NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES ISLAMABAD

Object Oriented Programming – Spring 2024

ASSIGNMENT- 3

Due Date: Sunday 31st March, 2024 at 11:59 pm on Google Classroom

Please follow the following submission instructions. Failure to submit according to the format would result in a deduction of marks. Submissions other than Google classroom (e.g., email etc.) will not be accepted. No late submission will be accepted. Correct and timely submission of the assignment is the responsibility of every student; hence no relaxation will be given to anyone.

Plagiarism: This is an individual assignment and any kind of plagiarism will result in a zero.

Instructions: Total Marks: 170

- Make sure that you read and understand each and every instruction. If you have any questions or comments, you are encouraged to discuss your problems with your colleagues and instructors on google classroom.
- The student is solely responsible for checking the final .cpp files for issues like corrupt files, viruses in the file, or mistakenly exe sent. If we cannot download the file from Google Classroom, it will lead to zero marks in the assignment.
- The displayed output should be well-mannered and well presented. Use appropriate comments and indentation in your source code.
- If there is a syntax error in the code, zero marks will be awarded in that part of the assignment.

Keep a backup of your work always that will be helpful in preventing any mishap and avoid last hour submissions. Submit a single '.zip' file for your assignment and each problem solution must be provided in a separate CPP file. For instance, you must name the file containing solution of first file as 'q1.cpp'

Submission Guidelines

- Dear students, we will be using auto-grading tools, so failure to submit according to the below format would result in zero marks in the relevant evaluation instrument.
- For each question in your assignment, make a separate .cpp file e.g., for question 1, make q1.cpp and so on. Each file that you submit must contain your name, student id, and assignment # on top of the file in the comments.
- Combine all your work in one folder. The folder must contain only .cpp files (no binaries, no exe files etc.).
- Rename the folder as YOUR_NAME_I23-XXXX_Asst_03.zip and compress the folder as a zip file. Strictly follow this naming convention, otherwise marks will be deducted.
- Submit the .zip file on Google Classroom within the deadline.
- The student is solely responsible to check the final zip files for issues like corrupt file, virus in the file, mistakenly exe sent. If we cannot download the file from Google classroom due to any reason it will lead to zero marks in the assignment.

Note: Start early so that you can finish it on time.

Question 01 [10 Marks]:

Soft Swirl offers a variety of delicious ice cream delights. Introducing the IceCream class, designed to manage and customize orders seamlessly.

Following are private data members:

- **char *flavor** – a string to store the ice cream flavor.
- **char *topping** – a string to store the ice cream topping(s).
- **char *servingType** – a string to store the serving type (e.g., cup, cone).
- **bool isReady** – a boolean to store the status of the ice cream, indicating if it's ready or not.
- **double price** – a double to store the price of the ice cream.

The class has following public member functions:

- **IceCream()** – a default constructor.
- **IceCream(char* flavor, char* topping, char* servingType, double price)** – a parameterized constructor
- **IceCream(char* topping, double price)** – a parameterized constructor
- **IceCream(const IceCream &iceCream)** – a copy constructor.
- **~IceCream()** – a destructor.
- **void setFlavor(char* flavor)** – a setter for flavor
- **void setTopping(char* topping)** – a setter for topping.
- **void setServingType(char* servingType)** – a setter for serving type.
- **void setPrice(float price)** – a setter for price.
- **char* getFlavor() const** – a getter for flavor.
- **char* getTopping() const** – a getter for topping.
- **char* getServingType() const** – a getter for serving type.
- **double getPrice() const** – a getter for price.
- **void makeIceCream()** – a function to make ice cream (check if topping is not NULL then set the value of isReady to true).
- **bool checkStatus()** – a function to check if the ice cream is ready or not.

Write a main function that takes appropriate input from the user and shows functionality of the class.

Question 02 [15 Marks]:

Design a class **Student** with the following private data members:

- **int stdID** – an integer to hold the student's identification number
- **string name** – a string to hold the student's first name
- **string* courseCodes** – a dynamic array of strings to hold the course codes of the courses taken by the student
- **int numOfCourses** – an integer to hold the number of courses taken by the student
- **int* courseGrades** – a dynamic array of integers to hold the grades obtained by the student in the courses taken
- **float gpa** – a float to hold the grade point average of the student, calculated as the sum of all course grades divided by the number of courses taken

The class should implement the following public methods:

- **int getStdID()** – a getter for the student's ID
- **string getName()** – a getter for the student's first name
- **int getNumOfCourses()** – a getter for the number of courses taken by the student
- **string getCourseCode(int i)** – a getter for the course code of the course taken at index i of the courseCodes array
- **int getCourseGrade(int i)** – a getter for the grade obtained by the student in the course taken at index i of the courseGrades array
- **float getGPA()** – a getter for the student's GPA
- **void setStdID(int id)** – a setter for the student's ID
- **void setName(string firstName)** – a setter for the student's first name
- **void setCourseGrade(string courseCode, int grade)** – a function to set the grade obtained by the student in the course with the given course code
- **void addCourse(string courseCode, int grade)** – a function to add a course with the given course code and grade to the student's record
- **void calcGPA()** – a function to calculate the student's GPA

Implement the following functions in global scope. All functions will take an array of Student objects as an argument. e.g. Student students[10] - an array of 10 Student objects

- **Student getStudentAt(Student students[], int index)** – returns the Student object at the given array index
- **float calcClassGPA(Student students[], int numStudents)** – calculates the average GPA of all students in the array
- **float getMaxGPA(Student students[], int numStudents)** – calculates the maximum GPA from all the students in the array

- **int getMinGPA(Student students[], int numStudents)** – calculates the minimumGPA from all the students in the array
- **void printStudentRecord(Student student)** – prints the record of the given student, including the student's ID, name, course codes, grades, and GPA
- **void printAllStudentRecords(Student students[], int numStudents)** – prints the records of all students in the array

Program should have a user interface to show all the functionality of the class.

Question 03 [10 Marks]:

You are developing a hotel management system for a booking agency. The system enables users to list available hotel rooms, view room details, and update booking information as needed.

HotelBooking Class have the following private data members:

- **string bookingID** – unique identification number of the rental booking.
- **string customerName** – a string to hold the name of the customer making the booking.
- **string hotelName** – a string to hold the name of Hotel.
- **string roomType** – a string to specify the type of room being booked (e.g., single, double, suite).
- **string checkIn** – a string to indicate the start date of the hotel stay.
- **string checkOut** – a string to indicate the end date of the hotel stay
- **int stayDuration** – an integer to calculate the duration of the hotel stay in days.
- **double roomRate** – a double to specify the room rate per day.

The class should have following Public Methods:

- **HotelBooking(string bookingID, string customerName, string hotelName, string roomType, string checkIn, string checkOut)** – a parametrized constructor.
- **validateBookingID() const** – a const member function to check the validity of the booking ID format. You can validate by checking if the reservation ID has the correct format by verifying if the ID has 8 characters, followed by 2 special characters, and finally ending with 4 numbers. The sum of the four numbers must be less than 18
- **calculateRoomRate() const** – a const member function to calculate and return the room rate based on room type.
- **calculateTotalCost() const** – a const member function to calculate and return the total cost of the stay.
- **getBookingDetails() const** – a const member function to return details of the hotel booking including booking ID, customer name, hotel, room type, check-in and check-out dates, stay duration, room rate, and total cost.

- **updateBookingInfo()** – a member function to update booking information such as check-in and check-out dates, and recalculate stay duration, rate, and total cost. They can choose what they want to update through this function. The function should also recalculate the stay duration, room rate, and total cost based on the updated information. Additionally, the function should validate the input data and make sure it is within acceptable ranges.

Program should have a user interface to show all the functionality of the class.

Note: you are allowed to set the rates yourself.

Question 04 [20 Marks]:

You've been assigned to create an advanced inventory management system for a retail store using C++ classes. The system should efficiently handle product data, transactions, and provide useful insights. You have to create 2 classes.

A product class with Attributes:

- **int id** – a unique identifier
- **char* name** – product name
- **float price** – product price
- **int quantity** – the quantity of product available

The class should have following Public Methods:

- Constructor to initialize attributes
- Getters and setters for attributes
- A destructor

An inventory class with the following attributes:

- **Product* products** – an array of products in the inventory created dynamically.
- **int count**– number of products currently in the inventory
- **int size** – maximum items the inventory can store

The class should have following Public Methods:

- **Inventory(int size)** - a constructor to initialize the inventory with a max size
- **void add_product(int id, char* name, float price, int quantity)** – a function to add a new product to the inventory with the specified ID, name, price, and quantity.

  void add_product(Product* product) – a function to add a new product to the inventory by sending a whole product.
- **void remove_product(int id)** – a function to remove a product from the inventory based on its unique identifier.

- **void remove_product(char* name)** – a function to remove one or more products from the inventory based on the name.
- **void remove_product(int id, int count)** – a function to remove a specified quantity of products based on their ID from the inventory.
- **void update_quantity(int id, int quantity_change)** – a function to update the quantity of a product in the inventory based on its ID. Quantity_change can be positive or negative to add or subtract from the existing quantity.
- **Product& get_product(int id)** – a function to retrieve product information (ID, name, price, quantity) based on the specified ID.
- **float calculate_inventory_value()** – a function to calculate the total value of the inventory by summing up the prices of all products.
- **void display()** – a function to display the entire product catalogue, including ID, name, price, and quantity.
- **~Inventory()** – a destructor

Program should have a user interface to show all the functionality of the class.

Question 05 [15 Marks]:

Implement a class Library with the following private data members:

- **char *bookTitle** – a string to hold the title of the book
- **char *author** – a string to hold the name of the author
- **int bookID** – an integer to hold book identification numbers between 1 and 100.
- **int quantity** – an integer to hold the number of copies of the book in the library
- **float price** – a float to hold the price of each book
- **static int totalBooks** – a static integer to hold the total number of books in the library

The class should implement the following public methods:

- **char* getBookTitle()** – a getter for book title
- **char* getAuthor()** – a getter for author name
- **int getBookID()** – a getter for book ID
- **int getQuantity()** – a getter for number of copies of the book
- **float getPrice()** – a getter for the price of each book
- **void setBookTitle(char* title)** – a setter for book title
- **void setAuthor(char* authorName)** – a setter for author name
- **void setBookID(int bookID)** – a setter for book ID
- **void setQuantity(int quantity)** – a setter for number of copies of the book
- **void setPrice(float price)** – a setter for the price of each book
- **static void setTotalBooks(int totalBooks)** – a setter for total number of books
- **void calcTotalPrice()** – a function to calculate the total price of all copies of the book

- **static int getTotalBooks()** – a static function to get the total number of books in the library

Implement the following functions in global scope. All functions will take an array of Library objects as an argument. e.g. Library books[10] – an array of 10 Library objects
Functions:

- **Library getBook_at(Library books[100], int index)** – returns the Library object at the given array index
- **void addBook(Library books[100], Library newBook)** – adds a new book to the library array
- **void removeBook(Library books[100], int bookID)** – removes the book with the given book ID from the library array
- **void SortByTitle(Library books[100])** – sorts the books in ascending order based on title
- **void SortByAuthor(Library books[100]**]) – sorts the books in ascending order based on author name
- **void SortByPrice(Library books[100])** – sorts the books in ascending order based on price
- **bool searchByTittle(Library books[10], char* titlename**) - returns true if the book with the titlename is found in the list
- **Library mostExpensiveBook(Library books[10]) -** returns the book with the most expensive price from the list

Note: Ensure all necessary input validation, for example, quantity cannot be a negative number. Also assume, no two books will have the same book ID.

Question 06 [30 Marks]:

Implement your own string class using only primitive data types. Your string class should contain some of the same functionality as the string class in C++.

Your class should have the following member variables:

- **char *data** – holds the character array that constitutes the string
- **int length** – the current length of the string that the object contains (number of characters in data)

Your object should have the following constructors and destructor:

- **String()** – default constructor
- **String(int size)** – alternate constructor that initializes length to size and initializes data to a new char array of size.

- **String(char* str)** – alternate constructor that initializes length to size of str, initializes data to a new char array of size length and fills the contents of data with the contents of str. Note: You can assume that any character array that is sent as str will be terminated by a null character ('\0')
- **String(const String &str)** – copy constructor
- **~String()** – destructor in which you should delete your data pointer.

Your class should have the following functions:

- **int strLength()** – returns the length of the string in the object
- **void clear()** – should clear the data in your string class (data and length)
- **bool empty()** – this method should check to see if data within the String class is empty or not.
- **int charAt(char c)** – this method returns the first index at which this character occurs in the string
- **char* getdata()** – a getter to get the actual string
- **bool equals(char*str)** – this method compares if the data in the calling string object is equal to str.
- **bool equalsIgnoreCase(char* str)** – this method compares the calling string object data with the data in str without considering the case.
- **char* substring(char* substr, int startIndex)** – this method should search for substr in data within the calling String object starting from the startIndex. The method should search for the substring from the startIndex and return the substring from the position found till the end. For example, if you had the string "awesome" and you wanted to find the substring 'es' starting from the beginning of the string (startIndex = 0). Your function should return "esome". Returns NULL if the substring is not found.
- **char* substring(char* substr, int startIndex, int endIndex)** – this method should search for substr in data within the calling String object between the startIndex and endIndex. For example, if you had the string "awesome" and you wanted to find the substring 'es' starting from startIndex=2 and endIndex=5. Your function should return "esom". Returns NULL if substring is not found.
- **void print()** – a function that will output the contents of the character array within the object. If the contents of the String are empty (length == 0) then you should print "NULL".
- **bool startsWith(char* substr) const** – this method checks whether the current string starts with the specified substring. It returns true if the string begins with the given substring; otherwise, it returns false.
- **bool endsWith(char* substr) const** – this method checks whether the current string ends with the specified substring. It returns true if the string ends with the given substring; otherwise, it returns false.
- **void concatenate(char* str)** – this method concatenates the specified string with the current string. It appends the characters of the provided string to the end of the current string, effectively extending its length
- .**void insert(int index, char* str)** – this method inserts the specified string at the specified

index within the current string.

- **void remove(char* substr)** – this method removes all occurrences of the specified substring from the current string. If the substring is found, it is removed; otherwise, the string remains unchanged.
- **void trim()** – Removes any leading and trailing whitespace characters from the current string.
- **void toUpperCase()** – Converts all characters in the current string to uppercase.
- **void toLowerCase()** – Converts all characters in the current string to lowercase.

Additional Specifications:

- Your program should have NO memory leaks
- You should not use the built-in C++ string class anywhere in your object.
- You must appropriately use the const keyword (you need to decide which functions will be constant qualified and which will not).
- You may define additional helper functions as needed.

Question 07 [70 marks]:

Minesweeper is a single-player logic puzzle game where the objective is to uncover all safe squares without detonating a mine. The game is played on an N x N square grid where some squares contain mines, while others contain clues about the number of neighboring mines. Clicking a safe square reveals its number and clicking a mine results in a game over.

You are given the task to recreate the minesweeper game by making appropriate use of classes. Create a Minesweeper class and implement the following attributes and functions.

Attributes:

- **int** Grid** – a 2D array representing the game board with hidden and revealed states for each cell.
- **int size** – size of the grid (7x7 minimum)
- **int Mines:** A certain number of randomly placed mines on the grid.

Functions:

- **Minesweeper(int size, int mines)** – a default constructor
- **~Minesweeper()** – a destructor
- setters and getters for all data members.
- **void reset()** – a function to reset the game, by keeping the board size and number of mines the same, but changing the location of the mines.
- **void revealAll()** – a function to reveal all cells without the mines. In other words, it

completes the game.

- **bool revealCell(int row, int col)** – a function to uncover a cell, displaying its value (number or mine). If a number, the cell is revealed and the game continues (returns true). If a mine, all cells are revealed and the game ends (game over, returns false).
- **void flagCell(int row, col)** – a function to allow players to mark a cell as potentially containing a mine.
- **bool checkWin()** – a function to determine if all non-mine cells have been revealed, indicating victory.
- **void displayGrid()** – a function to display the grid

<u>Note: You may define additional attributes and functions as needed.</u>

Additional Specifications:

- The user interacts with the game by providing coordinates for a cell to reveal or flag.
- The game provides feedback through the revealed grid and win/lose conditions.
- Develop a user interface to interact with the game.
- Your program should not stop until game is completed (at win or loss) or an exit functionality is defined.
- Your program should have NO memory leaks
- You are required to add all the necessary error handling checks for wrong inputs. The program should prompt the user to re-input rather than terminate.

Example Usage:

- User starts a game with a 10x10 grid and 10 mines.
- User selects cell (5, 3).
- User selects to either flag the cell or reveal it.
- If "reveal" is selected and the cell is safe with 2 adjacent mines, the revealed grid shows "2" at (5, 3).
- If "flag" is selected, the cell (5, 3), must be displayed as flagged (eg. use "F" as flagged)
- The board / game grid should display after the user makes their decision to flag or reveal.
This will happen at the end of every move.
- A cell that is revealed cannot be changed again.
- A cell that is flagged can be unflagged.
- User continues revealing cells and placing flags strategizing to avoid mines.
- User wins if all safe cells are revealed without hitting a mine.
- User loses if a mine is clicked, revealing all cells and ending the game.

GOOD LUCK!