C++ Midterm Exam Guidance for Group B

The midterm exam will require you to choose three out of five real-world examples to implement in C++. You will have 10 minutes for each example to present your solution and explain your code clearly. This document provides detailed guidance on which functions and concepts to utilize for each example.

Choose examples you are most comfortable with, and make sure to keep your implementations simple. Focus on using clear variable names, comments, and proper formatting to demonstrate your understanding of C++ concepts.

Luise	Brodey
Matope	Salum
Matthews	Jessica
Norcross	Landon
Perez	Lizbeth
Perez Ambrosio	Jose
Sosa	Jayden

Example 1: Employee Record Management System using Class

Create a program that allows a manager to input the names and salaries of five employees using an `Employee` class, store them, and then display the employee names along with their average salary.

Guidance:
- Create an `Employee` class with private members for name and salary, and public member functions for setting and getting these values.
- Use an array of `Employee` objects to store the employee data.
- Create a function called `input_employee_data()` to take input for names and salaries and store them in the `Employee` objects.
- Create a function called `display_employee_data()` to show all employee names and their salaries.
- Create a function called `calculate_average_salary()` to compute and display the average salary.

Example 2: Sorting Scores of Participants with File I/O

Create a program that allows a user to input the scores of ten participants, store them in an array, and then sort the scores in ascending order using selection sort. After sorting, display the sorted scores. Additionally, save the sorted scores to a file and provide an option to load and display scores from a file.

Guidance:
- Use an array to store the participant scores.
- Create a function called `input_scores()` to take input for participant scores.
- Create a function called `selection_sort()` to sort the scores in ascending order.
- Create a function called `display_sorted_scores()` to display the sorted scores.
- Implement file I/O functions: `save_to_file()` to write the sorted scores to a file and `load_from_file()` to read and display scores from a file.
- Use the selection sort algorithm to implement the `selection_sort()` function.

Example 3: Basic Inventory Management System

Create a program that manages an inventory of items, allowing the user to add, remove, or view items. Each item should have a name, quantity, and price.

Guidance:
- Use an array of structures or classes to store item details.
- Create functions to handle adding new items, removing items, and displaying the

inventory.
- Implement file I/O functions to save and load the inventory data from a file.
- Use formatted output to display the inventory in a table format.

Example 4: Simple Quiz Game

Create a quiz game that asks the user five multiple-choice questions. After the user answers all questions, display their score.

Guidance:
- Use arrays to store the questions, options, and correct answers.
- Create a function called `ask_questions()` to display each question and record the user's answer.
- Create a function called `calculate_score()` to compute and display the user's final score.
- Use a loop to ask each question in the array.

Example 5: Advanced Seating Arrangement Pattern

Create a program that generates and displays a seating arrangement for a theater using a 2D array. The program should allow the user to specify the number of rows and seats per row. Additionally, it should mark some seats as reserved.

Guidance:
- Use a 2D array to represent the seating arrangement.
- Create a function called `generate_seating(int rows, int seats_per_row)` to create the seating pattern.
- Create a function called `reserve_seat()` to mark specific seats as reserved.
- Use nested `for` loops to display the seating pattern, indicating which seats are reserved.
- Allow the user to specify the number of rows and seats per row.