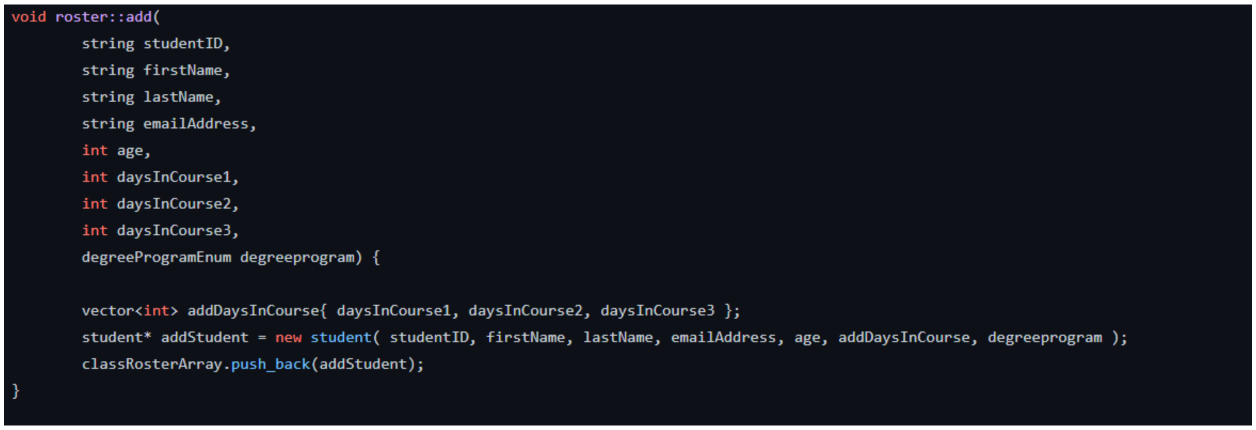
roster.cpp

This is the file for the implementation of the structure of the class.



```
#include "degree.h"
#include "student.h"
#include "roster.h"

//mutator functions
```

We need to link the header file to access to the structure of thdefined clas. For that we include the files in above.



```
void roster::add(
        string studentID,
        string firstName,
        string lastName,
        string emailAddress,
        int age,
        int daysInCourse1,
        int daysInCourse2,
        int daysInCourse3,
        degreeProgramEnum degreeprogram) {

        vector<int> addDaysInCourse{ daysInCourse1, daysInCourse2, daysInCourse3 };
        student* addStudent = new student( studentID, firstName, lastName, emailAddress, age, addDaysInCourse, degreeprogram );
        classRosterArray.push_back(addStudent);
}
```

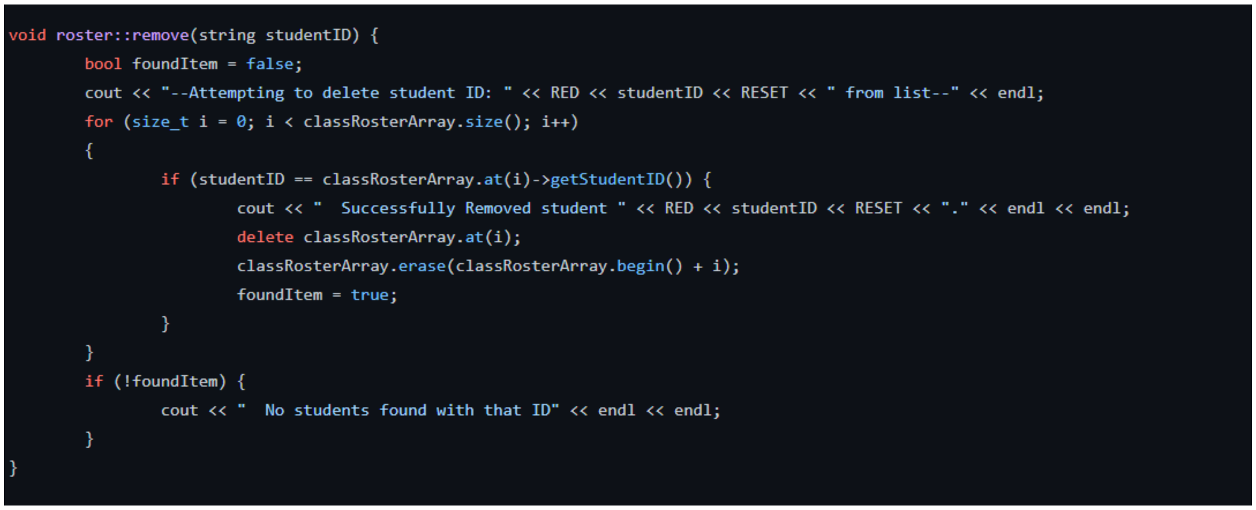
Roster has a function called add. We need to define the behaviour (what should this function do) in here. Before that we need to tell the compiler we need to implement the add function which is defind the roster header file. For that we use void roster:: add(…

In here add function take the arguments srudentID, firstName,lastName, emailAddress, age, daysInCourse1, daysInCourse2, daysInCourse3 and degreeProgramEnum data type names degreeprogram. (This data type defined in another file. See the above)

Inside the functionit is create and int vector called addDaysInCourse. And put values oof daysInCourse1,daysInCourse2 and daysInCpurse3.

Then addStudent is initialies to an object of the student. To create the object constructor of the stdent is used with the new keyweord. It will create a student object with given data (arguments).

That student is added to the array of roster object called classRosterArray.

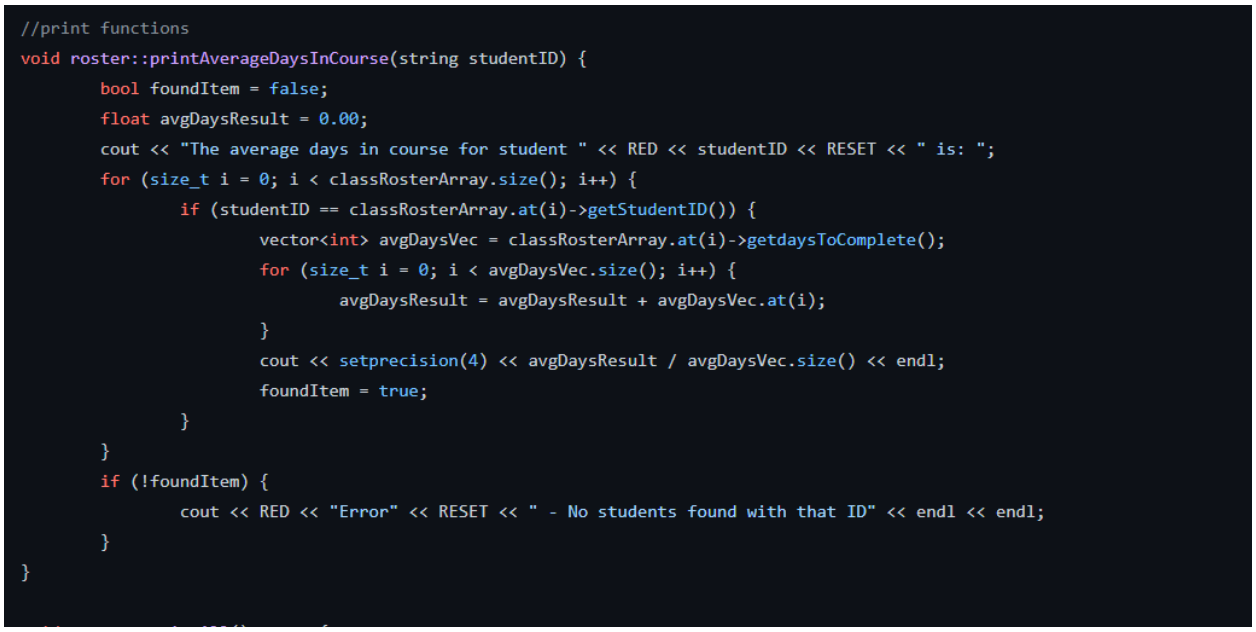


```
void roster::remove(string studentID) {
        bool foundItem = false;
        cout << "--Attempting to delete student ID: " << RED << studentID << RESET << " from list--" << endl;
        for (size_t i = 0; i < classRosterArray.size(); i++)
        {
                if (studentID == classRosterArray.at(i)->getStudentID()) {
                        cout << "  Successfully Removed student " << RED << studentID << RESET << "." << endl << endl;
                        delete classRosterArray.at(i);
                        classRosterArray.erase(classRosterArray.begin() + i);
                        foundItem = true;
                }
        }
        if (!foundItem) {
                cout << "  No students found with that ID" << endl << endl;
        }
}
```

Remove function will only get one argument which is tring type studentID. It initialize a Boolean varable foundItem to false.

The print the the message to the console (--Attempting to delet student ID: <red color (one character size) <studentID and clear the color then from list--

After that loop through the classRosterArray using a for loop. each round it will get the current I th value position student from the list and compare whether id of that student is matched with the given id.if match occurs print a message to console. Then delete that student from the vector. Then that position is empty. But we need to make vector such as contiguous. For that we use in buit method erase. And set the foundItem to true. After that we check foundItem value negation. If it is false still not student found from that id .



```
//print functions
void roster::printAverageDaysInCourse(string studentID) {
	bool foundItem = false;
	float avgDaysResult = 0.00;
	cout << "The average days in course for student " << RED << studentID << RESET << " is: ";
	for (size_t i = 0; i < classRosterArray.size(); i++) {
		if (studentID == classRosterArray.at(i)->getStudentID()) {
			vector<int> avgDaysVec = classRosterArray.at(i)->getdaysToComplete();
			for (size_t i = 0; i < avgDaysVec.size(); i++) {
				avgDaysResult = avgDaysResult + avgDaysVec.at(i);
			}
			cout << setprecision(4) << avgDaysResult / avgDaysVec.size() << endl;
			foundItem = true;
		}
	}
	if (!foundItem) {
		cout << RED << "Error" << RESET << " - No students found with that ID" << endl << endl;
	}
}
```

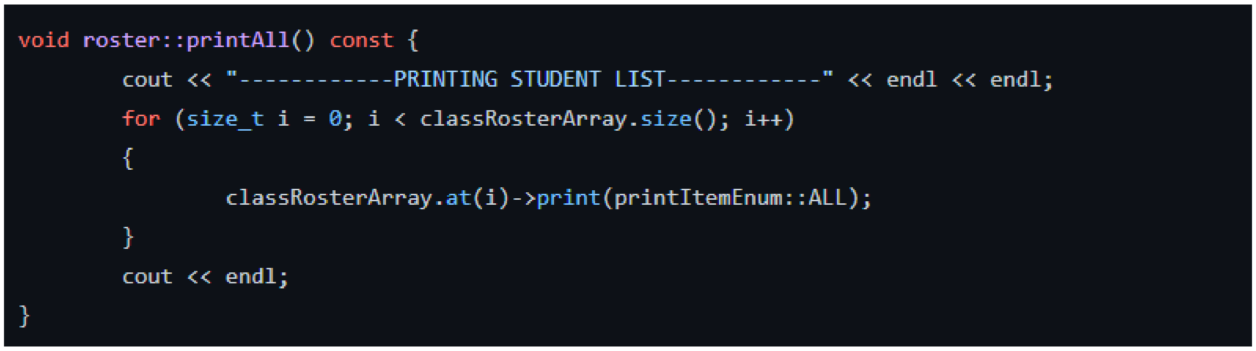
In here this function is used to print the information of a student giben in the relevant id as argument.

Same as the above function it initilizedd a foundItem boolen to false. Also avgDaysResult to 0.0.

Then print a message The average days in course for student <red> id and reselt the olor to normal is:

Then loop torugh the list until student is found. If the student found (match) avgDyas vector create with and set it is to daysToComplete array of student attribute. Then loop again through that int vector and cumulatively add the values in that array to avgDaysResult variable . Then set the the presion (number of figures of the result to four and print the value of avgDaysResult divide by the number of elemnt in the vector. Then Set the boolen to tru.

If the tudet not found boolen is still false if this happens print a error message.



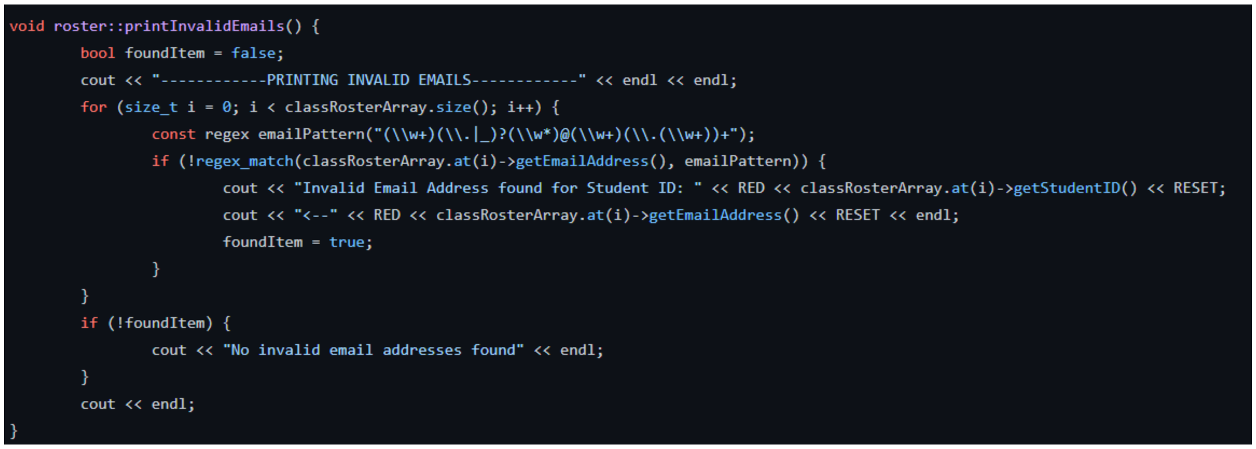
```
void roster::printAll() const {
	cout << "------------PRINTING STUDENT LIST------------" << endl << endl;
	for (size_t i = 0; i < classRosterArray.size(); i++)
	{
		classRosterArray.at(i)->print(printItemEnum::ALL);
	}
	cout << endl;
}
```

This function is used to print the student list.

It print message printing student list.

And loop through the classRosterArray. For each student print the the details using print function of student by setting printItemEnum as ALL.

Then print a new line

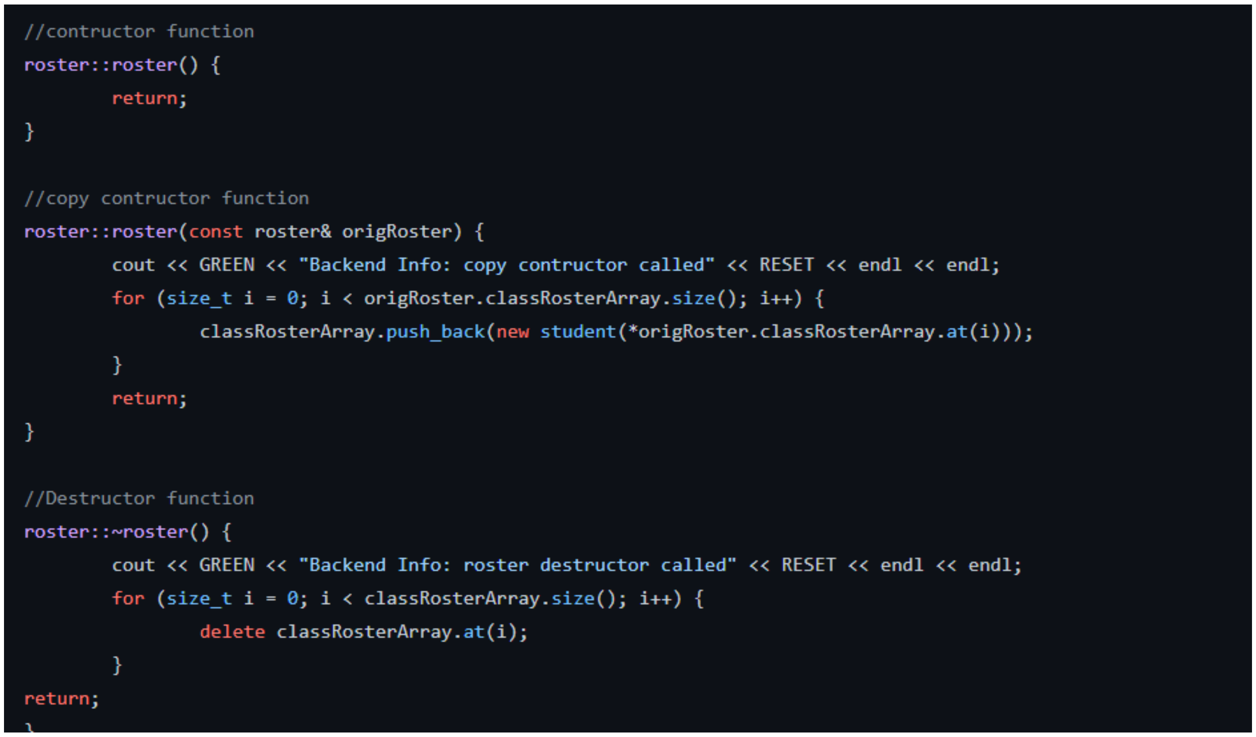


```
void roster::printInvalidEmails() {
        bool foundItem = false;
        cout << "------------PRINTING INVALID EMAILS------------" << endl << endl;
        for (size_t i = 0; i < classRosterArray.size(); i++) {
                const regex emailPattern("(\\w+)(\\.|_)?(\\w*)@(\\w+)(\\.(\\w+))+");
                if (!regex_match(classRosterArray.at(i)->getEmailAddress(), emailPattern)) {
                        cout << "Invalid Email Address found for Student ID: " << RED << classRosterArray.at(i)->getStudentID() << RESET;
                        cout << "<--" << RED << classRosterArray.at(i)->getEmailAddress() << RESET << endl;
                        foundItem = true;
                }
        }
        if (!foundItem) {
                cout << "No invalid email addresses found" << endl;
        }
        cout << endl;
}
```

This function will print the details of students with invalid email.

Initlailly set the boolen variable to false. And if the no student found print a invalid message at the end. (No invalid student)

Loop thoruh each student from the list check the validity of the email attribute using regular expression. If the email not met with the regex print the invalid email address with the student if and set boolen to true. This will go every student in the list.



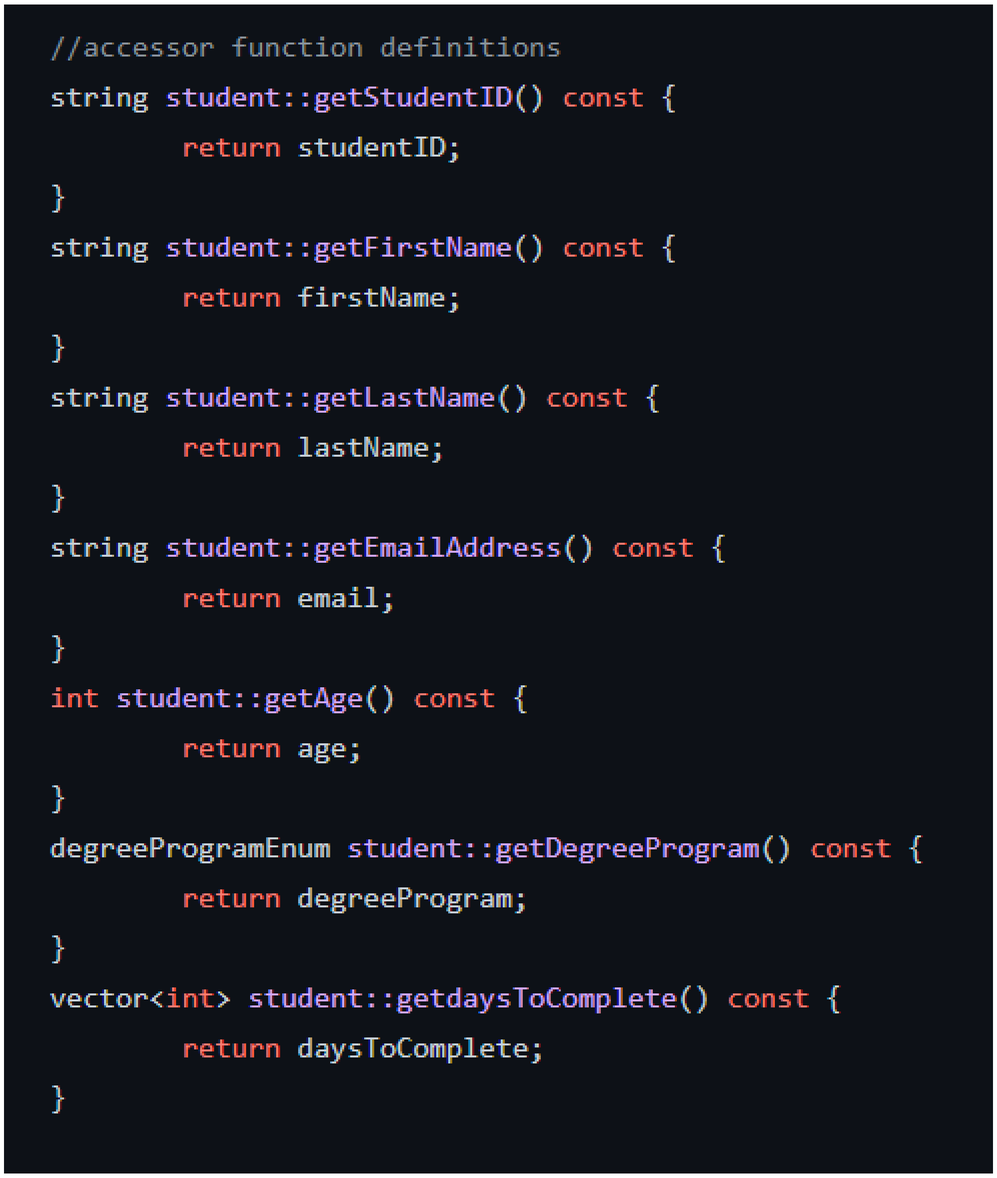
```
//contructor function
roster::roster() {
        return;
}

//copy contructor function
roster::roster(const roster& origRoster) {
        cout << GREEN << "Backend Info: copy contructor called" << RESET << endl << endl;
        for (size_t i = 0; i < origRoster.classRosterArray.size(); i++) {
                classRosterArray.push_back(new student(*origRoster.classRosterArray.at(i)));
        }
        return;
}

//Destructor function
roster::~roster() {
        cout << GREEN << "Backend Info: roster destructor called" << RESET << endl << endl;
        for (size_t i = 0; i < classRosterArray.size(); i++) {
                delete classRosterArray.at(i);
        }
return;
```

In the constructor it will not doing anything.

In the copy constructor it will print a message . then loop through each student and recreate the classRosterArray of new object.

In the destructor it will delete all the students details in the list before destroy the roster object

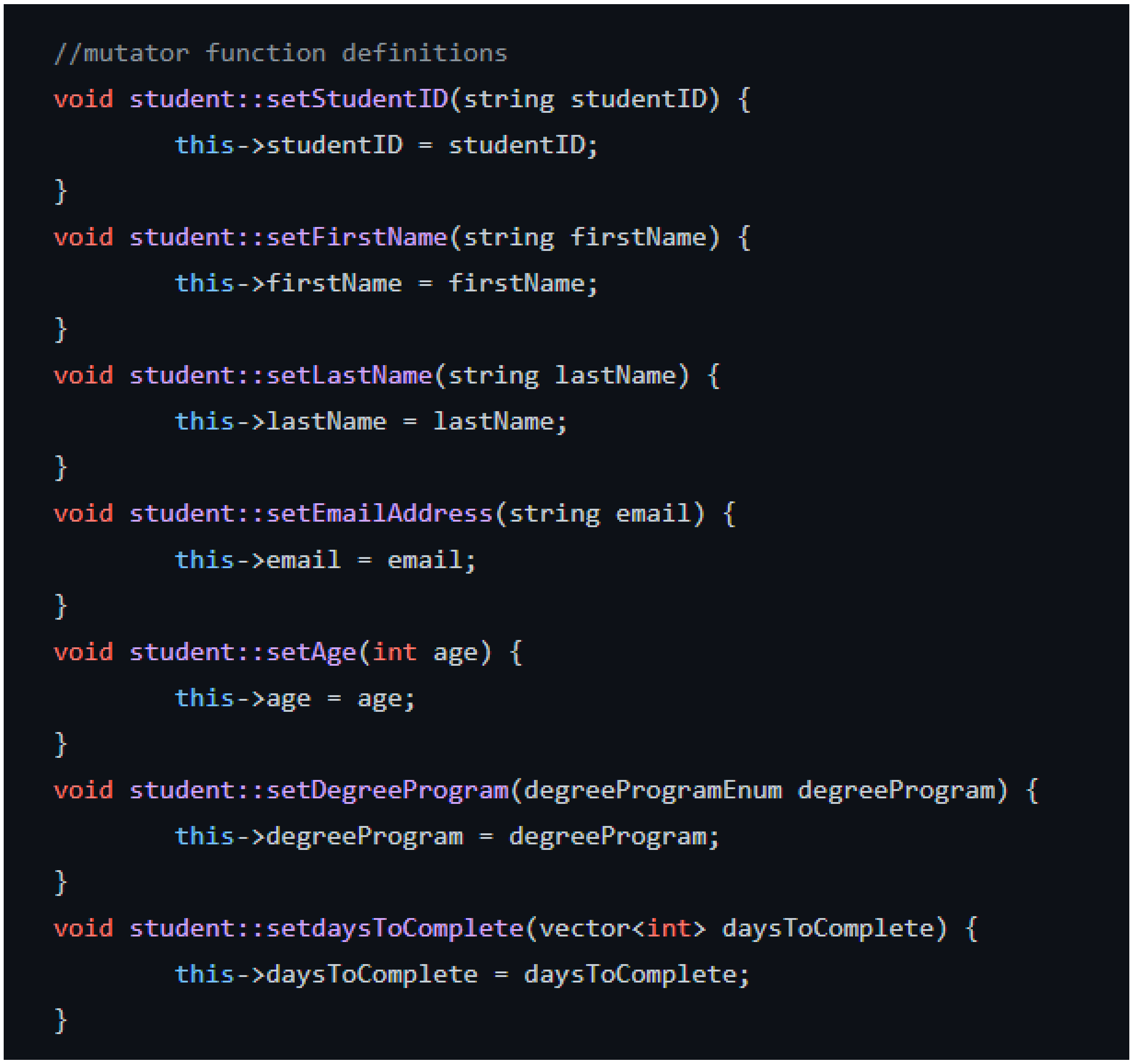
Student.cpp



```
//accessor function definitions
string student::getStudentID() const {
        return studentID;
}
string student::getFirstName() const {
        return firstName;
}
string student::getLastName() const {
        return lastName;
}
string student::getEmailAddress() const {
        return email;
}
int student::getAge() const {
        return age;
}
degreeProgramEnum student::getDegreeProgram() const {
        return degreeProgram;
}
vector<int> student::getdaysToComplete() const {
        return daysToComplete;
}
```

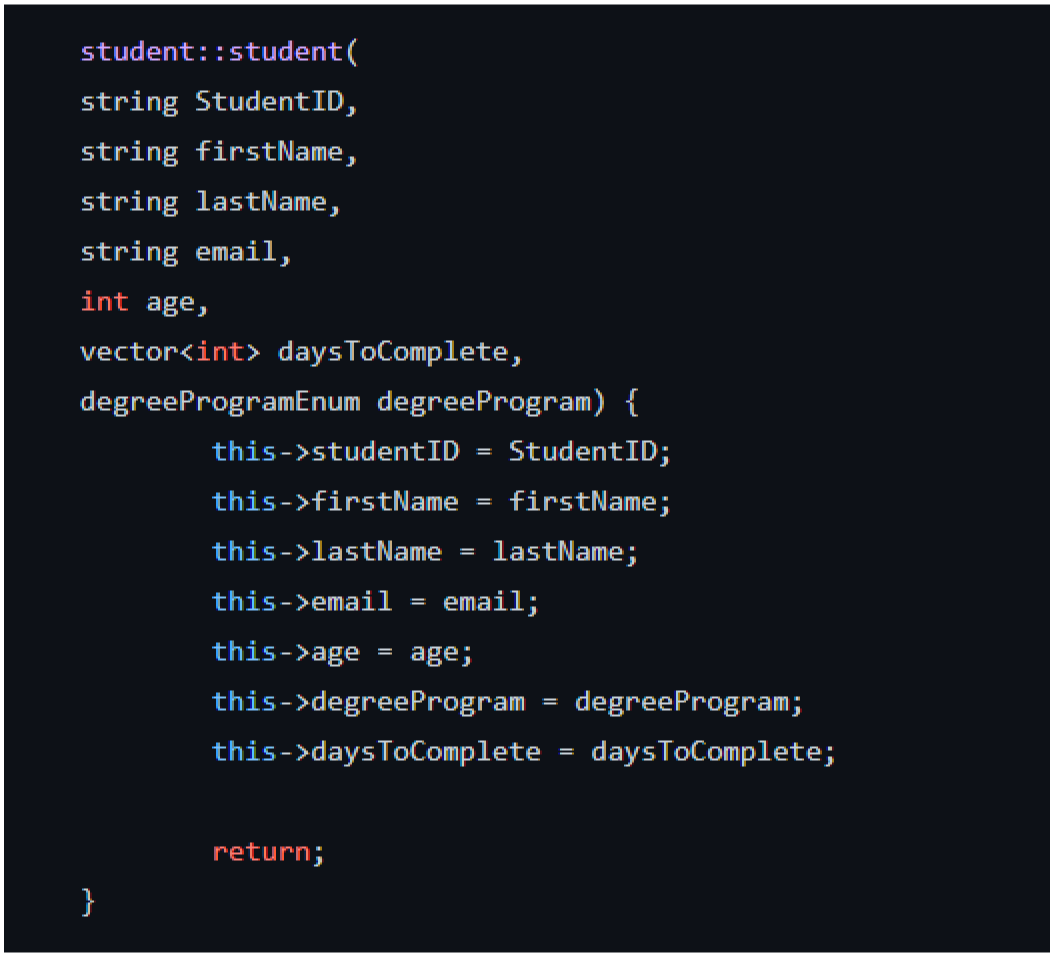
In the stdeunt class implementation initially implement the accessors.

Each of thses accessors are used to get the private attribute. Functions are defined such that it will return the relevant data type of the attribute. When we need to get the studentID of the studentit will returm the string which is studentId likewise.


```
//mutator function definitions
void student::setStudentID(string studentID) {
        this->studentID = studentID;
}
void student::setFirstName(string firstName) {
        this->firstName = firstName;
}
void student::setLastName(string lastName) {
        this->lastName = lastName;
}
void student::setEmailAddress(string email) {
        this->email = email;
}
void student::setAge(int age) {
        this->age = age;
}
void student::setDegreeProgram(degreeProgramEnum degreeProgram) {
        this->degreeProgram = degreeProgram;
}
void student::setdaysToComplete(vector<int> daysToComplete) {
        this->daysToComplete = daysToComplete;
}
```

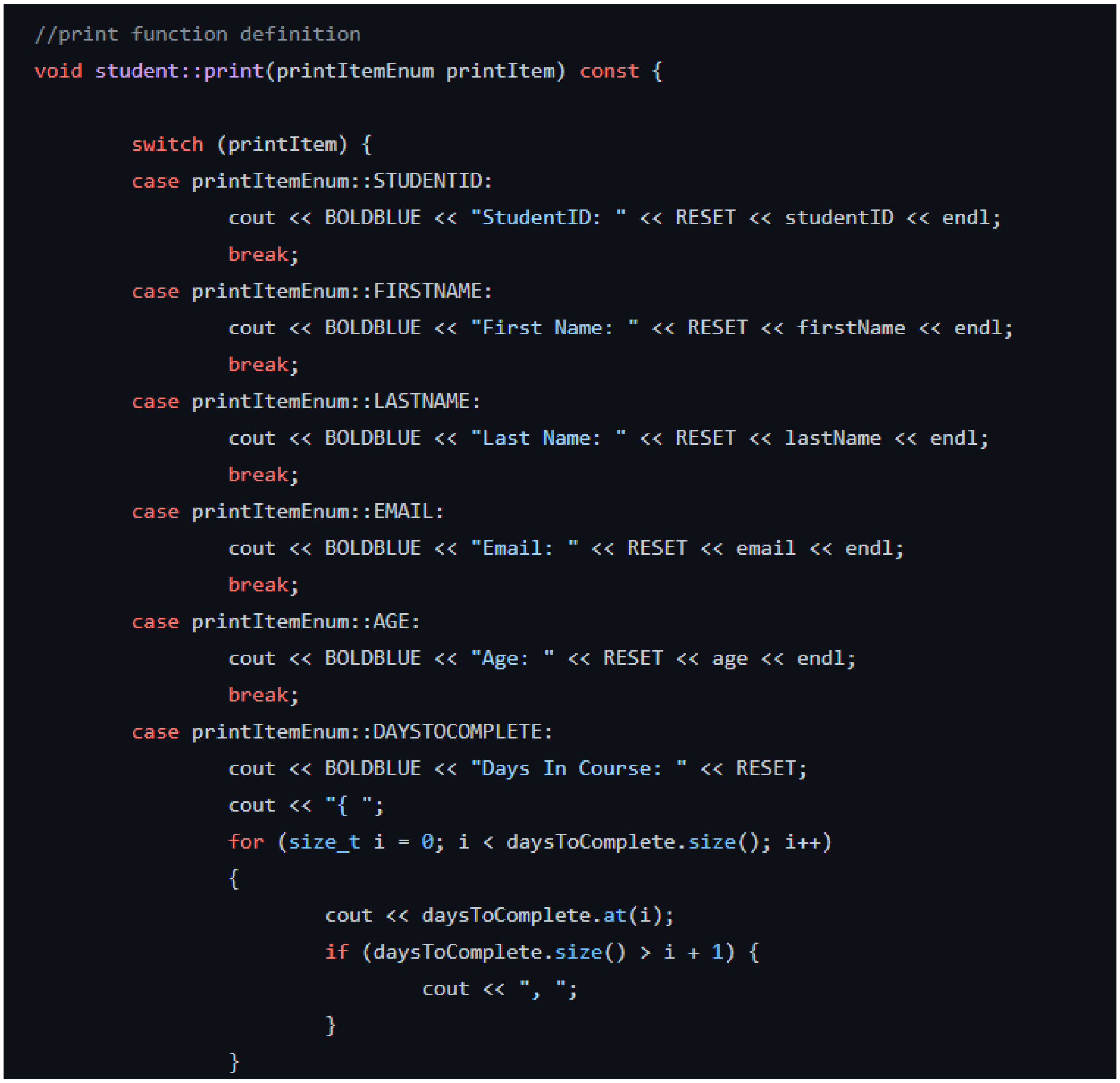
This functions are used to set(change) the values of attributes. These functions are get the relevant value that should be set to the attribute as argument. Then hat value is assigned to the relevant private varable.

In here this keyword is used. It means this refer to the object. For example setSstudentId() function hace same name variable as in the argument (studentID and studentID). To avoid that congestion we use this. Whe we use this it refers to the object. Means it refers current objects studentID attribute.



```
student::student(
string StudentID,
string firstName,
string lastName,
string email,
int age,
vector<int> daysToComplete,
degreeProgramEnum degreeProgram) {
        this->studentID = StudentID;
        this->firstName = firstName;
        this->lastName = lastName;
        this->email = email;
        this->age = age;
        this->degreeProgram = degreeProgram;
        this->daysToComplete = daysToComplete;

        return;
}
```

This is the constructor of the student class. When the object is created this function is automatically called. Tis will get several arguments. And assign each values to relevant attributes of the class. When the function is called it will create an object with given details.

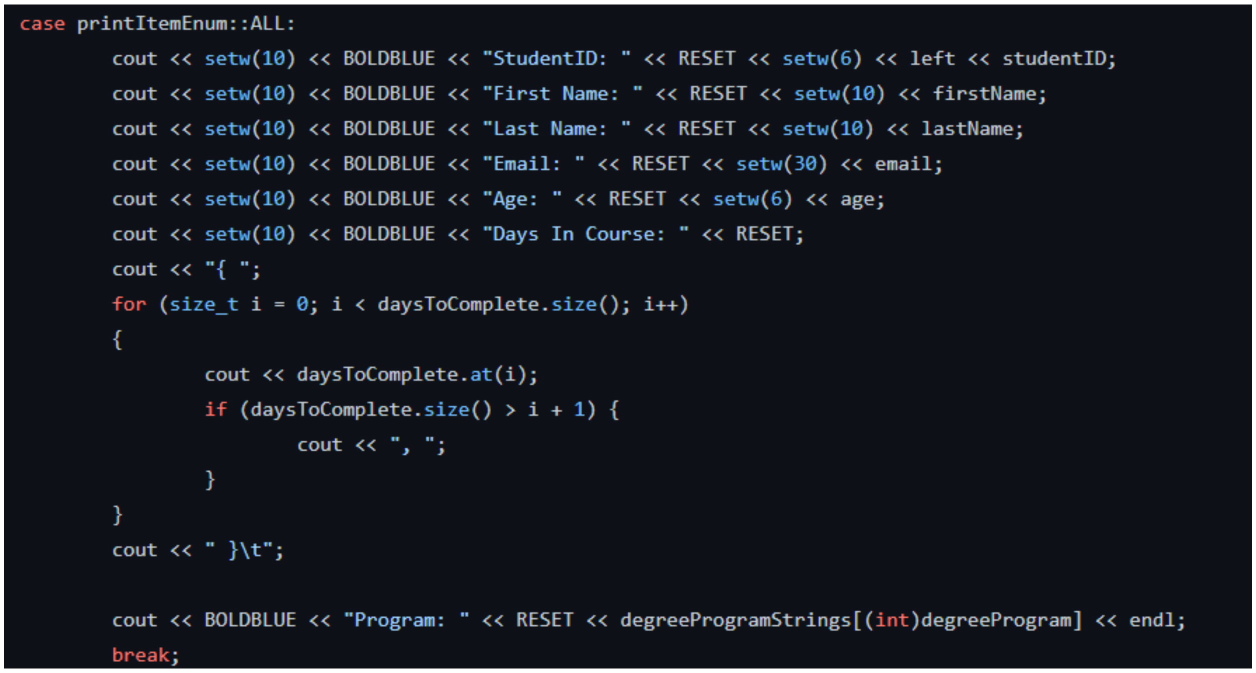


```cpp
//print function definition
void student::print(printItemEnum printItem) const {

	switch (printItem) {
	case printItemEnum::STUDENTID:
		cout << BOLDBLUE << "StudentID: " << RESET << studentID << endl;
		break;
	case printItemEnum::FIRSTNAME:
		cout << BOLDBLUE << "First Name: " << RESET << firstName << endl;
		break;
	case printItemEnum::LASTNAME:
		cout << BOLDBLUE << "Last Name: " << RESET << lastName << endl;
		break;
	case printItemEnum::EMAIL:
		cout << BOLDBLUE << "Email: " << RESET << email << endl;
		break;
	case printItemEnum::AGE:
		cout << BOLDBLUE << "Age: " << RESET << age << endl;
		break;
	case printItemEnum::DAYSTOCOMPLETE:
		cout << BOLDBLUE << "Days In Course: " << RESET;
		cout << "{ ";
		for (size_t i = 0; i < daysToComplete.size(); i++)
		{
			cout << daysToComplete.at(i);
			if (daysToComplete.size() > i + 1) {
				cout << ", ";
			}
		}
```

In here this function is print seceral details pattern according to the value given in the argument.

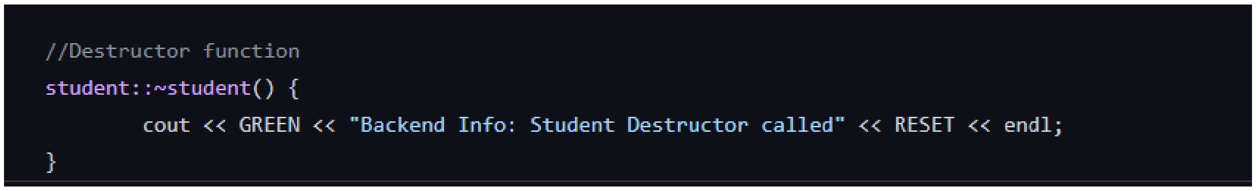
In here switch data structure is used. Switch is similar to if elseif elseif else ladder. In each case it will check the printItem value giben to the switch case. If that is equal to the STUDENTIF then print the student id. Etc

If the printItem is equal to the DAYSTOCOMPLETE then it calculate the date.



```
case printItemEnum::ALL:
        cout << setw(10) << BOLDBLUE << "StudentID: " << RESET << setw(6) << left << studentID;
        cout << setw(10) << BOLDBLUE << "First Name: " << RESET << setw(10) << firstName;
        cout << setw(10) << BOLDBLUE << "Last Name: " << RESET << setw(10) << lastName;
        cout << setw(10) << BOLDBLUE << "Email: " << RESET << setw(30) << email;
        cout << setw(10) << BOLDBLUE << "Age: " << RESET << setw(6) << age;
        cout << setw(10) << BOLDBLUE << "Days In Course: " << RESET;
        cout << "{ ";
        for (size_t i = 0; i < daysToComplete.size(); i++)
        {
                cout << daysToComplete.at(i);
                if (daysToComplete.size() > i + 1) {
                        cout << ", ";
                }
        }
        cout << " }\t";

        cout << BOLDBLUE << "Program: " << RESET << degreeProgramStrings[(int)degreeProgram] << endl;
        break;
```

If the printItem flasg is ALL then print the student id first name ,… days In course .here number of figures is set to 10.

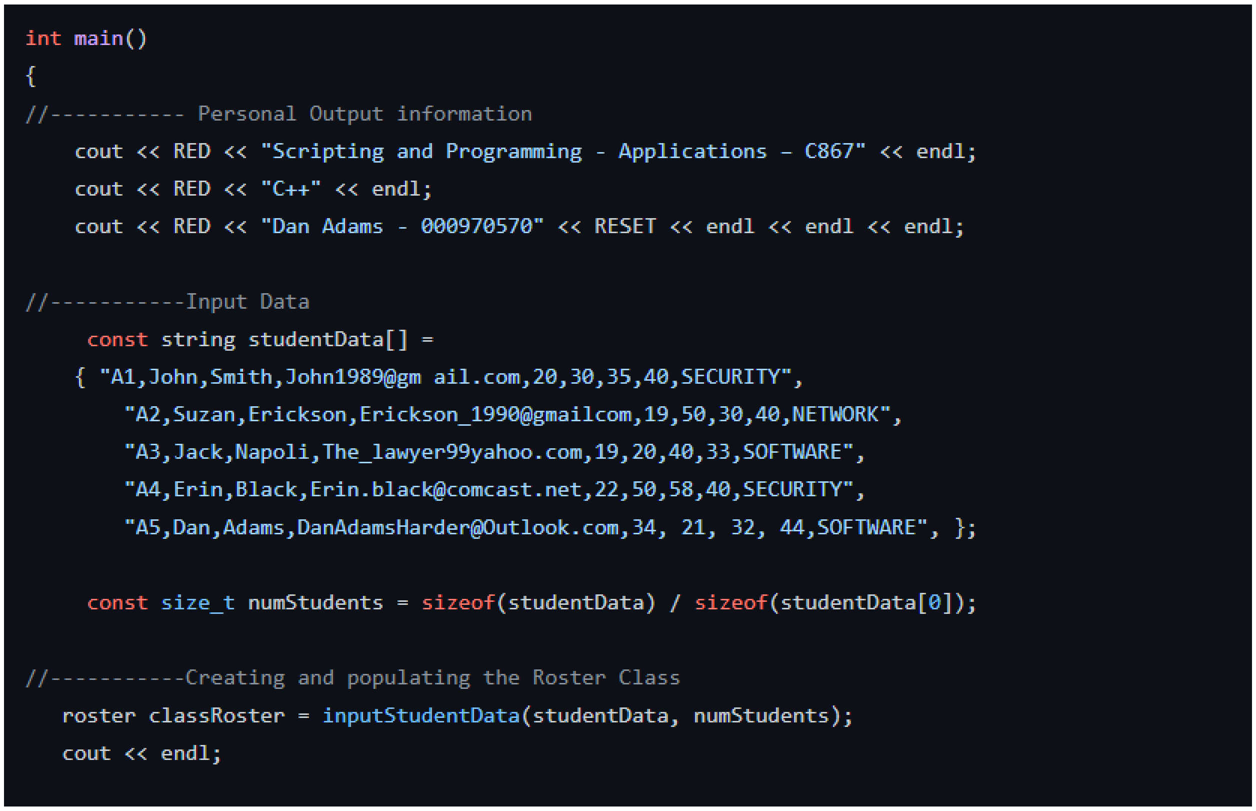


```
  //Destructor function
  student::~student() {
          cout << GREEN << "Backend Info: Student Destructor called" << RESET << endl;
  }
```

When the object is destroy it will print a message before it get destroy.

main.cpp

This is file where program run. In the function called main() will run the program first.



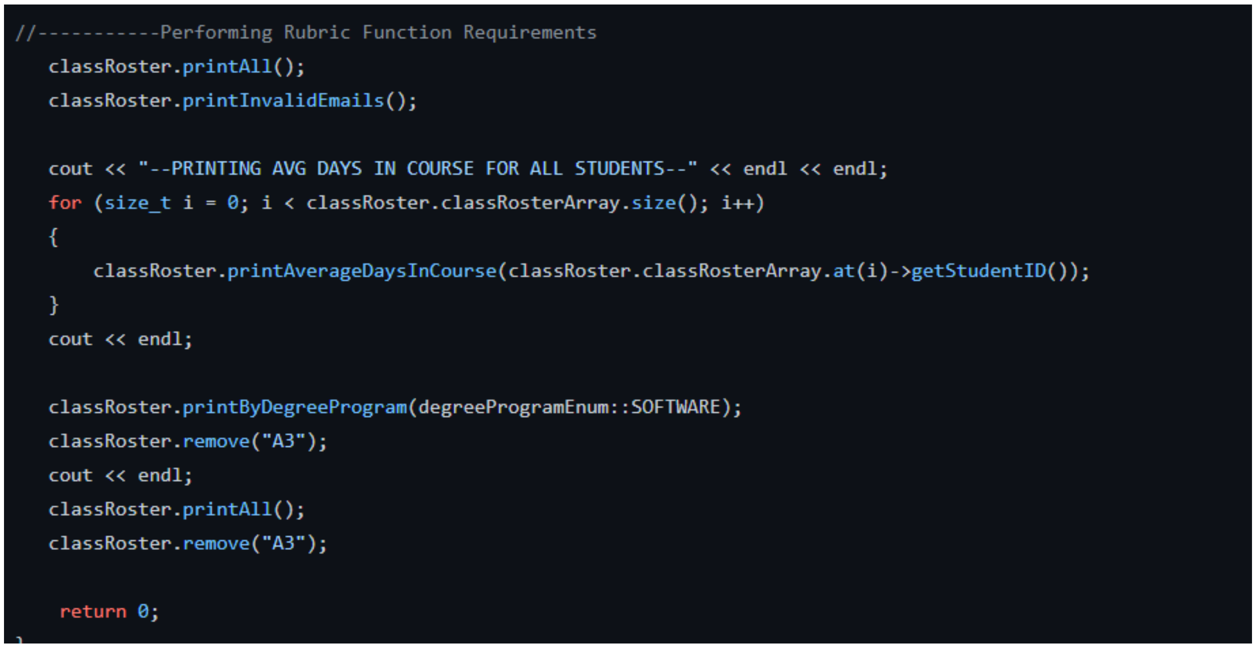
```
int main()
{
//----------- Personal Output information
    cout << RED << "Scripting and Programming - Applications – C867" << endl;
    cout << RED << "C++" << endl;
    cout << RED << "Dan Adams - 000970570" << RESET << endl << endl << endl;

//-----------Input Data
     const string studentData[] =
    { "A1,John,Smith,John1989@gm ail.com,20,30,35,40,SECURITY",
        "A2,Suzan,Erickson,Erickson_1990@gmailcom,19,50,30,40,NETWORK",
        "A3,Jack,Napoli,The_lawyer99yahoo.com,19,20,40,33,SOFTWARE",
        "A4,Erin,Black,Erin.black@comcast.net,22,50,58,40,SECURITY",
        "A5,Dan,Adams,DanAdamsHarder@Outlook.com,34, 21, 32, 44,SOFTWARE", };

     const size_t numStudents = sizeof(studentData) / sizeof(studentData[0]);

//-----------Creating and populating the Roster Class
   roster classRoster = inputStudentData(studentData, numStudents);
   cout << endl;
```

In the beginning personal details of the developer is print. Then there is a const string array called studentData. Since this is const we cannot change the data inside that array. Each student details is given as a one string and each attributes separated by a comma. Then there is a numStudents varable. Which holds the count of the elemnt in the array.

This get by this method. Sizeof will return a long integer with the bytes. (Total bytes of the array). And the sizeof(studentDat[0]) will give the bytes that associated for a one student. If we divide the total number of bytes by one elemnt it will given the number of elements in the array.

Then we need to create an object from the roster class. To hold that object classRoster variable is used.inputStudentData function will return a roster object that will be assigned to the roster object. (Impelemtation details will be below)

Then print a new lne



```
//-----------Performing Rubric Function Requirements
   classRoster.printAll();
   classRoster.printInvalidEmails();

   cout << "--PRINTING AVG DAYS IN COURSE FOR ALL STUDENTS--" << endl << endl;
   for (size_t i = 0; i < classRoster.classRosterArray.size(); i++)
   {
       classRoster.printAverageDaysInCourse(classRoster.classRosterArray.at(i)->getStudentID());
   }
   cout << endl;

   classRoster.printByDegreeProgram(degreeProgramEnum::SOFTWARE);
   classRoster.remove("A3");
   cout << endl;
   classRoster.printAll();
   classRoster.remove("A3");

    return 0;
```

Now the roster object (classRoster) is created and populated. (This is done by inputStudetData function).

Therefore now using the member function of the classRoster printall() we prin the all details of the student. Also invalid email student details is printed accorind to their implementation.

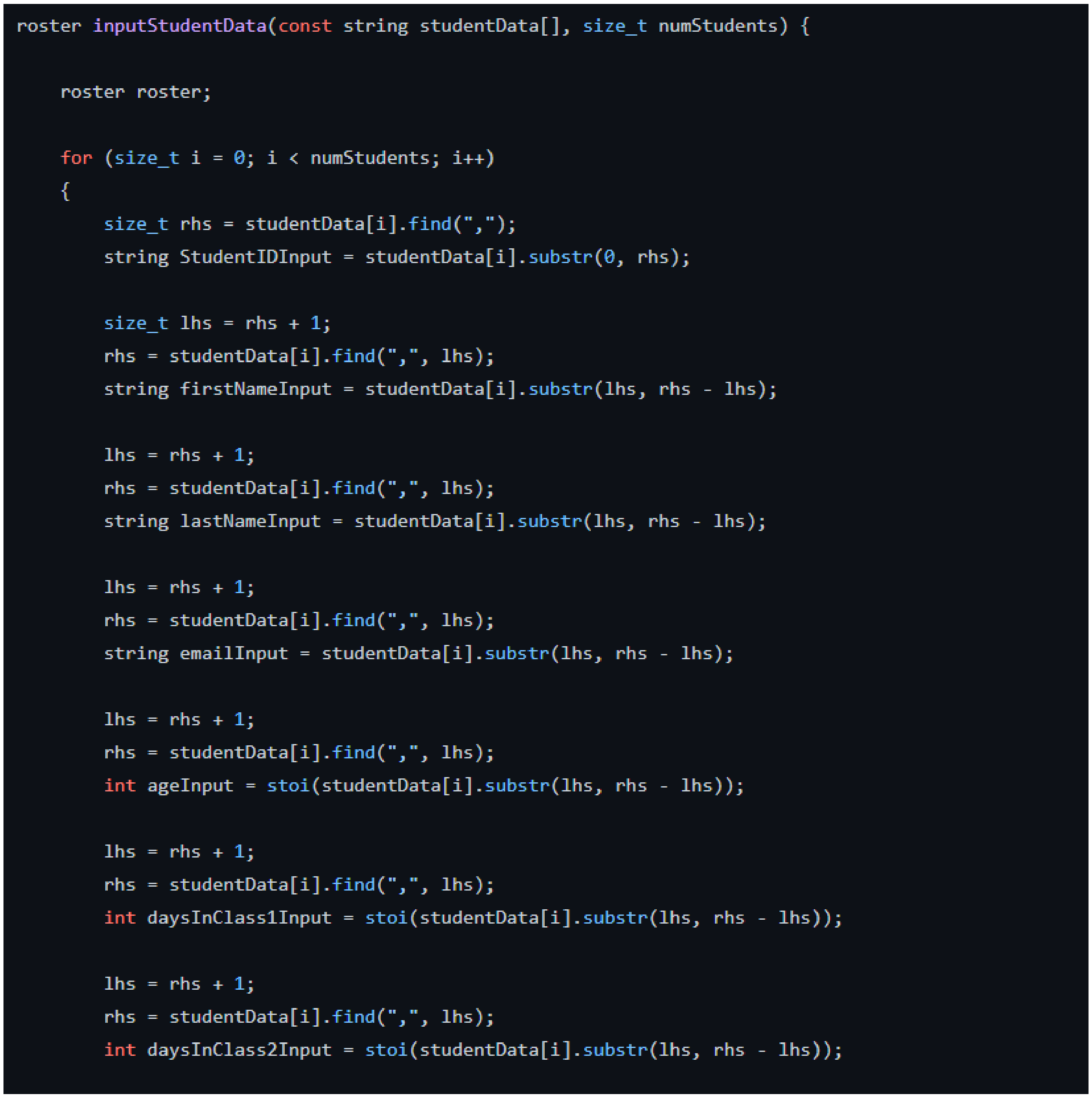
Then loop through th list of the classRoster and print averageDaysIncourse details using this function. And print the new line.

Then some print functions is tested in here.

Then remove the student from the list which has studentID “A3”

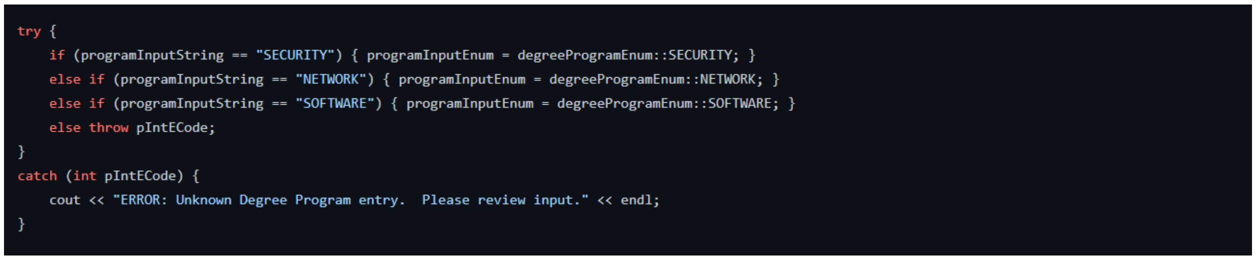
Then again call the list all students method to chceck whether A3 student si removed from the list.

Then again call remove function with same id. But since the student already removed it will print error message that declare in the remove function.



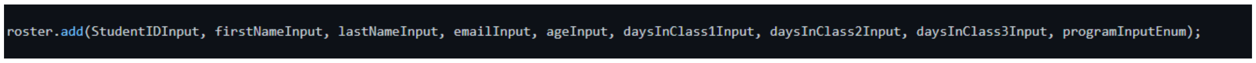
```
roster inputStudentData(const string studentData[], size_t numStudents) {

    roster roster;

    for (size_t i = 0; i < numStudents; i++)
    {
        size_t rhs = studentData[i].find(",");
        string StudentIDInput = studentData[i].substr(0, rhs);

        size_t lhs = rhs + 1;
        rhs = studentData[i].find(",", lhs);
        string firstNameInput = studentData[i].substr(lhs, rhs - lhs);

        lhs = rhs + 1;
        rhs = studentData[i].find(",", lhs);
        string lastNameInput = studentData[i].substr(lhs, rhs - lhs);

        lhs = rhs + 1;
        rhs = studentData[i].find(",", lhs);
        string emailInput = studentData[i].substr(lhs, rhs - lhs);

        lhs = rhs + 1;
        rhs = studentData[i].find(",", lhs);
        int ageInput = stoi(studentData[i].substr(lhs, rhs - lhs));

        lhs = rhs + 1;
        rhs = studentData[i].find(",", lhs);
        int daysInClass1Input = stoi(studentData[i].substr(lhs, rhs - lhs));

        lhs = rhs + 1;
        rhs = studentData[i].find(",", lhs);
        int daysInClass2Input = stoi(studentData[i].substr(lhs, rhs - lhs));
```

This is the implementation of the inputStudentData function. It initially loop through the studentData list that given from the argment. For each round it will get a string that contains the details of the student separated by comma. For each attribute it will find the postion of the comma. Then get attribute by devide the string to initial position to one lower than comma position. Then rhs is updated also lhs updated such that after the position comma is found. Likewise it will get the data by seprating comma and store in the relevant variables.



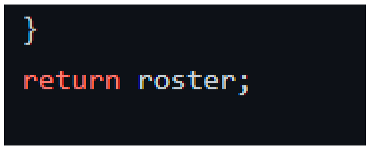
```
try {
    if (programInputString == "SECURITY") { programInputEnum = degreeProgramEnum::SECURITY; }
    else if (programInputString == "NETWORK") { programInputEnum = degreeProgramEnum::NETWORK; }
    else if (programInputString == "SOFTWARE") { programInputEnum = degreeProgramEnum::SOFTWARE; }
    else throw pIntECode;
}
catch (int pIntECode) {
    cout << "ERROR: Unknown Degree Program entry.  Please review input." << endl;
}
```

Then it will try to get the degree program by comparing each levels. This may generate the exception. To handle that we use a try block. If no error is get this will do the assigning. If error found it will jump to the catch block and generate error message.



```
roster.add(StudentIDInput, firstNameInput, lastNameInput, emailInput, ageInput, daysInClass1Input, daysInClass2Input, daysInClass3Input, programInputEnum);
```

If the all things in the each round is correct create an student object by extracted data and added to the array of the roster object. After that that roster object is reruen from the function. That object is assign the classRoster object in the main function.



```
}
return roster;
```