SCRIPTING & PROGRAMMING APPLICATIONS

Competencies:
4048.2.1: Introduction to Programming - *The graduate applies fundamental programming concepts in a specific programming environment.*
4048.2.2: Variables and Data Types - *The graduate prepares code which declares, initializes, and assigns values to variables of appropriate types as part of the application development process.*
4048.2.3: Control Structures - *The graduate writes code that implements decision and loop constructs to control the flow of a program.*
4048.2.4: Arrays - *The graduate creates arrays in order to solve complex problems.*
4048.2.5: Pointers and Memory -*The graduate applies pointers to solve complex problems.*
4048.2.6: Functions - *The graduate writes code that creates and manipulates functions and files.*
4048.2.7: Object-Oriented Paradigm - *The graduate applies object-oriented programming concepts in order to create a basic application.*

Task 1: Class Roster

Introduction:

Throughout your career in software development, you will develop and maintain new and existing applications. You will be expected to fix issues as well as add new enhancements or migrate existing applications to new platforms or different programming languages. As a software developer, your role will be to create a design of an application based on given business requirements. After the design is completed, you must implement the application based on the design document and provided requirements.

In this assessment, you will create a C++ application based on the scenario below. The skills you demonstrate in your completed application will be useful in responding to technical interview questions for future employment. This application may also be added to your portfolio to show to future employers.

This project will require an integrated development environment (IDE). You must use either Visual Studio or NetBeans for this assessment. Directions for accessing these IDEs can be found in the attached "IDE Instructions."

Scenario:

You are hired as a contractor to help a university migrate an existing student system to a new platform using C++ language. Since the application already exists, its requirements exist as well, and they are outlined in the next section. You are responsible for implementing the part of the system based on these requirements. A list of data is provided as part of these requirements. This part of the system is responsible for reading and manipulating the provided data.

You must write a program containing five classes (i.e., `Student`, `SecurityStudent`, `NetworkStudent`, `SoftwareStudent`, and `Roster`). The program will maintain a current roster of students within a given course. Student data for the program includes student ID, first name, last name, email address, age, an array of the number of days to complete each course, and degree. This information can be found in the studentData table below. The program will read a list of five students and use function calls to manipulate data (see part F4 in the requirements below). While parsing the list of data, the program should create student objects using the appropriate subclasses indicated by the degree program. The entire student list will be stored in one array of students called `classRosterArray`. Specific data-related output will be directed to the console.

studentData table

Student ID	**First Name**	**Last Name**	**Email**	**Age**	**Days in Course**	**Degree**
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	Your first name	Your last name	Your valid email address	Your age	Number of days to complete 3 courses	SOFTWARE

The data should be input as follows:

```
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,[firstname],[lastname],[emailaddress],[age],
[numberofdaystocomplete3courses],SOFTWARE"
```

You may not include third party libraries. Your submission should include one zip file with all the necessary code files to compile, support, and run your application. You must also provide evidence of the program's required functionality by taking a screen capture of the console run, saved as an image file or PDF.

Note: This assessment requires you to submit pictures, graphics, and/or diagrams. Each file must be an attachment no larger than 30 MB in size. Diagrams must be original and may be hand-drawn or drawn using a graphics program. Do not use CAD programs because attachments will be too large.

Requirements:

Your submission must be your original work. No more than a combined total of 30% of the submission and no more than a 10% match to any one individual source can be directly quoted or closely paraphrased from sources, even if cited correctly. Use the Turnitin Originality Report available in Taskstream as a guide for this measure of originality.

You must use the rubric to direct the creation of your submission because it provides detailed criteria that will be used to evaluate your work. Each requirement below may be evaluated by more than one rubric aspect. The rubric aspect titles may contain hyperlinks to relevant portions of the course.

Create a program that converts the array of strings found in the studentData table to an array of student objects by doing the following:

A. Modify the studentData table to include your personal information as the last item.

B. Create a C++ project in your integrated development environment (IDE) with the following files:
- degree.h
- student.h and student.cpp
- networkStudent.h and networkStudent.cpp
- securityStudent.h and securityStudent.cpp
- softwareStudent.h and softwareStudent.cpp
- roster.h and roster.cpp

Note: There must be a total of 11 source code files.

C. Define an enumerated data type *Degree* for the degree programs containing the following data elements *SECURITY*, *NETWORKING* and *SOFTWARE*.

Note: This information should be included in the degree.h file.

D. For the `Student` class, do the following:
1. Create the base class `Student` in the files student.h and student.cpp, which includes *each* the following variables:
   - student ID
   - first name

- last name
- email address
- age
- array of number of days to complete each course
- degree types

  Note: Degree type should be populated in subclasses only.

2. Create *each* of the following functions in the `Student` class:
   a. an accessor (i.e., getter) for *each* instance variable from part D1
   b. a mutator (i.e., setter) for *each* instance variable from part D1

   Note: All access and changes to the instance variables of the `Student` class should be done through the accessor and mutator functions.

   c. constructor using *all* of the input parameters provided in the table
   d. virtual `print()` to print specific student data
   e. destructor
   f. virtual `getDegreeProgram()`

   Note: Leave the implementation of the getDegreeProgram() function empty.

3. Create the **three** following classes as subclasses of `Student`, using the files created in part B:
   - `SecurityStudent`
   - `NetworkStudent`
   - `SoftwareStudent`

   Each subclass should override the `getDegreeProgram()` function. *Each* subclass should have a data member to hold the enumerated type for the degree program using the types defined in part C.

E. Create a `Roster` class (roster.cpp) by doing the following:
1. Create an array of pointers, `classRosterArray`, to hold the data provided in the studentData table.
2. Create a student object for *each* student in the data table by using the subclasses `NetworkStudent`, `SecurityStudent`, and `SoftwareStudent`, and populate `classRosterArray`.
   a. Apply pointer operations when parsing *each* set of data identified in the studentData table.
   b. Add *each* student object to `classRosterArray`.
3. Define the following functions:
   a. `public void add(string studentID, string firstName, string lastName, string emailAddress, int age, int daysInCourse1, int daysInCourse2, int daysInCourse3, <degree program>)` that sets the instance variables from part D1 and updates the roster
   b. `public void remove(string studentID)` that removes students from the roster by student ID. If the student ID does not exist, the function prints an error message indicating that the student was not found.

   c. `public void printAll()` that prints a complete tab-separated list of student data using accessor functions with the provided format: `1 [tab] First Name: John [tab] Last Name: Smith [tab] Age: 20 [tab]daysInCourse: {35, 40, 55} Degree Program: Security`. The `printAll()` function should loop through *all* the students in `classRosterArray` and call the `print()` function for *each* student.
   d. `public void printDaysInCourse(string studentID)` that correctly prints a student's average number of days in the three courses. The student is identified by the studentID parameter.
   e. `public void printInvalidEmails()` that verifies student email addresses and displays all invalid email addresses to the user

   Note: A valid email should include an at sign ('@') and period ('.') and should not include a space (' ').

   f. `public void printByDegreeProgram(int degreeProgram)` that prints out student information for a degree program specified by an enumerated type

F. Demonstrate the program's required functionality by adding a `void main()` function to roster.cpp, which will contain the required function calls to achieve the following results:

1. Print out to the screen, via your application, the course title, the programming language used, your student ID, and your name.
2. Create an instance of the `Roster` class called `classRoster`.
3. Add *each* student to `classRoster`.
4. Convert the following pseudo code to complete the rest of the `main()` function:

```
classRoster.printAll();
classRoster.printInvalidEmails();
//loop through classRosterArray and for each element:
classRoster.printAverageDaysInCourse(/*current_object's student id*/);
classRoster.printByDegreeProgram(SOFTWARE);
classRoster.remove("A3");
classRoster.remove("A3");
//expected: the above line should print a message saying such a student with this ID
was not found.
```

5. Call the destructor to release the `Roster` memory.

File Attachments:
IDE
Instructions
link opens
in new
window
Web Links:

1. **FPP Task 1 Rubric link opens in new window**