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.

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 Xcode for this assessment. Directions for accessing these IDEs can be found in the attached "IDE Instructions."

Your submission should include a zip file with all the necessary code files to compile, support, and run your application. The zip file submission must also keep the project file and folder structure intact for the Visual Studio IDE or Eclipse IDE.

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 two classes (i.e., `Student` and `Roster`). The program will maintain a current roster of students within a given course. Student data for the program include student ID, first name, last name, email address, age, an array of the number of days to complete each course, and degree program. 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. 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 Program
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.

Note: Each file must be an attachment no larger than 30 MB in size.

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. The originality report that is provided when you submit your task can be used as a guide.

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.

*Tasks may **not** be submitted as cloud links, such as links to Google Docs, Google Slides, OneDrive, etc., unless specified in the task requirements. All other submissions must be file types that are uploaded and submitted as attachments (e.g., .docx, .pdf, .ppt).*

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
- roster.h and roster.cpp
- main.cpp

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

C. Define an enumerated data type *DegreeProgram* for the degree programs containing the data type values *SECURITY, NETWORK,* and *SOFTWARE*.

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

D. For the `Student` class, do the following:
1. Create the class `Student` in the files student.h and student.cpp, which includes *each* of the following variables:
   - student ID
   - first name
   - last name
   - email address
   - age
   - array of number of days to complete each course
   - degree program
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
   c. All external access and changes to any instance variables of the `Student` class must be done using accessor and mutator functions.
   d. constructor using *all* of the input parameters provided in the table
   e. `print()` to print specific student data

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 and populate `classRosterArray`.
   a. Parse *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, DegreeProgram degreeprogram)` 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 in the provided format: `A1 [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 printAverageDaysInCourse(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(DegreeProgram 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 `main()` function in main.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 WGU 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*/);
```

Note: For the `current_object's student id`*, use an accessor (i.e., getter) for the* `classRosterArray` *to access the* `student id`.

```
classRoster.printByDegreeProgram(SOFTWARE);
classRoster.remove("A3");
classRoster.printAll();
classRoster.remove("A3");
//expected: the above line should print a message saying such a student with
this ID was not found.
```

5. Implement the destructor to release the memory that was allocated dynamically in `Roster`.

G. Demonstrate professional communication in the content and presentation of your submission.

File Restrictions

File name may contain only letters, numbers, spaces, and these symbols: ! - _ . * ' ( )
File size limit: 200 MB
File types allowed: doc, docx, rtf, xls, xlsx, ppt, pptx, odt, pdf, txt, qt, mov, mpg, avi, mp3, wav, mp4, wma, flv, asf, mpeg, wmv, m4v, svg, tif, tiff, jpeg, jpg, gif, png, zip, rar, tar, 7z

RUBRIC

A:PERSONAL INFORMATION

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
Personal information in the last item of the "studentData Table" is not provided.	Personal information in the last item of the "studentData Table" is incomplete.	Personal information in the last item of the "studentData Table" is complete.

B:C++ PROJECT

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
A C++ project in the IDE is not provided.	The C++ project is incorrectly created in the IDE or incorrectly uses the given files.	The C++ project is correctly created in the IDE and correctly uses the given files.

C:ENUMERATED DATA TYPE

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
The enumerated data type is not provided.	The enumerated data type is incorrectly defined.	The enumerated data type is correctly defined.

D1:CLASS `STUDENT`

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
The class `Student` is not provided.	The class `Student` is incorrectly created or incorrectly uses 1 or more of the given variables.	The class `Student` is correctly created by correctly including *each* of the given variables.

D2A:ACCESSOR

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
An accessor function in the `Student` class for *each* instance variable from part D1 is not provided.	The accessor function in the `Student` class for *each* instance variable from part D1 is not functional or is incomplete.	The accessor function in the `Student` class for *each* instance variable from part D1 is functional and is complete.

D2B:MUTATOR

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
A mutator function in the `Student` class for *each* instance variable from part D1 is not provided.	The mutator function in the `Student` class for *each* instance variable from part D1 is not functional or is incomplete.	The mutator function in the `Student` class for *each* instance variable from part D1 is functional or is complete.

D2C:ACCESSOR AND MUTATOR USAGE

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
External access and changes to instance variables of the `Student` class are done using neither accessor nor mutator functions.	Some external access or changes to instance variables of the `Student` class are not done using accessor or mutator functions.	External access and changes to all instance variables of the `Student` class are done using accessor and mutator functions.

D2D:CONSTRUCTOR

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
A constructor in the `Student` class is not provided or does not use *any* of the input parameters from the data table.	The constructor function in the `Student` class inaccurately uses 1 or more of the input parameters from the data table.	The constructor function in the `Student` class accurately uses *all* of the input parameters from the data table.

D2E:PRINTING SPECIFIC DATA

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
A `print()` function in the `Student` class is not provided.	The `print()` function in the `Student` class inaccurately	The `print()` function in the `Student` class accurately prints specific student data.

	prints specific student data.	

E1:ARRAY

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
An array is not provided.	The array of pointers created to hold the data provided in the "studentData Table" is incomplete or is incorrect.	The array of pointers created to hold the data provided in the "studentData Table" is complete and is correct.

E2:STUDENT OBJECT

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
A student object for *each* student in the data table is not provided.	The student object for *each* student in the `classRosterArray` is incorrectly populated.	The student object for *each* student in the `classRosterArray` is correctly populated.

E2A:PARSED DATA

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
The data is not parsed.	The data is parsed incorrectly.	The data is parsed correctly.

E2B:ADDING STUDENT OBJECTS TO ARRAY

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
Each student object is not added to the `classRosterArray`.	1 or more student objects are incorrectly added to the `classRosterArray`.	*Each* student object is correctly added to the `classRosterArray`.

E3A:ADD FUNCTION

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT

The `public void add(string studentID, string firstName, string lastName, string emailAddress, int age, int daysInCourse1, int daysInCourse2, int daysInCourse3, DegreeProgram degreeprogram)` function is not defined.	The `public void add(string studentID, string firstName, string lastName, string emailAddress, int age, int daysInCourse1, int daysInCourse2, int daysInCourse3, DegreeProgram degreeprogram)` function is incorrectly defined to set the instance variables from part D1 and update the roster.	The `public void add(string studentID, string firstName, string lastName, string emailAddress, int age, int daysInCourse1, int daysInCourse2, int daysInCourse3,DegreeProgram degreeprogram)` function is correctly defined to set the instance variables from part D1 and update the roster.

E3B:REMOVE FUNCTION

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
The `public void remove(string studentID)` that removes students from the roster by student ID is not defined.	The `public void remove(string studentID)` that removes students from the roster by student ID is incorrectly defined.	The `public void remove(string studentID)` that removes students from the roster by student ID is correctly defined.

E3C:PRINT ALL FUNCTION

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
The `public void printAll()` is not defined.	The `public void printAll()` prints a complete tab-separated list of student data. Or the `printAll()` incorrectly loops through 1 or more of the students in the student list or incorrectly calls the `print()` function for 1 or more students.	The `public void printAll()` prints a complete tab-separated list of student data. The `printAll()` correctly loops through *all* of the students in the student list and correctly calls the `print()` function for *each* student.

E3D:PRINT AVERAGE FUNCTION

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
A `public void printAverageDaysInCourse(string studentID)` that prints a student's average number of days in the 3 courses is	A `public void printAverageDaysInCourse(string studentID)` incorrectly prints a student's aver-	A `public void printAverageDaysInCourse(string studentID)` correctly prints a student's average number of days in the 3 courses

not defined. Identification of the student by the student-ID parameter is not provided.	age number of days in the 3 courses by student ID. The student is not correctly identified by the student-ID parameter.	by student ID. The student is correctly identified by the student-ID parameter.

E3E:PRINT INVALID EMAILS FUNCTION

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
A `public void printInvalidEmails()` is not defined.	A `public void printInvalidEmails()` incorrectly identifies student email addresses as valid.	A `public void printInvalidEmails()` correctly verifies student email addresses and displays *all* invalid email addresses to the user.

E3F:PRINT DEGREE PROGRAM FUNCTION

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
A `public void printByDegreeProgram(DegreeProgram degreeProgram)` is not provided.	A `public void printByDegreeProgram (DegreeProgram degreeProgram)` incorrectly prints out students for a degree program specified by an enumerated type.	A `public void printByDegreeProgram (DegreeProgram degreeProgram)` correctly prints out student information for a degree program specified by an enumerated type.

F1:SCREEN PRINT OUT

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
The course title, programming language used, WGU student ID, and student name are not *all* provided.	The course title, programming language used, WGU student ID, or student name is incorrectly printed at the top.	The course title, programming language used, WGU student ID, and student name are *all* correctly printed at the top.

F2:ROSTER CLASS INSTANCE

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
An instance of the `Roster` class called `classRoster` is not provided.	The instance of the `Roster` class called `classRoster` is incorrectly created.	The instance of the `Roster` class called `classRoster` is correctly created.

F3:ADD STUDENTS

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
Students are not added to `classRoster` .	1 or more students are missing from `classRoster` .	*All* students are added to `classRoster` .

F4:PSEUDO CODE CONVERSION

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
Evidence of converting the pseudo code to complete the rest of the `main()` function is not provided.	*Some* of the pseudo code is incorrectly converted to complete the rest of the `main()` function or is out of order.	*All* pseudo code is correctly converted to complete the rest of the `main()` function and is in order.

F5:DESTRUCTOR

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
Evidence of a destructor to release the `Roster` memory is not provided.	Evidence of a destructor to release the `Roster` memory is provided, but the destructor was incorrectly implemented.	The `Roster` memory is released by implementing the destructor.

G:PROFESSIONAL COMMUNICATION

NOT EVIDENT	APPROACHING COMPETENCE	COMPETENT
Content is unstructured, is disjointed, or contains pervasive errors in mechanics, usage, or grammar. Vocabulary or tone is unprofessional or distracts from the topic.	Content is poorly organized, is difficult to follow, or contains errors in mechanics, usage, or grammar that cause confusion. Terminology is misused or ineffective.	Content reflects attention to detail, is organized, and focuses on the main ideas as prescribed in the task or chosen by the candidate. Terminology is pertinent, is used correctly, and effectively conveys the intended meaning. Mechanics, usage, and grammar promote accurate interpretation and understanding.

SUPPORTING DOCUMENTS