CS 202 – Assignment #3 **Fall 2024**

Welcome to University of Science and Engineering

(Tuitions and Fee Estimator)

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

Assignment:

Design and implement four C++ classes- studentDetails, studentType, calculator, and fileHandler. The assignment outcome is to manage student data, including personal information and fee calculations based on their degree and residency status. The system consists of several classes, each responsible for different aspects of student management. The goal is to generate an Estimated Fee report of students' to an output text file and also print on the console window in order to complete their degree requirements. Each of the classes is described below.

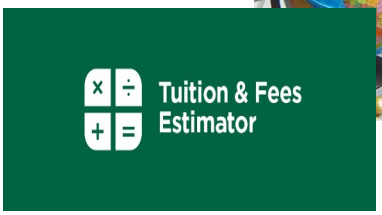


Image Source_2: https://www.algonquincollege.com/ro/pay/fee-estimator/
Image Source_3: https://international.unime.it/study-us/fees-and-fundings

Video Link for Assignment_3: https://unlv.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=3f7dbc35-bafb-4cd7-a21b-b1e600511d34

Classes and Their Responsibilities

studentDetails (Base Class)

Purpose: Manages basic student details such as name, birth year, and year enrolled.

Methods:

- Constructors to initialize student details.
- Getter methods to retrieve individual details.
- Method to print student details.

studentType (Derived Class)

Purpose: Inherits from studentDetails and adds more specific information such as student ID, department, degree, residency, and credits required.

Methods:

- Constructors to initialize a complete set of student data.
- Setter methods to update student information.
- Getter methods to retrieve specific attributes.
- Method to check if the student ID is valid.
- Method to print all student data.

calculator (Composition)

Purpose: Calculates tuition fees based on residency status and degree type.

Methods:

- calculateTuition: Computes the tuition fee for a given student.
- printAllStudentData: Displays student data.

fileHandler (Composition)

Purpose: Handles reading from and writing to files for student data and fee calculations.

Methods:

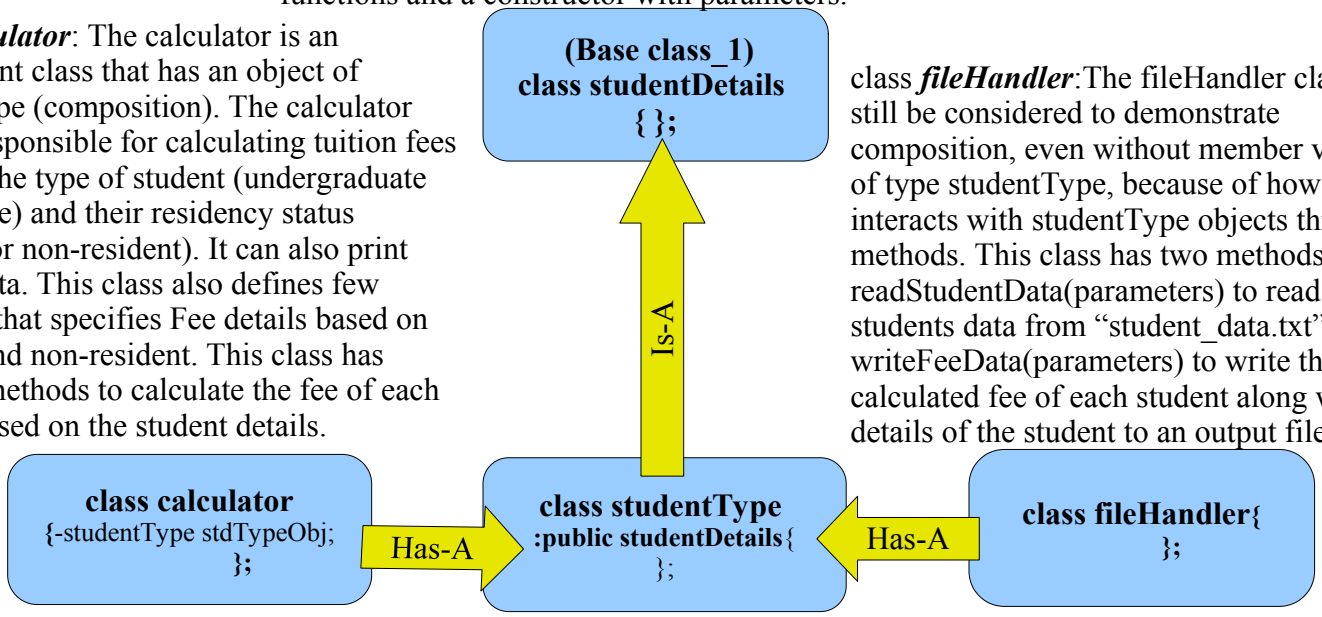
- readStudentData: Reads student data from a specified file and returns a vector of studentType objects.
- writeFeeData: Writes calculated fees for students to a specified file.

Class Hierarchy

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

class ***studentDetails***:The studentDetails is a base class(Base class) encompasses basic student's information like first name, last name, birth year, and year the student enrolled; also setter functions and a constructor with parameters.

class ***calculator***: The calculator is an independent class that has an object of studentType (composition). The calculator class is responsible for calculating tuition fees based on the type of student (undergraduate or graduate) and their residency status (resident or non-resident). It can also print student data. This class also defines few constants that specifies Fee details based on resident and non-resident. This class has multiple methods to calculate the fee of each student based on the student details.



class ***fileHandler***:The fileHandler class can still be considered to demonstrate composition, even without member variables of type studentType, because of how it interacts with studentType objects through its methods. This class has two methods- readStudentData(parameters) to read the students data from "student_data.txt" and writeFeeData(parameters) to write the calculated fee of each student along with the details of the student to an output file.

class ***studentType***: A student type can be an undergraduate or graduate. It has direct relationship (inheritance: Is-A relationship) with the studentDetails. Apart from the basic details of student like name, birth year and year enrolled, this class additional information like student ID, major (department), degree (grad or undergrad), residency (resident or international), credits required for the degree, and estimated number of years to graduate. So, by deriving the studentType class from the studentDetails class, it can use the studentDetails class data and functions to set firstName, lastName, birthMonth, and yearEnrolled without having to re-write the code.

studentDetails Class

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

studentDetails
-lastName, firstName: string
-birthMonth: int
-yEnrolled: int
+studentDetails()
+studentDetails(string fName, string lName, int bYear, int yEnroll)
+getFirstName() const: string
+getLastName() const: string
+getBirthYear() const: int
+getYearEnrolled() const: int
+printStudentDetails() const: void

Function Descriptions:

The studentDetails class serves as a base class for managing essential information about a student, including their first name, last name, birth year, and the year they enrolled. This class provides constructors, mutator and accessor methods, and functionality to print student details. **The following are more detailed descriptions of the required functions.**

Function Implementations

1. Constructors

```
studentDetails(); //Default constructor
```

Purpose: Initializes a studentDetails object with default values.
Parameters: None.
Return Value: Constructs an object with empty strings for names and zero for birth year and enrollment year.

2. Parameterized Constructor

```
studentDetails(string fName, string lName, int bYear, int yEnroll);
```

Purpose: Initializes a studentDetails object with specified values.
Parameters:

- **fName:** First name of the student (string).
- **lName:** Last name of the student (string).
- **bYear:** Birth year of the student (integer).
- **yEnroll:** Year the student enrolled (integer).

Return Value: Constructs an object initialized with the provided values.

3. Mutator Methods

```
void setStudentDetails(string fName, string lName, int bYear, int yEnroll);
```

Purpose: Sets multiple student attributes in one call.
Parameters:

- **fName:** First name of the student (string).
- **lName:** Last name of the student (string).
- **bYear:** Birth year of the student (integer).
- **yEnroll:** Year the student enrolled (integer).

Return Value: None.

4. Accessor Methods

```
string getFirstName() const;
```

Purpose: Retrieves the student's first name.
Implementation: Return the value of firstName.

```
string getLastName() const;
```

Purpose: Retrieves the student's last name.
Implementation: Return the value of lastName.

```
int getBirthYear() const;
```

Purpose: Retrieves the student's birth year.
Implementation: Return the value of birthYear.

```
int getYearEnrolled() const;
```

Purpose: Retrieves the year the student enrolled.
Implementation: Return the value of yEnrolled.

5. Utility Methods

```
void printStudentDetails() const;
```

Purpose: Prints the details of the student object, including first name, last name, birth year, and enrollment year.
Parameters: None.
Return Value: None.

studentType Class

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

studentType
-studentID: string
-department: string
-stdDegree: string
-residency: string
-creditsRequired: int
-yearstoGrad: int
+studentType()
+studentType(string fName, string lName, int bYear, int yEnroll, string ID, string dept, string degree, string residence, int credsReqd, int gradYears)
+setStudentType(string fName, string lName, int bYear, int yEnroll, string ID, string dept, string degree, string residence, int credsReqd, int gradYears)
+getStudentID() const: string
+getDepartment() const: string
+getDegree() const: string
+getResidency() const: string
+getCreditsRequired() const: int
+getYearstoGrad() const: int
+toLowerCase(const string &): string
+checkStudentID(string idTemp) const: bool
+printStudentTypeData() const: void

Function Descriptions:

Each function in the *studentType* class serves a specific purpose, from initialization and data setting to retrieval and printing. **The following are more detailed descriptions of the required functions.**

Function Implementations

1. Constructors

```
studentType(); //Default constructor
```

Purpose: Initializes a studentType object with default values.
Parameters: None.
Return Value: Constructs an object with empty strings for studentID, department, stdDegree, residency, and zero for creditsRequired and yearstoGrad

2. Parameterized Constructor

```
studentType(string fName, string lName,
                int bYear, int yEnroll, string ID, string dept,
                string degree, string residence, int credsReqd,
                int gradYears);
```

Purpose: Initializes a studentType object with specified values.
Description: This constructor initializes all member variables with the provided parameters. It also initializes the base class studentDetails with the first name, last name, birth year, and year enrolled.
Parameters:

fName: First name of the student (string).
lName: Last name of the student (string).

bYear: Birth year of the student (integer).
yEnroll: Year the student enrolled (integer).
ID: Unique student ID (string).
dept: Academic department (string).
degree: Type of degree (string).
residence: Residency status (string).
credsReqd: Credits required for graduation (integer).
gradYears: Years remaining until graduation (integer).

Return Value: Constructs an object initialized with the provided values.

3. Mutator Methods

```
void setStudentDetails(string fName, string lName,
                       int bYear, int yEnroll, string ID, string dept,
                       string degree, string residence, int credsReqd,
                     int gradYears);
```

Purpose: Sets multiple student attributes in one call.

Description: This function updates all the details of the student. It first calls the base class's setStudentDetails method to set the basic information, then assigns values to the specific member variables of studentType.

Parameters:

fName: First name of the student (string).
lName: Last name of the student (string).
bYear: Birth year of the student (integer).
yEnroll: Year the student enrolled (integer).
ID: Unique student ID (string).
dept: Academic department (string).
degree: Type of degree (string).
residence: Residency status (string).
credsReqd: Credits required for graduation (integer).
gradYears: Years remaining until graduation (integer).

Return Value: None.

4. Accessor Methods

```
string getStudentID() const;
```

Purpose: Retrieves the student's ID.

Implementation: Returns the value of *studentID*.

```
string getDepartment() const;
```

Purpose: Retrieves the student's department.

Implementation: Returns the value of *department*.

```
string getDegree() const;
```

Purpose: Retrieves the student's degree.

Implementation: Returns the value of *stdDegree*.

```
string getResidency() const;
```

Purpose: Retrieves the student's residency status.

Implementation: Returns the value of *residency*.

```
int getCreditsRequired() const;
```

Purpose: Retrieves the number of credits required to graduate.

Implementation: Returns the value of *creditsRequired*.

```
int getYearstoGrad() const;
```
Purpose: Retrieves the years remaining until graduation..
Implementation: Return the value of *yearstoGrad.*

5. Helper Method

```
static string toLowerCase(const string &str);
```
Purpose: Converts a given string to lowercase.
Description: This function takes a string as input and converts it to lowercase. It iterates through each character, checking if it is uppercase and converting it accordingly.
Parameters:
str: Input string to be converted (string).
Return Value: Returns the lowercase version of the input string.

6. Validation Method

```
bool checkStudentID(string idTemp) const;
```
Purpose: Validates the student ID based on specific criteria.
Description: This function checks if the provided student ID (idTemp) is valid. It constructs the expected ID based on the last name, birth year, and enrollment year, then compares it (both in original and lowercase forms) to the provided ID. The criteria for the ID is"
The First three characters of the last name.
First two digits of the birth year.
Third and fourth digits of the enrollment year.
Parameters:
idTemp: Temporary ID string to check (string).
Return Value: Returns true if the ID is valid; otherwise, false.

7. Utility Method

```
void printStudentTypeData() const;
```
Purpose: Prints the details of the student object, including all attributes.
Description: This function prints all relevant details about the student. It first calls the base class's method to print basic information including first name, last name, birth year, and enrollment year. Then prints the specific details of studentType, including ID, department, degree, residency status, credits required, and years remaining until graduation.
Parameters: None.
Return Value: None.

fileHandler Class

The ***fileHandler*** class manages reading and writing student data and their corresponding tuition fees to and from files. The *fileHandler* UML class diagram is as follows:

fileHandler
+readStudentData(filename: string): vector<studentType>
+writeFeeData (filename: string, students: vector<studentType>): void

Function Descriptions:

The fileHandler class has methods that operate on studentType objects. It reads and writes data related to students using instances of studentType, indicating that fileHandler is composed of studentType instances. **The following are more detailed descriptions of the required functions.**

Function Implementations

1. Read File Method

```
vector<studentType>      readStudentData(const      std::string&
filename);
```

Description: This method reads student data from a specified file and populates a vector of studentType objects.
Parameters: const std::string& filename: The name of the file from which to read the student data.
Return Value: Returns a vector of studentType objects containing the data read from the file.
a studentType object with default values.
Implementation Steps:

Open the File: Use an ifstream object to open the specified file.
Read Header: Read the first line to skip the header.
Read Data.
Extract fields such as first name, last name, birth year, year enrolled, student ID, department, degree, residency, credits required, and years to graduate.
Create studentType Object: Construct a studentType object using the extracted data.
Validate Student ID: Call checkStudentID on the studentType object to validate the ID. If invalid, print an error message. Check the sample output for those details.
Add to Vector: Push the valid studentType object into the students vector.
Close the File: After reading all data, close the file.
Return the Vector: Return the populated vector of studentType objects.

2. Write File Method

```
void    writeFeeData(const    std::string&    filename,    const
vector<studentType>& students);
```

Description: This method writes the calculated fee data for each student into a specified file.
Parameters:

const std::string& filename: The name of the file to which the fee data will be written.
const std::vector<studentType>& students: A vector of studentType objects containing student data.

Return Value: This method does not return a value.
Implementation Steps:

Open the File: Use an ofstream object to open the specified file.
Write Header: Write a header line indicating the contents (e.g., "Student Fee Calculations:").
Set Precision: Set the output precision to two decimal places for fee calculations.
Iterate Over Students: Print student count (Check sample output)
For each studentType object in the vector:
Validate the student ID using checkStudentID.
If valid, create a calculator object to compute the tuition fee.
Write the student's details along with the calculated fee to the file.
If invalid, write a message indicating the invalid ID. Check the sample output for those details.
Close the File: After processing all students, close the file.
Error Handling: If the file cannot be opened, output an error message to the console.

calculator Class

The ***calculator*** class is not a base class; it is a standalone class that uses composition (has-a relationship) to include an instance of the studentType class. The *calculator* UML class diagram is as follows:

calculator
-stdTypeObj: studentType
<<static>> + undergradResident: double = 109.55 + undergradNonResident: double = 317.25 + gradResident: double = 123.25 + gradNonResident: double = 388.75 + nonResidentYearFee: double = 18150.10 + intInsuranceYearFee: double = 4100.00
+calculator ()
+calculator (studentType &)
+calculateTuition(const studentType &, bool printDetails =false): double
+printAllStudentData()const: void

Function Descriptions:

The calculator class is responsible for calculating tuition fees based on the type of student (undergraduate or graduate) and their residency status (resident or non-resident). It can also print student data. **The following are more detailed descriptions of the required functions.**

Function Implementations

1. Constructors

```
calculator(); //Default constructor
```

Purpose: Initializes a studentType object with default values.
Parameters: None.

2. Parameterized Constructor

```
calculator(studentType &stdObj);
```

Purpose: Initializes a calculator object with a specific studentType instance.
Description: This constructor allows for the creation of a calculator object that is associated with a specific student. By passing a reference to a studentType object, the constructor initializes the internal stdTypeObj member variable, enabling the calculator to perform operations based on the details of the provided student.
Parameters:

studentType &stdObj: A reference to a studentType object that contains the details of the student. This object is used to access the student's information (such as degree, residency, and credits required) within the calculator class.

Return Value: This constructor does not return a value. It initializes the calculator object for further operations.

3. Evaluation Method

```
double calculateTuition(const studentType &stdObj, bool printDetails);
```

Description: This method calculates the tuition fee for a given student based on their degree and residency status. It can also print detailed calculations if specified.
Parameters:

const studentType &stdObj: A constant reference to a studentType object containing the student's details (e.g., degree, residency, credits required).

bool printDetails: A boolean flag indicating whether to print detailed calculation steps (default is false).
Return Value: Returns the total calculated tuition fee as a double.

Implementation Steps:

Initialize Fee: Start with a fee variable set to 0.0.
Determine Degree and Residency:
If the degree is "Undergraduate":
If the residency is "Resident", calculate the fee using undergradResident multiplied by the number of credits required.
If the residency is "Non-Resident", calculate the fee using undergradNonResident, and add the nonResidentYearFee and intInsuranceYearFee for the years until graduation.
If the degree is "Graduate":
If the residency is "Resident", calculate the fee using gradResident.
If the residency is "Non-Resident", calculate the fee using gradNonResident, and add the nonResidentYearFee and intInsuranceYearFee for the years until graduation.
Print Details (if specified): If printDetails is true, print each step of the calculation to the console.
Return Fee: At the end of the method, return the total calculated fee.

4. Utility Method

```
void printAllStudentData() const;
```

Description: This method prints the details of the student associated with the calculator, utilizing the printStudentTypeData method from the studentType class.
Parameters: None.
Return Value: This method does not return a value.
Implementation Steps:

Call Print Method: Use the printStudentTypeData method of the stdTypeObj object to display the student's details, including name, ID, degree, residency status, and any other relevant information.

5. Constants Used

undergradResident: Fee per credit for undergraduate residents = **109.55**.
undergradNonResident: Fee per credit for undergraduate non-residents = **317.25**.
gradResident: Fee per credit for graduate residents = **123.25**.
gradNonResident: Fee per credit for graduate non-residents = **388.75**.
nonResidentYearFee: Annual fee for non-resident students = **18150.10**.
intInsuranceYearFee: Annual insurance fee for international students = **4100.00**.

Development and Testing

In order to simplify development and testing, the project is split into four parts: *studentDetails, studentType, calculator,* and *fileHandler* classes.

- The ***studentDetails (Base class)*** can be developed first and this class can be tested independently of the other class. This class sets firstName, lastName, birthMonth, and yEnrolled.
- The ***studentType (Derived from studentDetails)*** must be developed next (before ***calculator***). This class can be tested after implementing studentDetails.
- The ***calculator (Inheritance- studentType object)*** must be developed next. The calculator class provides a structured way to calculate student tuition based on various parameters. Each function is clearly defined, and the logic for calculations is contained within the calculateTuition method, while student data can be printed with printAllStudentData. Make sure to integrate this class with the studentType class to ensure data consistency and access to student-specific information.
- The ***fileHandler (inheritance)*** must be developed last as this has two methods such as read and write. Both methods need the methods of studentType.
writeFeeData(parameters) also needs the object of **calculator** to calculate the fee of each student.

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

- makefile.
- main.cpp.
- studentDetails.h
- studentType.h
- calculator.h
- fileHandler.h
- CS202_Assignment3_F24.pdf
- CS202_Ast3_Sample_Output_F24.pdf

Submission:
Submit the following files for this assignment.

- studentDetailsImp.cpp
- studentTypeImp.cpp
- calculatorImp.cpp
- fileHandlerImp.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

Guidelines for Submitting Your Work:

- 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*.

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%.

Note: Check the sample output for more details.

What is printed on the console window?

The output on console window prints all the Test cases- 1, 2, 3, and 4 that are declared in main.cpp. Test case 3 is the students' data from input file (student_data.txt contains 10 students information) irrespective of student id valid or invalid.

What is printed into the Estimated_Fees.txt?

```
fh.writeFeeData("Estimated_Fees.txt", students);//from main.cpp
```

The output file is named Estimated_Fees.txt. This file provides only the students' information that is provided in the student_data.txt file. However, it does not include the anticipated cost information for students with invalid IDs.

How fee is calculated for International Graduate Student?

Michael Johnson 2002 2023 joh2023 Mechanical Engineering Graduate International 34 3 //Example
Estimated Fee in Total=
Number of creditsRequired *gradNonResident + yearstoGrad (nonResidentYearFee + intInsuranceYearFee)
Estimated Fee in Total= 34 * 388.75 + 3(18150.10) + 3(4100.00) = 79967.80

How fee is calculated for Residential Graduate Student?

John Doe 1999 2024 doe1924 Computer Science Graduate Resident 30 2 //Example
Estimated Fee in Total= Number of creditsRequired *gradResident = 30 * 123.25 = 3697.50

Example: How the composition Works?

From ***studentType*** class

```
// Test case 1: Create studentType object that can set the following parameters
studentType student("Sophia", "Taylor", 1998, 2020, "tay2920",
  "Computer Science", "Undergraduate", "International", 120, 4);
```

//The above student object invokes studentType parameterized constructor. However, the implementation is to set the firstName, lastName, birthMonth, and yEnrolled, it invokes studentDetails parameterized constructor and the remaining parameters are initialized according to the passed parameters.

```
student.printStudentTypeData();
```

//This function call is from studentType. In this implementation, it calls the base class print method to print the firstName, lastName, birthMonth, and yEnrolled. and the remaining parameters are printed from the studentType member variables or getter functions of studentType.

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

Example Execution:

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.

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

Your original text for output should be in black color only.

```
kishore@kishore-ubuntu:/Ast3_F24$ make
g++ -Wall -Wextra -pedantic -std=c++11 -g -c main.cpp
g++ -Wall -Wextra -pedantic -std=c++11 -g -o main main.o studentDetailsImp.o
studentTypeImp.o calculatorImp.o fileHandlerImp.o

kishore@kishore-ubuntu:/Ast3_F24$ ./main

Invalid ID for the student: Jane Smith
Invalid ID for the student: Emily Brown
Invalid ID for the student: David Lee
Invalid ID for the student: Jack Cox
Invalid ID for the student: Alexander Jones
Invalid ID for the student: Emily Webb


Student Fee Calculations:
Student: 1
Name: Sophia Taylor, Birth Year: 1998, Year Enrolled: 2020
Student ID: tay2920, Department: Computer Science, Degree: Undergraduate
Residency: International
Credits Required: 120, Years to Graduate: 4
Undergraduate Non-Resident credit Fee: 317.25*120 = 38070.00
Undergraduate Non-Resident Year Fee: 18150.10*4 = 72600.40
Undergraduate Non-Resident Year Insurance Fee: 4100.00*4 = 16400.00
Estimated Fee in Total: 127070.40

//Similarly other student details on console window

kishore@kishore-ubuntu:/Ast3_F24$ cat Estimated_Fees.txt


Student Fee Calculations:
Student: 1
Name: John Doe
Birth Year: 1999, Year Enrolled: 2024
Student ID: doe1924, Department: Computer Science
Degree: Graduate, Residency: Resident
Credits Required: 30, Years to Graduate: 2
Estimated Total: 3697.50

Student: 2

Invalid ID for the student (To file): Jane Smith

//Similarly other student details on Estimated_Fees.txt
```

fileHandler::readStudentData(const std::string& filename)

main.cpp

```
studentType::printStudentTypeData()
{
studentDetails::printStudentDetails();
//Other details
}
```

calculator::calculateTuition(parameters)

fileHandler::writeFeeData(parameters)

Note: Check the file "CS202_Ast3_Sample_Output_F24.pdf" for more detailed outputs on console window and students estimated fee calculations stored in output file.