November 15, 2024

Module Two Final Test

Chill and answer the questions, And **DO NOT** go to **CHAT GPT** unless you reallllllly have to!

1. Write a function `ResizeSlice` that takes a slice of integers and a target length as arguments. If the slice is smaller than the target length, append zeros until it reaches the target length. If it's larger, truncate the slice. Ensure that the function handles the edge cases correctly (e.g., when the target length is zero or negative).

2. Implement a function `WordFrequency` that takes a slice of strings representing words and returns a map with each unique word as a key and its count as the value. Ensure the function is case-insensitive and ignores punctuation.

3. Write a function `FindEmployeeAge` that takes a map of employee names (string) to ages (int) and an employee name as input. Return the age of the employee if found, or an error if the employee does not exist in the map.

4. Write a function `RemoveDuplicates` that takes a slice of integers and returns a new slice without any duplicates. Ensure that the order of the original elements is maintained.

5. Write a function `PartitionEvenOdd` that takes an array of integers and returns two slices: one containing all even numbers and the other containing all odd numbers.

6. Implement a function `MergeAndSort` that takes two sorted slices of integers as input and returns a single sorted slice with all elements from both slices. The function should return an error if any of the slices contain duplicate elements.

7. Create a `MultiKeyMap` type that can store values based on multiple keys (a combination of two or more strings). Implement functions to `Set`, `Get`, and `Delete` values from this map. For example, `Set("first", "last", 25)` should store a value `25` that can be retrieved with `Get("first", "last")`.

8. Write a function `WindowSum` that takes a slice of integers and a window size `k` and returns a new slice where each element is the sum of a sliding window of size `k` from the input slice. If `k` is greater than the length of the slice or less than 1, return an error.

9. Write a function `ReplaceMapKeys` that takes a map of string keys to integer values and replaces every key by reversing its string. The function should perform the replacement in-place (modifying the original map without creating a new one).

10. Define a custom error type `SliceError` that includes the operation performed (e.g., "append", "remove"), the index involved (if applicable), and a descriptive message. Then, implement a function `SafeRemove` that takes a slice of integers and an index, removes the element at that index, and returns the new slice or an appropriate `SliceError` if the index is out of bounds.

11. Write a function `SumVariadic` that takes a variadic parameter of integers and returns their sum. Demonstrate how to call this function with both a slice of integers and individual integer arguments.

12. Create a function `IsPalindrome` that takes a string as input and checks whether it is a palindrome. Make the function case-insensitive and ignore spaces or punctuation.

13. Write a function `Swap` that takes two integers as pointers and swaps their values. Explain the concept of pointers in Go and why they are used in this context.

14. Implement a higher-order function `ApplyToEach` that takes a slice of integers and a function as arguments. The function should apply the provided function to each element of the slice and return a new slice with the results.

15. Write a function `GCD` (Greatest Common Divisor) using recursion in Go. Explain the base case and the recursive case in your solution.

16. Implement a simple memoization function in Go called `MemoizeFib` for calculating Fibonacci numbers. It should use a map to store already computed Fibonacci values and reduce redundant calculations.

17. Create a function `FilterSlice` that takes a slice of integers and a predicate function (a function that returns a boolean). The function should return a new slice containing only the elements that satisfy the predicate function.

18. Write a function `ReverseString` that takes a string as input and returns its reverse. Use rune slices to handle Unicode characters properly.

19. Define a function `ExecuteWithRecovery` that takes a function as an argument, executes it, and recovers from any panics. Demonstrate how to use this function with an example that intentionally causes a panic.

20. Write a function `Retry` that takes another function and a number of retries as arguments. If the provided function returns an error, the `Retry` function should attempt to call it again up to the specified number of retries before returning the error.