CS 202 – Assignment #3 **Spring 2023**

Welcome to University of Science and Technology

(Academic Evaluation Report)

Purpose: Learn class inheritance, composition, multi-file class implementation, and make utility.
Points: 100

Assignment:

Design and implement four C++ classes- studentInfo, studentGradeDetails, studentType, and calculateFee. The assignment outcome is to generate an academic evaluation report of students' that provides a roadmap for completing their degree requirements. Each of the classes is described below.

Video Link for Assignment_3:
https://www.youtube.com/watch?v=qtOpjCGh6Yw

Image Source: https://acei-global.org/

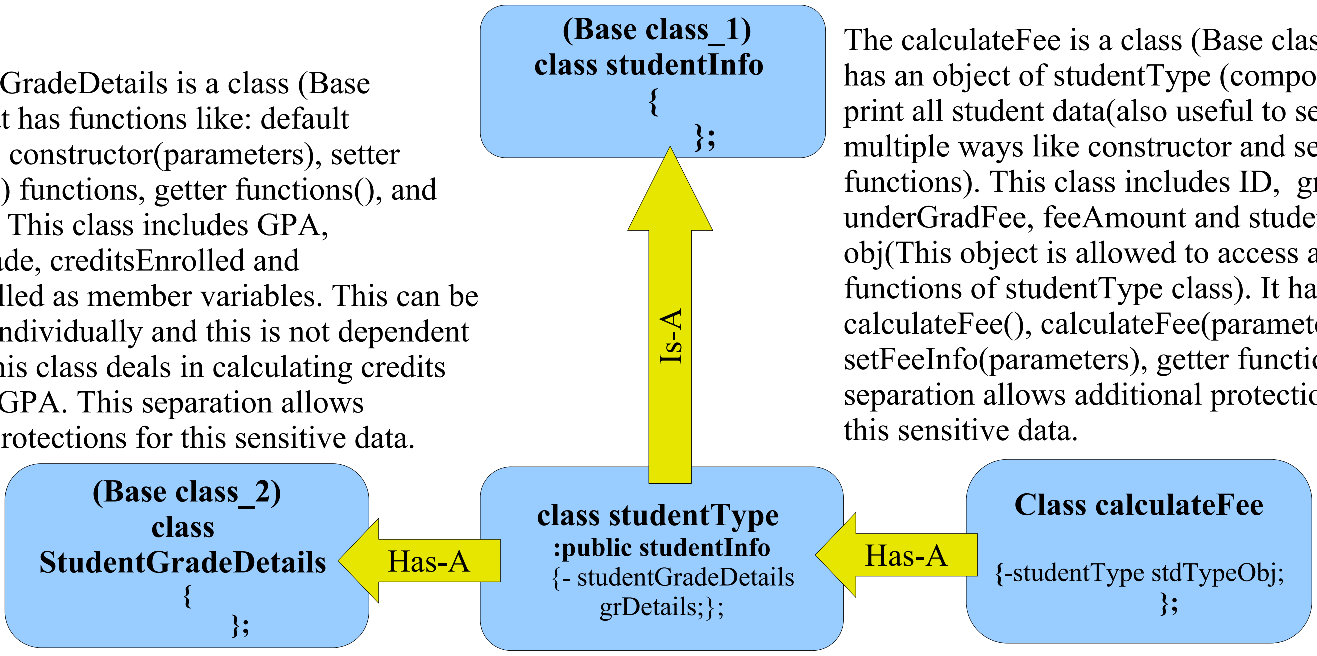
Class Hierarchy

The following diagram should help understand the class hierarchy for this project.

The studentInfo is a base class(Base class_1) encompasses basic student's information like first name, last name and birth month; also setter functions and a constructor with parameters.

The studentGradeDetails is a class (Base class_2) that has functions like: default constructor, constructor(parameters), setter (parameters) functions, getter functions(), and printData(). This class includes GPA, courseXGrade, creditsEnrolled and creditsFulfilled as member variables. This can be developed individually and this is not dependent on other. This class deals in calculating credits earned and GPA. This separation allows additional protections for this sensitive data.

The calculateFee is a class (Base class_3) that has an object of studentType (composition) to print all student data(also useful to setID in multiple ways like constructor and setter functions). This class includes ID, gradFee, underGradFee, feeAmount and studentType obj(This object is allowed to access all the functions of studentType class). It has calculateFee(), calculateFee(parameters), setFeeInfo(parameters), getter functions. This separation allows additional protections for this sensitive data.



A student type can be an undergraduate or graduate. It has direct relationship (inheritance: Is-A relationship) with the studentInfo. StudentType studentID, creditsRequired, creditstoGo, and studentGradeDetails obj (composition: Has-A relationship). StudentType constructor has parameters like ID, fName, lName, bMonth, letter grades-c1g, c2g, c3g, c4g, creditsTaken and creditsReqd.

So, by deriving the studentType class from the studentInfo class, it can use the studentInfo class data and functions to set fName, lName, and bMonth without having to re-write the code. With the studentGradeDetails grDetails(obj), it is easy to set the parameters like c1g, c2g, c3g, c4g, and creditsTaken with the grDetails(obj). It is only necessary to add the additional data and functions as needed to set other parameters like studentID, creditsRequired, creditstoGo using parameterized constructor and setter functions.

This class will calculateFee for each student based on their type like undergrad and grad. The fee is different for undergrad and grad students. Total fee is the product of creditstoGo with fee per credit of that student type.

The classes we will implement are as follows:

- **studentInfo Class**
  - Implement a basic ***studentInfo*** class(Base class_1), will track students basic information and provide standard support functions for all students. This will include first name, last name, and birth month. This class has a default constructor, parameterized constructor, setter and getter functions. This will be the *base class* as this will apply to any student (undergraduate or graduate). The studentType class will inherit this functionality, so this class must be fully working before working on the studentType.

- **studentGradeDetails Class**
  - Implement a class, **studentGradeDetails** (Base class_2), that contains letter grades for 4 courses, credits enrolled, credits fulfilled, and GPA as member variables. This class can set all above except credits fulfilled, by using setter functions or parameterized constructor. This class isolates the letter grades information of student. This can be tested independently of other class.

- **studentType Class**
  - This ***studentType*** class is derived(public inheritance) from ***studentInfo*** and sets the first name, last name and birth month of a student. This class has an object (grDetails; has-a relationship—composition) of ***studentGradeDetails.*** The grDetails object can access all the functions of studentGradeDetails and so is easy to calculateGPA, calculateCreditsEarned, setGradeDetails etc. This class also contains studentID, creditsRequired, and creditstoGo as member variables, which is useful to set additional details of that student. This class have parameterized constructor and setter functions to set all the essential data that can be useful to calculate like GPA, how many credits yet to go. This class has getter functions that are useful to get the results like ID, credits required (separate for an undergrad and grad student), etc. The other major function in this class is to checkStudentID(string idTemp). To set the studentID, there are certain rules and if they meet the conditions, then the studentID is set accordingly else print error messages accordingly. Check the sample output for more details.

- **calculateFee Class**
  - Implement a small class, ***calculateFee*** class(Base class_3), that contains ID, gradFee, underGradFee, feeAmount, and studentType stdTypeObj(This object is useful to access print function of studentType) as member variables. This class sets the above parameters using setter functions or parameterized constructor. This class has calculateFee(string id, studentType stdObj) and **void** setFeeInfo(string id, studentType stdObj)to calculate fee according to the number of credits required and fee amount of the student like grad or underGrad. This class has **void** printAllStudentData() function to print the stdTypeObj.printStudentTypeData()and other details like credits to go and feeAmount. Check the sample output for more details.

Inheritance, composition and fundamental concepts are important in object oriented programming and it is essential to fully understand. Refer to the text and class lectures for additional information regarding inheritance and composition.

<u>Development and Testing</u>
In order to simplify development and testing, the project is split into four parts: *studentInfo, studentGradeDetails, studentType,* and *calculateFee* classes.

- The ***studentInfo*** can be developed first and this class can be tested independently of the other class. This class sets firstName, lastName, and birthMonth.

- The ***studentGradeDetails*** must be developed next (before ***studentType***). This class can be tested independently of the other class.

- The ***studentType*** must be developed next. This class is inherited (public) from ***studentInfo.*** The ***studentType*** class has parameters like ID, first name, last name, birth month, letter grades of course1, course2, course3, course4; credits taken and credits required. The above parameters are set using studentInfo functions and the other functions with the object (grDetails) of studentGradeDetails (composition).
- The ***calculateFee*** must be developed last as this has the studentType stdTypeObj (composition). The ***calculateFee*** parameterized constructor and **void** setFeeInfo(string id, studentType stdObj) are the major functions in this class, where those functions are used to set all parameters accordingly. This function prints the fee amount only if the ID of this class matches with the studentID of ***studentType***.

Submission:

- All files must compile and execute on Ubuntu and compile with C++11.

- Submit source files
  - Submit a copy of the **source** files via the on-line submission.
  - *Note*, do ***not*** submit the provided mains (we have them).

- Once you submit, the system will score the project and provide feedback.
  - If you do not get full score, you can (and should) correct and resubmit.
  - You can re-submit an unlimited number of times before the due date/time.

- Late submissions will be accepted for a period of 24 hours after the due date/time for any given lab. Late submissions will be subject to a ~2% reduction in points per an hour late. If you submit 1 minute - 1 hour late -2%, 1-2 hours late -4%, … , 23-24 hours late -50%. This means after 24 hours late submissions will receive an automatic 0(zero).

Program Header Block

All program and header files must include your name, section number, assignment, NSHE number, input and output summary. The required format is as follows:

```
/*
Name: MY_NAME, NSHE, CLASS-SECTION, ASSIGNMENT
Description: <per assignment>
Input: <per assignment>
Output: <per assignment>
*/
```

Failure to include your name in this format will result in a loss of up to 5%.

Code Quality Checks

A C++ linter[1] is used to perform some basic checks on code quality. These checks include, but are not limited to, the following:

- Unnecessary 'else' statements should not be used.
- Identifier naming style should be either camelCase or snake_case (consistently chosen).
- Named constants should be used in place of literals.
- Correct indentation should be used.
- Redundant return/continue statements should not be used.
- Selection conditions should be in the simplest form possible.
- Function prototypes should be free of top level *const*.

1 For more information, refer to: https://en.wikipedia.org/wiki/Lint_(software)

Not all of these items will apply to every program. Failure to to address these guidelines will result in a loss of up to 5%.

studentInfo Class

Implement a basic *studentInfo* class to provide basic information of a student. The *studentInfo* UML class diagram is as follows:

studentInfo
-lastName, firstName: string
-birthMonth: int
+studentInfo()
+studentInfo(string fName, string lName, int bMonth)
+setName(string first, string last): void
+getFirstName() const: string
+getLastName() const: string
+print() const: void

Function Descriptions:

The following are more detailed descriptions of the required functions.

- studentInfo() is a default constructor that initializes firstName, lastName and birthMonth.
- studentInfo(string fName, string lName, int bMonth); This is a parameterized constructor is used to initialize the member variables according to the parameters in the constructor body.
- **void** setName(string first, string last); This function is to set firstName and lastName according to the parameters.
- string getFirstName() **const**; This function is to return the first name.
- string getLastName() **const**; This function is to return the last name.
- **void** print() **const**; This function is to output the first name and last name in the form firstName lastName and birthMonth in next line.

studentGradeDetails Class

Implement the Base class_2 named ***studentGradeDetails*** to provide student academic details like letter grades, credits enrolled, credits fulfilled and GPA. The *studentGradeDetails* UML class diagram is as follows:

studentGradeDetails
-GPA: double
-creditsEnrolled: int
-creditsFulfilled: int
-c1Grade: char
-c2Grade: char
-c3Grade: char
-c4Grade: char
+studentGradeDetails()
+studentGradeDetails(char, char, char, char, int)
+setGradeDetails(char, char, char, char, int): void
+calculateGPA(char, char, char, char): void
+calculateCreditsEarned(char, char, char, char): void

`+getGPA() const: double`
`+getCreditsEarned() const: int`
`+printData() const: void`

Function Descriptions:
The following are more detailed descriptions of the required functions.

- studentGradeDetails() is a default constructor to initialize all the member variables
- studentGradeDetails(**char** c1g, **char** c2g, **char** c3g, **char** c4g, **int** credsTaken) is a parameterized constructor to initialize the member variables according to the parameters in the constructor body.
- **void** setGradeDetails(**char** c1g, **char** c2g, **char** c3g, **char** c4g, **int** credsTaken) is to set the member variables according to the parameters in the constructor body.
- **void** calculateGPA(**char** c1g, **char** c2g, **char** c3g, **char** c4g) This function should calculate GPA by considering the letter grades of c1Grade, c2Grade, c3Grade, and c4Grade. You can use the following mathematical equation to calculate GPA. Any letter grade greater than F receive 0 grade points for that course.

  GPA=Total Grade Points of 4 courses/Total credits of 4 courses
  For example, if the letter grades of the 4 courses are like 'A', 'C', 'B', 'F'
  Then GPA = (12+6+9+0)/(3+3+3+3)=27/12=2.25

Credits per Course	Letter Grade	Grade Points / Credit	Total Grade Points for each course
3	A	4	4*3=12
3	B	3	3*3=9
3	C	2	2*3=6
3	D	1	1*3=3
3	F	0	0*3=0

- **void** calculateCreditsEarned(***char*** c1g, **char** c2g, **char** c3g, **char** c4g*)* This function calculates the credits earned based on the letter grades. The letter grades >= D should not be considered as fulfilled credits.
  For example, if the letter grades of the 4 courses are like 'A', 'C', 'D', 'F'
  Then credits earned for A and C only, that is 6 credits in total.
- **double** getGPA() **const** return GPA.
- **int** getCreditsEarned() **const** return creditsFulfilled.
- **void** printData() **const** prints letter grades, credits enrolled, and GPA.

studentType Class

studentType is inherited from ***studentInfo (is-a relationship)*** and has an object of ***studentGradeDetails (has-a relationship).*** The ***studentType*** UML class diagram is as follows:

studentType
`-studentID: string`
`-creditsRequired: int`
`-creditstoGo: int`
`-grDetails: studentGradeDetails`
`+studentType()`

`+studentType(string, string, string, int, char, char, char, char, int, int)`
`+setStudentDetails(string, string, string, int, char, char, char, char, int, int)`
`+setStudentID(string): void`
`+getStudentID() const: string`
`+getCreditsRequired() const: int`
`+calculateCreditstoGo(): void`
`+getCreditstoGo() const: int`
`+checkStudentID(string idTemp) const: bool`
`+printStudentTypeData() const: void`

<u>Function Descriptions:</u>

The following are more detailed descriptions of the required functions.

- studentType() is a default constructor to initialize all the member variables
- studentType(string ID, string fName, string lName, **int** bMonth, **char** c1g, **char** c2g, **char** c3g, **char** c4g, **int** credsTaken, **int** credsReqd) is a parameterized constructor to initialize the member variables according to the parameters in the constructor body.
- **void** setStudentDetails(string ID, string fName, string lName, **int** bMonth, **char** c1g, **char** c2g, **char** c3g, **char** c4g, **int** credsTaken, **int** credsReqd) is to set the member variables according to the parameters in the constructor body.
- **void** setStudentID(string ID) is to set the studentID. In this function print stars using the following instructions:

  ```
  string stars;
  stars.append(65, '*');
  cout << stars << endl;
  ```

  This function has to print “Given Student ID: ” and validate ID using checkStudentID(ID) and if the ID is invalid, print "Error: " << ID << " is Invalid StudentID "
- **string** getStudentID() **const** return StudentID
- **int** getCreditsRequired() **const** return creditsRequired
- **void** calculateCreditstoGo() subtract credits earned from credits required and assign the result to the member variable creditstoGo.
- **int** getCreditstoGo() **const** return creditstoGo
- **bool** checkStudentID(string idTemp) **const** is to check studentID based on the following conditions:
  - i. if the idTemp is less than 7 characters print “Invalid ID - Less than 7 characters”
  - ii. The first 3 characters should be alphabets else print "Invalid Alphabets ID"
  - iii. The 4th and 5th characters should be numbers else print "Invalid Digits ID"
  - iv. The last 2 characters should be alphabets and they should be a combination of ug (undergraduate) or gr (graduate) else print "Invalid Last 2 Characters ID"
  - v. if the above statements are valid return true else false
- **void** printStudentTypeData() **const**- Firstly, it prints stars and dashes(check sample output). If the student is undergraduate(based on last two characters of ID) print “Under Graduate Student”; ff the student is graduate print “Graduate Student”; then print studentID; print the student details like first name, last name, and birth month; print letter grades, credits enrolled, GPA; print Credits Required, Credits Fulfilled, Credits to Go and finally dashes.

  Note: Following my Instructions on printing stars and dashes is not mandatory. However, it is essential to match your output with the example output. You can move those stars and dashes to any function according to your logic.

calculateFee Class

The ***calculateFee*** has an object of studentType (has-a relationship). The *calculateFee* UML class diagram is as follows:

calculateFee
-ID: string
-gradFee: double
-underGradFee: double
-feeAmount: double
-stdTypeObj: studentType
+calculateFee()
+calculateFee (string, studentType)
+setFeeInfo(string, studentType): void
+setID(string): void
+getID(): string
+getFeeAmount(): double
+printAllStudentData()const: void

Function Descriptions:

The following are more detailed descriptions of the required functions.

- calculateFee() is a default constructor to initialize all the member variables.
- calculateFee (string **id**, studentType stdObj) is a parameterized constructor to initialize the member variables according to the parameters in the constructor body. The studentType stdObj (studentType stdObj(parameters)- parameterized constructor) is passed as a 2nd parameter in *calculateFee* parameterized constructor.
  In this constructor, if the student is graduate(based on the last two characters of ID, for example gr) then feeAmount = creditstoGo * gradFee. Similarly, calculate feeAmount for underGraduate student.
- **void** setFeeInfo(string **id**, studentType stdObj) is to set the member variables according to the parameters in the constructor body. The obj of studentType is passed as a 2nd parameter in this parameterized constructor. In this setter function, if the student is graduate(based on the last two characters of ID, for example gr) then feeAmount = creditstoGo * gradFee. Similarly, calculate feeAmount for underGraduate student.
- **void** setID(string **id**) is to set ID according to the parameter. In this function, you no need to check the validation of ID.
- string getID() **const** return ID.
- **double** getFeeAmount() **const** return feeAmount.
- **void** printAllStudentData()- Firstly, print stars. Then compare the ID of this class with the ID of studentType. If they are not matched print "ID Mismatch or Invalid ID is passed; ID from calculateFee is: " << ID; else print the data according to the type of student (graduate or undergraduate). First print the data with the object of studentType, then uderGradFee or gradFee per credit; then finally FeeAmount. Please check the example output for more details.

The *studentInfo* and *studentGradeDetails* classes must be fully completed prior to *studentType* and *calculateFee* classes. Refer to the example executions for formatting. Make sure your program includes the appropriate documentation.

Note: Your implementation files should be fully commented.

Files Provided:
The following files are available to download for this assignment.

- makefile.
- main.cpp.
- studentInfo.h
- studentGradeDetails.h
- studentType.h
- calculateFee.h
- CS202_Assignment3_Sp23 (Instruction pdf)
- CS202_Ast3_Sample_Output_Sp23

Submission:
Submit the following files for this assignment.

- studentInfoImp.cpp
- studentGradeDetailsImp.cpp
- studentTypeImp.cpp
- calculateFeeImp.cpp

How to Compile?
The provided make file assumes the source file names are as described. If you change the names, the make file will need to be edited. To build the given classes, one main provided and it has good and bad data. To compile, simply type:

- `make` (Which will create the *main* for executables respectively.)

How to Run your code?
Use the following commands to run.

- ./main

Example: How the composition Works?

From ***studentType*** class
studentType(string ID, string fName, string lName, **int** bMonth, **char** c1g, **char** c2g, **char** c3g, **char** c4g, **int** credsTaken, **int** credsReqd);

In above parameters- fName, lName, and bMonth can be set by using ***studentInfo*** setter functions as ***studentType*** is directly inherited from ***studentInfo*** class.
char c1g, **char** c2g, **char** c3g, **char** c4g, **int** credsTaken can be set from the object (composition) of ***studentGradeDetails***
string ID and credsReqd should be initialized according to the parameters in constructor body.

From ***calculateFee*** class
calculateFee (string **id**, studentType stdObj);

calculateFee class is having an object of studentType(composition). So this stdObj can access the functions related to ***studentType***
From main.cpp
Line 1: studentType Garcia("Kar03gr", "Karen", "Garcia", 11, 'C', 'A', 'A', 'C', 12, 30);
Line 2: calculateFee GarciaFee("Kar03gr", Garcia);

In Line 1, Garcia is an object of studentType and is also a parameterized constructor. You pass that Garcia as a parameter to Line 2 in calculateFee parameterized constructor. With Garcia, you can access all the functions of ***studentType***.

Example Execution:

(Note: I highlighted some example output in color to emphasize errors and the function calls needed to print that information)

Your original text color for output should be black only.

Below is an example program execution output for this Assignment. Don't make any changes in header files and main.cpp. Your code/logic is acceptable only in implementation files.

Note: Below example execution format may have few extra white spaces on few pages. To show the output clean, I added a few white spaces in this example output. The codeGrade ignores the whitespace and so no need to be worried on that. Please check the example execution text file that shows expected formatting style.

```
kishore@kishore-virtual-machine:/CS202_Ast3_Sp23_Ubuntu$ make
g++ -Wall -Wextra -pedantic -std=c++11 -g -c main.cpp
g++ -Wall -Wextra -pedantic -std=c++11 -g -c studentInfoImp.cpp
g++ -Wall -Wextra -pedantic -std=c++11 -g -c studentGradeDetailsImp.cpp
g++ -Wall -Wextra -pedantic -std=c++11 -g -c studentTypeImp.cpp
g++ -Wall -Wextra -pedantic -std=c++11 -g -c calculateFeeImp.cpp
g++ -Wall -Wextra -pedantic -std=c++11 -g -o main main.o studentInfoImp.o
studentGradeDetailsImp.o studentTypeImp.o calculateFeeImp.o
kishore@kishore-virtual-machine:/CS202_Ast3_Sp23_Ubuntu$ ./main

######################################################################
          Welcome to University of Science and Technology
######################################################################

*****************************************************************
Given Student ID: you11ug


__________________________________________________
Under Graduate Student:
Student ID: you11ug
Name: Mark Young
Birth Month: 11

Letter Grades: A B C A
Credits Enrolled: 12
GPA: 3.25

Credits Required: 120
Credits Fulfilled: 12
Credits to Go: 108
__________________________________________________

*****************************************************************
Given Student ID: mil08gr


__________________________________________________
Graduate Student:
Student ID: mil08gr
Name: David Miller
Birth Month: 08

Letter Grades: B B B B
Credits Enrolled: 12
GPA: 3

Credits Required: 26
Credits Fulfilled: 12
Credits to Go: 14
__________________________________________________
```

```
*****************************************************************
Given Student ID: whi04uu       →  studentType::setStudentID(string ID)
Invalid Last 2 Characters ID  → studentType::checkStudentID(string idTemp)
Error: whi04uu is Invalid StudentID
Invalid Birth Month  → studentInfo::setStudentInfo(string fName, string lName, int bMonth)

__________________________________________________
Graduate Student:
Student ID:      → studentType::printStudentTypeData() const
Name: Lisa White
Birth Month: 00  →  studentInfo::print()

Letter Grades: C A B C
Credits Enrolled: 12
GPA: 2.75        → grDetails.printData()

Credits Required: 124
Credits Fulfilled: 12
Credits to Go: 112  →  studentType::printStudentTypeData() const
__________________________________________________
```

```
studentType White("whi04uu", "Lisa", "White", -04, 'C', 'A', 'B', 'C', 12, 124);
   White.printStudentTypeData(); //Invalid last 2 chars, studentID, birthMonth
```

Note: The highlighted formatted output is from the White.printStudentTypeData();

```
*****************************************************************
Given Student ID: JAM01ug

__________________________________________________
Under Graduate Student:
Student ID: JAM01ug
Name: Nicholas James
Birth Month: 01

Letter Grades: A A A A
Credits Enrolled: 12
GPA: 4

Credits Required: 120
Credits Fulfilled: 12
Credits to Go: 108
__________________________________________________

*****************************************************************
Given Student ID: Wil02gr

__________________________________________________
Graduate Student:
Student ID: Wil02gr
Name: John Williams
Birth Month: 02

Letter Grades: B C C C
Credits Enrolled: 12
GPA: 2.25

Credits Required: 30
Credits Fulfilled: 12
Credits to Go: 18
__________________________________________________

*****************************************************************
Given Student ID: CLA04gr

__________________________________________________
Graduate Student:
Student ID: CLA04gr
Name: Anthony Clark
Birth Month: 04

Letter Grades: A A C B
Credits Enrolled: 12
GPA: 3.25
```

```
Credits Required: 32
Credits Fulfilled: 12
Credits to Go: 20
__________________________________________________

*****************************************************************
Given Student ID: Kar03gr

__________________________________________________
Graduate Student:
Student ID: Kar03gr
Name: Karen Garcia
Birth Month: 11

Letter Grades: C A A C
Credits Enrolled: 12
GPA: 3

Credits Required: 30
Credits Fulfilled: 12
Credits to Go: 18
__________________________________________________

*****************************************************************
Given Student ID: Sha9gr
Invalid ID - Less than 7 characters
Error: Sha9gr is Invalid StudentID

__________________________________________________
Graduate Student:
Student ID:
Name: Janet Shaw
Birth Month: 09

Letter Grades: B B D B
Credits Enrolled: 12
GPA: 2.5

Credits Required: 30
Credits Fulfilled: 9
Credits to Go: 21
__________________________________________________

*****************************************************************
Given Student ID: 07WEAgr
Invalid Alphabets ID
Invalid Digits ID
Invalid Grade Entry

__________________________________________________
Graduate Student:
Student ID: 07WEAgr
Name: Noah Weaver
Birth Month: 07

Letter Grades: A K B B
Credits Enrolled: 12
GPA: 0

Credits Required: 120
Credits Fulfilled: 9
Credits to Go: 111
__________________________________________________


######################################################################

Using Setter Function to
Fix Previous Invalid Entries
######################################################################
```

```
****************************************************************
Given Student ID: whi04ug

__________________________________________________
Under Graduate Student:
Student ID: whi04ug
Name: Lisa White
Birth Month: 04

Letter Grades: C A B C
Credits Enrolled: 12
GPA: 2.75

Credits Required: 124
Credits Fulfilled: 12
Credits to Go: 112
__________________________________________________

****************************************************************
Given Student ID: Sha09gr

__________________________________________________
Graduate Student:
Student ID: Sha09gr
Name: Janet Shaw
Birth Month: 09

Letter Grades: B B D B
Credits Enrolled: 12
GPA: 2.5

Credits Required: 30
Credits Fulfilled: 9
Credits to Go: 21
__________________________________________________


######################################################################

Default Constructor and Setter Function

######################################################################

****************************************************************
Given Student ID: gom05ug

__________________________________________________
Under Graduate Student:
Student ID: gom05ug
Name: Emily Gomez
Birth Month: 05

Letter Grades: A A F B
Credits Enrolled: 12
GPA: 2.75

Credits Required: 120
Credits Fulfilled: 9
Credits to Go: 111
__________________________________________________

****************************************************************
Given Student ID: ort11gr
Inavlid Credits Enrolled Entry
Invalid Grade Entry

__________________________________________________
Graduate Student:
Student ID: ort11gr
```

```
Name: Laura Ortiz
Birth Month: 11

Letter Grades: B F G B
Credits Enrolled: 0
GPA: 0

Credits Required: 34
Credits Fulfilled: 6
Credits to Go: 28
__________________________________________________

*****************************************************************
Given Student ID: jer04gr
Invalid Birth Month
Inavlid Credits Enrolled Entry
Invalid Grade Entry

__________________________________________________
Graduate Student:
Student ID: jer04gr
Name: Jerry Webb
Birth Month: 00

Letter Grades: F F Z D
Credits Enrolled: 0
GPA: 0

Credits Required: 30
Credits Fulfilled: 0
Credits to Go: 30
__________________________________________________


######################################################################

Calculate Fee for Under Graduate and Graduate Students

######################################################################

*****************************************************************
                                  ➡ calculateFee::printAllStudentData() const
__________________________________________________
Graduate Student:
Student ID: Kar03gr
Name: Karen Garcia
Birth Month: 11       ➡ Check previous color coding instructions

Letter Grades: C A A C
Credits Enrolled: 12
GPA: 3

Credits Required: 30
Credits Fulfilled: 12
Credits to Go: 18
__________________________________________________

Graduate Fee per credit: $ 660.00
FeeAmount for 18 credits is: $ 11880.00  ➡ calculateFee::printAllStudentData() const

*****************************************************************
ID Mismatch or Invalid ID is passed; ID from calculateFee is: gomo5ug

*****************************************************************
ID Mismatch or Invalid ID is passed; ID from calculateFee is: CLA03gr
                                  ➡ calculateFee::printAllStudentData() const
```

```
calculateFee GarciaFee("Kar03gr", Garcia);
   GarciaFee.printAllStudentData();
```

Note: The highlighted formatted output is from the GarciaFee.printAllStudentData();

```
*****************************************************************

__________________________________________________
Graduate Student:
Student ID: ort11gr
Name: Laura Ortiz
Birth Month: 11

Letter Grades: B F G B
Credits Enrolled: 0
GPA: 0.00

Credits Required: 34
Credits Fulfilled: 6
Credits to Go: 28
__________________________________________________

Graduate Fee per credit: $ 660.00
FeeAmount for 28 credits is: $ 18480.00

*****************************************************************

__________________________________________________
Graduate Student:
Student ID: jer04gr
Name: Jerry Webb
Birth Month: 00

Letter Grades: F F Z D
Credits Enrolled: 0
GPA: 0.00

Credits Required: 30
Credits Fulfilled: 0
Credits to Go: 30
__________________________________________________

Graduate Fee per credit: $ 660.00
FeeAmount for 30 credits is: $ 19800.00

*****************************************************************

__________________________________________________
Under Graduate Student:
Student ID: you11ug
Name: Mark Young
Birth Month: 11

Letter Grades: A B C A
Credits Enrolled: 12
GPA: 3.25

Credits Required: 120
Credits Fulfilled: 12
Credits to Go: 108
__________________________________________________

underGradFee per credit: $ 456.00
FeeAmount for 108 credits is: $ 49248.00

*****************************************************************

__________________________________________________
Under Graduate Student:
Student ID: JAM01ug
Name: Nicholas James
Birth Month: 01

Letter Grades: A A A A
Credits Enrolled: 12
```

```
GPA: 4.00

Credits Required: 120
Credits Fulfilled: 12
Credits to Go: 108
__________________________________________________

underGradFee per credit: $ 456.00
FeeAmount for 108 credits is: $ 49248.00

######################################################################
kishore@kishore-virtual-machine:/CS202_Ast3_Sp23_Ubuntu$
```