Go Language Exercises

(Intermediate to Advanced Concepts**)**

Exercise 1: Employee Management System

Topics Covered: Go Conditions, Go Loops, Go Constants, Go Functions, Go Arrays, Go Strings, Go Errors

Case Study:

A company wants to manage its employees' data in memory. Each employee has an ID, name, age, and department. You need to build a small application that performs the following:

1. **Add Employee**: Accept input for employee details and store them in an array of structs. Validate the input:
   - ID must be unique.
   - Age should be greater than 18. If validation fails, return custom error messages.
2. **Search Employee**: Search for an employee by ID or name using conditions. Return the details if found, or return an error if not found.
3. **List Employees by Department**: Use loops to filter and display all employees in a given department.
4. **Count Employees**: Use constants to define a department (e.g., "HR", "IT"), and display the count of employees in that department.

Bonus:

Refactor the repetitive code using functions, and add error handling for invalid operations like searching for a non-existent employee.

Exercise 2: Bank Transaction System

Topics Covered: Go Constants, Go Loops, Go Break and Continue, Go Functions, Go Strings, Go Errors

Case Study:

You need to simulate a bank transaction system with the following features:

1. **Account Management**: Each account has an ID, name, and balance. Store the accounts in a slice.
2. **Deposit Function**: A function to deposit money into an account. Validate if the deposit amount is greater than zero.
3. **Withdraw Function**: A function to withdraw money from an account. Ensure the account has a sufficient balance before proceeding. Return appropriate errors for invalid amounts or insufficient balance.
4. **Transaction History**: Maintain a transaction history for each account as a string slice. Use a loop to display the transaction history when requested.
5. **Menu System**: Implement a menu-driven program where users can choose actions like deposit, withdraw, view balance, or exit. Use constants for menu options and break the loop to exit.

Exercise 3: Inventory Management System

Topics Covered: Go Conditions, Go Type Casting, Go Functions, Go Arrays, Go Strings, Go Errors

Case Study:

A store needs to manage its inventory of products. Build an application that includes the following:

1. **Product Struct**: Create a struct to represent a product with fields for ID, name, price (float64), and stock (int).
2. **Add Product**: Write a function to add new products to the inventory. Use type casting to ensure price inputs are converted to float64.
3. **Update Stock**: Implement a function to update the stock of a product. Use conditions to validate the input (e.g., stock cannot be negative).
4. **Search Product**: Allow users to search for products by name or ID. If a product is not found, return a custom error message.
5. **Display Inventory**: Use loops to display all available products in a formatted table.

Bonus:

- Add sorting functionality to display products by price or stock in ascending order.

Exercise 4: Online Examination System

Topics Covered: Go Loops, Go Break and Continue, Go Constants, Go Strings, Go Functions, Go Errors

Case Study:

Develop an online examination system where users can take a quiz.

1. **Question Bank**: Define a slice of structs to store questions. Each question should have a question string, options (array), and the correct answer.
2. **Take Quiz**: Use loops to iterate over questions and display them one by one. Allow the user to select an answer by entering the option number.
   - Use continue to skip invalid inputs and prompt the user again.
   - Use break to exit the quiz early if the user enters a specific command (e.g., "exit").
3. **Score Calculation**: After the quiz, calculate the user's score and display it. Use conditions to classify performance (e.g., "Excellent", "Good", "Needs Improvement").
4. **Error Handling**: Handle errors like invalid input during the quiz (e.g., entering a non-integer value for an option).

Bonus:

- Add a timer for the quiz, limiting each question to a fixed amount of time.

Exercise 5: Climate Data Analysis

Topics Covered: Go Arrays, Go Strings, Go Type Casting, Go Functions, Go Conditions, Go Loops

Case Study:

You are tasked with analyzing climate data from multiple cities. The data includes the city name, average temperature (°C), and rainfall (mm).

1. **Data Input**: Create a slice of structs to store data for each city. Input data can be hardcoded or taken from the user.
2. **Highest and Lowest Temperature**: Write functions to find the city with the highest and lowest average temperatures. Use conditions for comparison.

3. **Average Rainfall**: Calculate the average rainfall across all cities using loops. Use type casting if necessary.
4. **Filter Cities by Rainfall**: Use loops to display cities with rainfall above a certain threshold. Prompt the user to enter the threshold value.
5. **Search by City Name**: Allow users to search for a city by name and display its data.

Bonus:

- Add error handling for invalid city names and invalid input for thresholds.