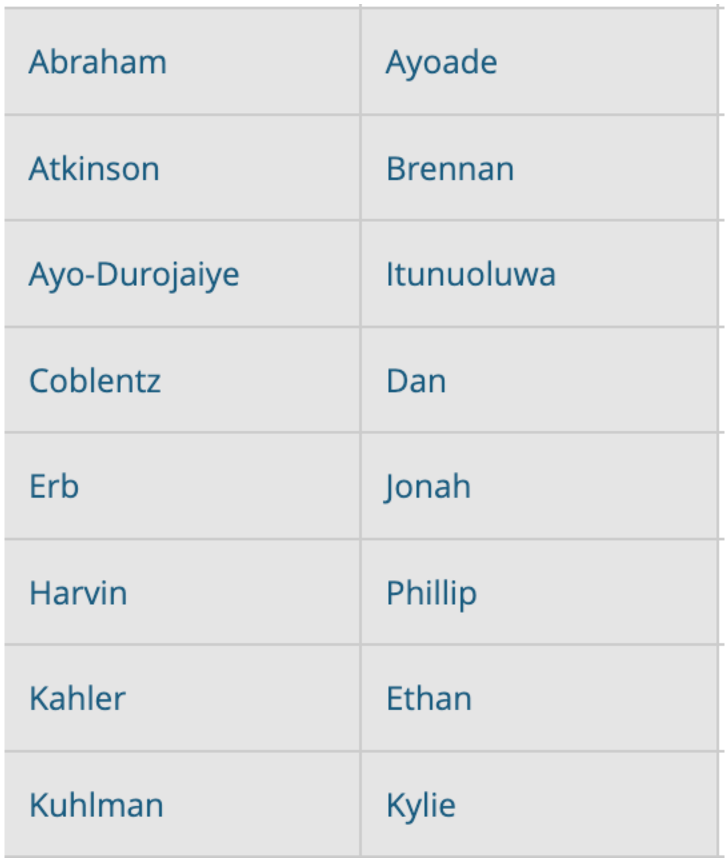
C++ Midterm Exam Guidance (Group A)

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.



Abraham	Ayoade
Atkinson	Brennan
Ayo-Durojaiye	Itunuoluwa
Coblentz	Dan
Erb	Jonah
Harvin	Phillip
Kahler	Ethan
Kuhlman	Kylie

Example 1: Student Information Management System using Class

Create a program that allows a teacher to input the names and grades of five students using a `Student` class, store them, and then display the student names along with their average grade.

Guidance:
- Create a `Student` class with private members for name and grade, and public member functions for setting and getting these values.
- Use an array of `Student` objects to store the student data.
- Create a function called `input_student_data()` to take input for names and grades and store them in the `Student` objects.
- Create a function called `display_student_data()` to show all student names and their grades.
- Create a function called `calculate_average()` to compute and display the average grade for the class.

Example 2: Sorting Product Prices with File I/O

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

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

Example 3: Create a Simple Calculator

Create a program that functions as a simple calculator, allowing the user to add, subtract, multiply, or divide two numbers.

Guidance:
- Use a `switch` statement or `if-else` structure to handle the four operations.
- Create separate functions for each operation: `add()`, `subtract()`, `multiply()`, and

`divide()`.
- Use a loop to allow the user to perform multiple calculations until they choose to exit.

Example 4: Number Guessing Game

Create an advanced number guessing game where the user has to guess a random number between 1 and 100. The game should include the following additional features:
1. The program keeps track of the number of guesses made.
2. The user can choose to play multiple rounds, with the program displaying the number of rounds won by the user.
3. The program provides a summary of the user's performance (e.g., total guesses, average guesses per round).

Guidance:
- Use the `rand()` function to generate a random number.
- Use a `while` loop to keep asking for the user's guess until they get it right.
- Create a function `play_round()` to handle a single round of guessing.
- Create a function `display_summary()` to show the user's performance at the end of the game.
- Use a counter to keep track of the number of guesses and rounds.

Example 5: Classroom Seating Arrangement with 2D Array

Create a program that helps visualize seating arrangements in a classroom using a 2D array. The user should be able to enter the number of rows and columns in the classroom, and the program should generate a seating arrangement pattern with a unique seat number for each position.

Guidance:
- Use a 2D array to store the seating arrangement.
- Create a function called `generate_seating_pattern(int rows, int columns)` to fill the 2D array with seat numbers (e.g., 1, 2, 3, etc.).
- Use nested `for` loops to traverse the 2D array and display the seat numbers in a grid format.
- Allow the user to specify the number of rows and columns.
- Provide an option for the user to change seat numbers or rearrange seats.