Slip 1

Q1. A) Write a program in GO language to accept user choice and print answers using arithmetic operators.

Solutions:-

```
package main

import "fmt"

func main() {
	var num1, num2 float64
	var choice string
	fmt.Print("Enter the first number: ")
	fmt.Scanln(&num1)
	fmt.Print("Enter the second number: ")
	fmt.Scanln(&num2)
	fmt.Print("Enter the arithmetic operation (+, -, *, /): ")
	fmt.Scanln(&choice)
	switch choice {
	case "+":
		fmt.Printf("%.2f + %.2f = %.2f\n", num1, num2,
	num1+num2)
	case "-":
		fmt.Printf("%.2f - %.2f = %.2f\n", num1, num2,
	num1-num2)
	case "*":
		fmt.Printf("%.2f * %.2f = %.2f\n", num1, num2,
	num1*num2)
	case "/":
		if num2 != 0 {
			fmt.Printf("%.2f / %.2f = %.2f\n", num1, num2,
	num1/num2)
		} else {
			fmt.Println("Error: Division by zero is not
	allowed.")
		}
	default:
		fmt.Println("Invalid arithmetic operation.")
	}
}
```

OR

B) Write a program in GO language to accept n student details like roll_no, stud_name, mark1,mark2, mark3. Calculate the total and average of marks using structure.

Solution:-

```
package main

import "fmt"

type Student struct {
	rno   int
	sname string
	mark1 int
	mark2 int
	mark3 int
}

func totalMark(s Student) int {
	return s.mark1 + s.mark2 + s.mark3
}
func avgMark(s Student) float64 {
	return float64(totalMark(s)) / 3.0
}
func main() {
	var n int
	fmt.Println("Enter the number of students:")
	fmt.Scanln(&n)
	// Initialize an array of Student structs
	S := make([]Student, n)
	// Accept details for each student
	for i := 0; i < n; i++ {
		fmt.Println("Enter details for Student", i+1)
		fmt.Println("Enter roll no:")
		fmt.Scanln(&S[i].rno)
		fmt.Println("Enter name:")
		fmt.Scanln(&S[i].sname)
		fmt.Println("Enter mark1:")
		fmt.Scanln(&S[i].mark1)
```

```
            fmt.Println("Enter mark2:")
            fmt.Scanln(&S[i].mark2)
            fmt.Println("Enter mark3:")
            fmt.Scanln(&S[i].mark3)
        }
        // Display total and average marks for each student
        fmt.Println("\nStudent details:")
        for i, student := range S {
            total := totalMark(student)
            average := avgMark(student)
            fmt.Printf("Student %d: Roll No: %d, Name: %s, Total
        Marks: %d, Average Marks: %.2f\n", i+1, student.rno,
        student.sname, total, average)
        }
    }
```

.....

Slip 2

Q1. A) Write a program in GO language to print Fibonacci series of nterms.

Solutions:_

```
package main


import "fmt"


func fibonacci(n int) int {
	if n <= 1 {
		return n
	}
	return fibonacci(n-1) + fibonacci(n-2)
}
func printFibonacciSeries(n int) {
	for i := 0; i < n; i++ {
		fmt.Printf("%d ", fibonacci(i))
	}
	fmt.Println()
}
func main() {
	var num int
	fmt.Print("Enter the number of terms in the Fibonacci series: ")
	fmt.Scan(&num)
```

```
	fmt.Printf("Fibonacci series up to %d terms: ", num)

	printFibonacciSeries(num)

}
```

OR

B) Write a program in GO language to print file information.

Solution:-

```
package main

import (
	"fmt"
	"os"
)

func main() {
	// Specify the file path
	filePath := "example.txt"
	// Get file information
	fileInfo, err := os.Stat(filePath)
	if err != nil {
		fmt.Println("Error:", err)
		return
	}
	// Print file information
	fmt.Println("File Name:", fileInfo.Name())
	fmt.Println("Size (bytes):", fileInfo.Size())
	fmt.Println("Mode:", fileInfo.Mode())
	fmt.Println("Last Modified:", fileInfo.ModTime())
	fmt.Println("Is Directory:", fileInfo.IsDir())
}
```

****
****

Q1. A) Write a program in the GO language using function to check whether accepts number is palindrome or not.

Solution:

```
package main

import (
    "fmt"
)

// Function to check if a number is palindrome or not
func isPalindrome(num int) bool {
    original := num
    reverse := 0

    // Reverse the number
    for num > 0 {
        remainder := num % 10
        reverse = reverse*10 + remainder
        num /= 10
    }

    // Check if the reversed number is equal to the original
    return original == reverse
}

func main() {
    var num int
    fmt.Print("Enter a number: ")
    fmt.Scan(&num)

    if isPalindrome(num) {
        fmt.Println(num, "is a palindrome.")
    } else {
        fmt.Println(num, "is not a palindrome.")
    }
}
```

OR

B) Write a Program in GO language to accept n records of employee information (eno,ename,salary) and display record of employees having maximum salary.

Solution:-

```
package main

import "fmt"

type Employee struct {
	eno    int
	ename  string
	salary int
}

func maxSalary(E []Employee, n int) {
	max := E[0].salary
	for i := 1; i < n; i++ {
		if max <= E[i].salary {
			max = E[i].salary
		}
	}
	for i := 0; i < n; i++ {
		if max == E[i].salary {
			fmt.Println("Display employee details with maximum salary:-
")
			fmt.Println("eno:- ", E[i].eno)
			fmt.Println("ename:- ", E[i].ename)
			fmt.Println("Salary:- ", E[i].salary)
			break
		}
	}
}
func main() {
	var n int
	fmt.Println("Enter n:- ")
	fmt.Scanln(&n)
	emp := make([]Employee, n)
	fmt.Println()
	for i := 0; i < n; i++ {
```

```
            fmt.Println("Enter employee number:")
            fmt.Scanln(&emp[i].eno)
            fmt.Println("Enter employee name:")
            fmt.Scanln(&emp[i].ename)
            fmt.Println("Enter employee salary:")
            fmt.Scanln(&emp[i].salary)
        }
        maxSalary(emp, n)
}
```

****
****

Q1. A) Write a program in GO language to print a recursive sum of digits of a given number.

Solution:

```
package main

import (
	"fmt"
)

// Function to calculate the sum of digits recursively
func recursiveSumOfDigits(num int) int {
	// Base case: If the number is less than 10, return the number itself
	if num < 10 {
		return num
	}
	// Recursive case: Calculate the sum of digits by recursively summing the digits of the number
	return num%10 + recursiveSumOfDigits(num/10)
}

func main() {
	// Input number from user
	var num int
	fmt.Print("Enter a number: ")
	fmt.Scan(&num)

	// Calculate the recursive sum of digits
	sum := recursiveSumOfDigits(num)

	// Print the result
	fmt.Println("The recursive sum of digits of", num, "is:", sum)
}
```

OR

B) Write a program in GO language to sort array elements in ascending order.

Solution:

```
package main

import (
	"fmt"
	"sort"
)

func main() {
	var s int
	fmt.Println("Enter the size of the array to sort:")
	fmt.Scanln(&s)
	var a = make([]int, s)
	fmt.Println("Enter the array:")
	for i := 0; i < s; i++ {
		fmt.Printf("Enter element %d", i)
		fmt.Scanln(&a[i])
	}
	sort.Ints(a)
	fmt.Println(a)
}
```

****
****

Q1. A) Write a program in GO language program to create Text file

Solution:-

```
package main


import (
    "bufio"
    "fmt"
    "os"
)


func main() {
    // Specify the file name
    fileName := "output.txt"
    // Create the file
    file, err := os.Create(fileName)
    if err != nil {
        fmt.Println("Error:", err)
        return
    }
    defer file.Close()
    // Write content to the file
    content := "Hello, this is a text file created using Go programming
    language."
    writer := bufio.NewWriter(file)
    _, err = writer.WriteString(content)
```

```
    if err != nil {
        fmt.Println("Error:", err)
        return
    }
    writer.Flush()
    fmt.Println("Text file", fileName, "has been created successfully.")
}
```

OR

B) Write a program in GO language to accept n records of employee information (eno,ename,salary) and display records of employees having minimum salary.

Solution:

```
package main

import "fmt"

type Employee struct {
        eno    int
        ename  string
        salary int
}

func minSalary(E []Employee, n int) {
        min := E[0].salary
        for i := 1; i < n; i++ {
                if min >= E[i].salary {
                        min = E[i].salary
                }
        }
        for i := 0; i < n; i++ {
                if min == E[i].salary {
                        fmt.Println("eno:- ", E[i].eno)
                        fmt.Println("ename:- ", E[i].ename)
                        fmt.Println("Salary:- ", E[i].salary)
                        break
```

```
            }
        }
}
func main() {
        var n int
        fmt.Println("Enter n:- ")
        fmt.Scanln(&n)
        emp := make([]Employee, n)
        fmt.Println()
        for i := 0; i < n; i++ {
                fmt.Println("Enter employee number:")
                fmt.Scanln(&emp[i].eno)
                fmt.Println("Enter employee name:")
                fmt.Scanln(&emp[i].ename)
                fmt.Println("Enter employee salary:")
                fmt.Scanln(&emp[i].salary)
        }
        minSalary(emp, n)
}
**************************************************************************
   **********************
```

Slip 6

Q1. A) Write a program in GO language to accept two matrices and display its multiplication

Solution:

```
package main

import (
	"fmt"
)

func main() {
	var rows1, cols1 int
	fmt.Println("Enter the dimensions of the first matrix:")
	fmt.Print("Number of rows: ")
	fmt.Scanln(&rows1)
	fmt.Print("Number of columns: ")
	fmt.Scanln(&cols1)
	matrix1 := make([][]int, rows1)
	fmt.Println("Enter the elements of the first matrix:")
	for i := range matrix1 {
		matrix1[i] = make([]int, cols1)
		for j := range matrix1[i] {
			fmt.Printf("Enter element [%d][%d]: ", i, j)
			fmt.Scanln(&matrix1[i][j])
		}
	}
	var rows2, cols2 int
	fmt.Println("\nEnter the dimensions of the second matrix:")
	fmt.Print("Number of rows: ")
	fmt.Scanln(&rows2)
	fmt.Print("Number of columns: ")
	fmt.Scanln(&cols2)
	if cols1 != rows2 {
		fmt.Println("Error: Number of columns in the first matrix must be equal to the number of rows in the second matrix for multiplication.")
		return
	}
	matrix2 := make([][]int, rows2)
	fmt.Println("Enter the elements of the second matrix:")
```

```
	for i := range matrix2 {
		matrix2[i] = make([]int, cols2)
		for j := range matrix2[i] {
			fmt.Printf("Enter element [%d][%d]: ", i, j)
			fmt.Scanln(&matrix2[i][j])
		}
	}
	result := multiplyMatrices(matrix1, matrix2)
	fmt.Println("\nResult of matrix multiplication:")
	displayMatrix(result)
}
func multiplyMatrices(matrix1, matrix2 [][]int) [][]int {
	rows1, cols1 := len(matrix1), len(matrix1[0])
	_, cols2 := len(matrix2), len(matrix2[0])
	result := make([][]int, rows1)
	for i := range result {
		result[i] = make([]int, cols2)
	}
	for i := 0; i < rows1; i++ {
		for j := 0; j < cols2; j++ {
			for k := 0; k < cols1; k++ {
				result[i][j] += matrix1[i][k] * matrix2[k][j]
			}
		}
	}
	return result
}
func displayMatrix(matrix [][]int) {
	for _, row := range matrix {
		for _, element := range row {
			fmt.Printf("%d\t", element)
		}
		fmt.Println()
	}
}
```

OR

B) Write a program in GO language to copy all elements of one array into another using a method.

Solution:

```
package main

import "fmt"

// Function to copy elements from source array to destination array
func copyArray(source []int, destination []int) {
    // Iterate through the source array and copy each element to the destination array
    for i := 0; i < len(source); i++ {
        destination[i] = source[i]
    }
}

func main() {
    // Source array
    source := []int{1, 2, 3, 4, 5}

    // Destination array
    destination := make([]int, len(source))

    // Copy elements from source to destination array
    copyArray(source, destination)

    // Print the destination array to verify
    fmt.Println("Destination Array:", destination)
}
```

****
****

Q1. A) Write a program in GO language to accept one matrix and display its transpose.

Solution:

```
package main

import (
    "fmt"
)

// Function to calculate transpose of a matrix
func transpose(matrix [][]int) [][]int {
    rows := len(matrix)
    cols := len(matrix[0])

    // Initialize a new matrix to store the transpose
    transposed := make([][]int, cols)
    for i := range transposed {
        transposed[i] = make([]int, rows)
    }

    // Calculate transpose by swapping rows and columns
    for i := 0; i < rows; i++ {
        for j := 0; j < cols; j++ {
            transposed[j][i] = matrix[i][j]
        }
    }

    return transposed
}

func main() {
    var rows, cols int

    fmt.Print("Enter number of rows: ")
    fmt.Scan(&rows)

    fmt.Print("Enter number of columns: ")
    fmt.Scan(&cols)
```

```
    // Initialize the matrix
    matrix := make([][]int, rows)
    fmt.Println("Enter the elements of the matrix:")
    for i := 0; i < rows; i++ {
        matrix[i] = make([]int, cols)
        for j := 0; j < cols; j++ {
            fmt.Printf("Enter element [%d][%d]: ", i, j)
            fmt.Scan(&matrix[i][j])
        }
    }

    // Display the original matrix
    fmt.Println("Original Matrix:")
    for i := 0; i < rows; i++ {
        for j := 0; j < cols; j++ {
            fmt.Printf("%d ", matrix[i][j])
        }
        fmt.Println()
    }

    // Calculate and display the transpose of the matrix
    transposed := transpose(matrix)
    fmt.Println("Transpose of the Matrix:")
    for i := 0; i < cols; i++ {
        for j := 0; j < rows; j++ {
            fmt.Printf("%d ", transposed[i][j])
        }
        fmt.Println()
    }
}
```

OR

B) Write a program in GO language to create structure student. Write A method show() whose receiver is a pointer of struct student.

Solution:-

```
package main

import "fmt"

type student struct {
        Name  string
        Age   int
```

```
        Grade string
}

func (s *student) show() {
        fmt.Printf("Name: %s\n", s.Name)
        fmt.Printf("Age: %d\n", s.Age)
        fmt.Printf("Grade: %s\n", s.Grade)
}
func main() {
        s := &student{
                Name:  "John Doe",
                Age:   18,
                Grade: "A",
        }
        s.show()
}
```

****
****

Q1. A) Write a program in GO language to accept the book details such as BookID, Title, Author, Price. Read and display the details of 'n' number of books.

Solution:-

```
package main

import (
	"fmt"
)

type book struct {
	bookId int
	title  string
	author string
	price  int
}

func main() {
	var n int
	fmt.Println("Enter n:- ")
	fmt.Scanln(&n)
	books := make([]book, n)
	for i := 0; i < n; i++ {
		fmt.Println("Enter book id:= ")
		fmt.Scanln(&books[i].bookId)
		fmt.Println("Enter Title:- ")
		fmt.Scanln(&books[i].title)
		fmt.Println("Enter author:- ")
		fmt.Scanln(&books[i].author)
		fmt.Println("Enter Price:- ")
		fmt.Scanln(&books[i].price)
	}
	for i := 0; i < n; i++ {
		fmt.Println("book id:= ", books[i].bookId)
		fmt.Println("Title:- ", books[i].title)
		fmt.Println("author:- ", books[i].author)
		fmt.Println("Price:- ", books[i].price)
```

```
        }
    }
```

OR

B) Write a program in GO language to create an interface shape that includes area and perimeter. Implements these methods in circle and rectangle type.

Solution:

```
package main

import (
    "fmt"
    "math"
)

type Shape interface {
    Area() float64
    Perimeter() float64
}
type Circle struct {
    Radius float64
}

func (c Circle) Area() float64 {
    return math.Pi * c.Radius * c.Radius
}
func (c Circle) Perimeter() float64 {
    return 2 * math.Pi * c.Radius
}

type Rectangle struct {
    Length float64
    Width  float64
}

func (r Rectangle) Area() float64 {
    return r.Length * r.Width
}
func (r Rectangle) Perimeter() float64 {
    return 2 * (r.Length + r.Width)
}
func main() {
    circle := Circle{Radius: 5}
    rectangle := Rectangle{Length: 4, Width: 3}
```

```
        fmt.Println("Circle:")
        fmt.Printf("Area: %.2f\n", circle.Area())
        fmt.Printf("Perimeter: %.2f\n", circle.Perimeter())
        fmt.Println("\nRectangle:")
        fmt.Printf("Area: %.2f\n", rectangle.Area())
        fmt.Printf("Perimeter: %.2f\n", rectangle.Perimeter())
    }
```

****
****

Slip 9

Q1. A) Write a program in GO language using a function to check whether the accepted number is palindrome or not.

Solution:

```
package main

import (
    "fmt"
    "strconv"
)

// Function to check if a number is palindrome or not
func isPalindrome(num int) bool {
    // Convert the number to a string
    numStr := strconv.Itoa(num)

    // Initialize left and right pointers
    left, right := 0, len(numStr)-1

    // Iterate until left pointer is less than or equal to right pointer
    for left <= right {
        // If characters at left and right pointers are not equal, return false
        if numStr[left] != numStr[right] {
            return false
        }
        // Move the pointers towards each other
        left++
        right--
    }
    // If loop completes without returning false, the number is a palindrome
    return true
}

func main() {
    // Input number from user
    var num int
    fmt.Print("Enter a number: ")
    fmt.Scan(&num)

    // Check if the number is palindrome
    if isPalindrome(num) {
        fmt.Println(num, "is a palindrome.")
    } else {
        fmt.Println(num, "is not a palindrome.")
```

```
    }
}
```

OR

B) Write a program in GO language to create an interface shape that includes area and volume. Implements these methods in square and rectangle type.

Solution:-

```
package main

import (
        "fmt"
        "math"
)

type Shape interface {
        Area() float64
        Perimeter() float64
}
type Circle struct {
        Radius float64
}

func (c Circle) Area() float64 {
        return math.Pi * c.Radius * c.Radius
}
func (c Circle) Perimeter() float64 {
        return 2 * math.Pi * c.Radius
}

type Rectangle struct {
        Length float64
        Width  float64
}

func (r Rectangle) Area() float64 {
        return r.Length * r.Width
}
func (r Rectangle) Perimeter() float64 {
        return 2 * (r.Length + r.Width)
```

```
}
func main() {
	circle := Circle{Radius: 5}
	rectangle := Rectangle{Length: 4, Width: 3}
	fmt.Println("Circle:")
	fmt.Printf("Area: %.2f\n", circle.Area())
	fmt.Printf("Perimeter: %.2f\n", circle.Perimeter())
	fmt.Println("\nRectangle:")
	fmt.Printf("Area: %.2f\n", rectangle.Area())
	fmt.Printf("Perimeter: %.2f\n", rectangle.Perimeter())
}
```

****
****

Slip 10

Q1. A) Write a program in GO language to create an interface and display its values with the help of type assertion.

Solution:

```
package main

import "fmt"

// Define an interface
type Shape interface {
	Area() float64
}

// Define a struct for Circle
type Circle struct {
	Radius float64
}

// Implement the Area() method for Circle
func (c Circle) Area() float64 {
	return 3.14 * c.Radius * c.Radius
}

// Define a struct for Rectangle
type Rectangle struct {
	Length float64
```

```
    Width  float64
}

// Implement the Area() method for Rectangle
func (r Rectangle) Area() float64 {
    return r.Length * r.Width
}
func main() {
    // Create instances of Circle and Rectangle
    circle := Circle{Radius: 5}
    rectangle := Rectangle{Length: 4, Width: 3}
    // Create a slice of Shape interface containing Circle and Rectangle
    shapes := []Shape{circle, rectangle}
    // Iterate through the shapes and display their areas
    for _, shape := range shapes {
        // Check the underlying type of the shape
        switch s := shape.(type) {
        case Circle:
            fmt.Printf("Circle Area: %.2f\n", s.Area())
        case Rectangle:
            fmt.Printf("Rectangle Area: %.2f\n", s.Area())
        default:
            fmt.Println("Unknown shape")
        }
    }
}
```

OR

B) Write a program in GO language to read and write Fibonacci series to the using channel.

Solution:

```
package main

import (
	"fmt"
)

// fibonacciGenerator generates Fibonacci numbers up to n and sends them to the channel ch
func fibonacciGenerator(n int, ch chan<- int) {
	a, b := 0, 1
	for i := 0; i < n; i++ {
		ch <- a
		a, b = b, a+b
	}
	close(ch) // Close the channel after sending all Fibonacci numbers
}

// fibonacciReader reads Fibonacci numbers from the channel ch and prints them
func fibonacciReader(ch <-chan int) {
	for num := range ch {
		fmt.Println(num)
	}
}
func main() {
	n := 10 // Number of Fibonacci numbers to generate
	// Create a channel
	ch := make(chan int)
	// Start a goroutine to generate Fibonacci numbers and send them to the channel
	go fibonacciGenerator(n, ch)
	// Start a goroutine to read Fibonacci numbers from the channel and print them
	go fibonacciReader(ch)
	// Wait for the goroutines to finish
	var input string
```

```
        fmt.Println("Press Enter to exit")
        fmt.Scanln(&input)
}
```

****
****

Q1. A) Write a program in GO language to check whether the accepted number is two digit or not.

Solution:

```
package main


import "fmt"


func main() {
	var x int
	fmt.Println("Enter number:== ")
	fmt.Scanln(&x)
	if x >= 10 && x <= 99 {
		fmt.Println("Number is double digit")
	} else {
		fmt.Print("Number is not a double digit")
	}
}
```

OR

B) Write a program in GO language to create a buffered channel, store few values in it and find channel capacity and length. Read values from channel and find modified length of a channel

Solution:

```
package main

import (
	"fmt"
)

func main() {
	// Creating a buffered channel with capacity 3
	ch := make(chan int, 3)
	// Storing values in the channel
	ch <- 1
	ch <- 2
```

```
        ch <- 3
        // Finding channel capacity
        capacity := cap(ch)
        fmt.Println("Channel capacity:", capacity)
        // Finding channel length
        length := len(ch)
        fmt.Println("Channel length before reading:", length)
        // Reading from channel
        for i := 0; i < length; i++ {
                fmt.Println("Value read from channel:", <-ch)
        }
        // Finding modified channel length
        length = len(ch)
        fmt.Println("Channel length after reading:", length)
}
```

****
****

Slip 12

Q1. A) Write a program in GO language to swap two numbers using call by reference concept

Solution:

```
package main

import "fmt"

// Function to swap two numbers using call by reference
func swapByReference(a *int, b *int) {
   temp := *a
   *a = *b
   *b = temp
}

func main() {
   var num1, num2 int

   // Input two numbers from the user
   fmt.Print("Enter the first number: ")
   fmt.Scan(&num1)
   fmt.Print("Enter the second number: ")
   fmt.Scan(&num2)

   // Display the numbers before swapping
   fmt.Println("Before swapping:")
   fmt.Println("First number:", num1)
   fmt.Println("Second number:", num2)

   // Swap the numbers using call by reference
   swapByReference(&num1, &num2)

   // Display the numbers after swapping
   fmt.Println("\nAfter swapping:")
   fmt.Println("First number:", num1)
   fmt.Println("Second number:", num2)
}
```

OR

B) Write a program in GO language that creates a slice of integers, checks numbers from the slice are even or odd and further sent to respective go routines through channel and display values received by goroutines.

Solution:

```
package main

import (
	"fmt"
)

func isEven(num int) bool {
	return num%2 == 0
}
func evenOddChecker(numbers []int, evenCh chan<- int, oddCh chan<- int) {
	for _, num := range numbers {
		if isEven(num) {
			evenCh <- num
		} else {
			oddCh <- num
		}
	}
	close(evenCh)
	close(oddCh)
}
func evenPrinter(evenCh <-chan int) {
	for num := range evenCh {
		fmt.Println("Even:", num)
	}
}
func oddPrinter(oddCh <-chan int) {
	for num := range oddCh {
		fmt.Println("Odd:", num)
	}
}
func main() {
	numbers := []int{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
	evenCh := make(chan int)
	oddCh := make(chan int)
	go evenOddChecker(numbers, evenCh, oddCh)
	go evenPrinter(evenCh)
```

```
        go oddPrinter(oddCh)
        // Wait for goroutines to finish
        var input string
        fmt.Println("Press Enter to exit")
        fmt.Scanln(&input)
    }
```

Q1. A) Write a program in GO language to print sum of all even and odd numbers separately between 1 to 100.

Solution:

```
package main


import "fmt"


func main() {
    var sumEven, sumOdd int


    // Iterate from 1 to 100
    for i := 1; i <= 100; i++ {
        if i%2 == 0 {
            // If the number is even, add it to the sum of even numbers
            sumEven += i
        } else {
            // If the number is odd, add it to the sum of odd numbers
            sumOdd += i
        }
    }


    // Print the sum of even and odd numbers separately
    fmt.Println("Sum of even numbers between 1 to 100:", sumEven)
    fmt.Println("Sum of odd numbers between 1 to 100:", sumOdd)
}
```

OR

B) Write a function in GO language to find the square of a number and write a benchmark for it.

Solution:

```
package main

import (
	"fmt"
	"testing"
)

// Square returns the square of a given number
func Square(x int) int {
	return x * x
}
func TestSquare(t *testing.T) {
	tests := []struct {
		input    int
		expected int
	}{
		{0, 0},
		{1, 1},
		{-2, 4},
		{5, 25},
	}
	for _, test := range tests {
		result := Square(test.input)
		if result != test.expected {
			t.Errorf("Square(%d) = %d; want %d",
	test.input, result, test.expected)
		}
	}
}
func BenchmarkSquare(b *testing.B) {
	for i := 0; i < b.N; i++ {
		Square(10) // Square of number 10
	}
}
func main() {
	fmt.Println(Square(5)) // Test the function with an
	example
```

}

****
****

Q1. A) Write a program in GO language to demonstrate working of slices (like append, remove, copy etc.)

Solution:

```
package main

import (
	"fmt"
)

func main() {
	slice := []int{1, 2, 3, 4, 5}
	fmt.Println("Original Slice:", slice)
	slice = append(slice, 6)
	fmt.Println("Slice after appending 6:", slice)
	indexToRemove := 2
	if indexToRemove >= 0 && indexToRemove < len(slice) {
		slice = append(slice[:indexToRemove],
slice[indexToRemove+1:]...)
		fmt.Println("Slice after removing element at index 2:", slice)
	} else {
		fmt.Println("Index out of range. Cannot remove element.")
	}
	copySlice := make([]int, len(slice))
	copy(copySlice, slice)
	fmt.Println("Copied Slice:", copySlice)
}
```

OR

B) Write a program in GO language using go routine and channel that will print the sum of the squares and cubes of the individual digits of a number. Example if number is 123 then squares = (1 * 1) + (2 * 2) + (3 * 3)
cubes = (1 * 1 * 1) + (2 * 2 * 2) + (3 * 3 * 3).

Solution:

```
package main

import (
```

```
	"fmt"
)

func calculateSquaresAndCubes(num int, squaresCh chan int, cubesCh chan
int) {
	squareSum := 0
	cubeSum := 0
	for num > 0 {
		digit := num % 10
		squareSum += digit * digit
		cubeSum += digit * digit * digit
		num /= 10
	}
	squaresCh <- squareSum
	cubesCh <- cubeSum
}
func main() {
	num := 123
	squaresCh := make(chan int)
	cubesCh := make(chan int)
	go calculateSquaresAndCubes(num, squaresCh, cubesCh)
	squaresSum := <-squaresCh
	cubesSum := <-cubesCh
	fmt.Printf("Sum of squares: %d\n", squaresSum)
	fmt.Printf("Sum of cubes: %d\n", cubesSum)
}
```

Slip 15

Q1. A) Write a program in GO language to demonstrate function return multiple values.

Solution:

```
package main

import "fmt"

// Function to calculate the sum and product of two numbers
func sumAndProduct(a, b int) (int, int) {
    sum := a + b
    product := a * b
    return sum, product
}

func main() {
    // Call the function and capture the returned values
    sum, product := sumAndProduct(3, 4)

    // Print the returned values
    fmt.Println("Sum:", sum)
    fmt.Println("Product:", product)
}
```

OR

B) Write a program in GO language to read XML file into structure and display structure

Solution:

```
package main

import (
        "encoding/xml"
        "fmt"
        "io/ioutil"
        "os"
)

// Define a struct that matches the structure of the XML file
type Person struct {
        XMLName xml.Name `xml:"person"`
        Name    string   `xml:"name"`
        Age     int      `xml:"age"`
```

```
        City    string   `xml:"city"`
    }

    func main() {
        // Open the XML file
        xmlFile, err := os.Open("example.xml")
        if err != nil {
            fmt.Println("Error opening XML file:", err)
            return
        }
        defer xmlFile.Close()
        // Read the XML file content
        byteValue, err := ioutil.ReadAll(xmlFile)
        if err != nil {
            fmt.Println("Error reading XML file:", err)
            return
        }
        // Define a variable to store the decoded XML data
        var person Person
        // Unmarshal the XML data into the structure
        err = xml.Unmarshal(byteValue, &person)
        if err != nil {
            fmt.Println("Error unmarshalling XML:", err)
            return
        }
        // Print the structure
        fmt.Println("Name:", person.Name)
        fmt.Println("Age:", person.Age)
        fmt.Println("City:", person.City)
    }
```

Q1. A) Write a program in GO language to create a user defined package to find out the area of a rectangle.

Solution:

```
// rectangle.go

package geometry

// Area returns the area of a rectangle given its length and width

func Area(length, width float64) float64 {

return length * width

}

// main.go

package main

import (

"fmt"

"geometry" // Import the user-defined package

)

func main() {

length := 5.0

width := 3.0

// Calculate the area of the rectangle using the user-defined package

area := geometry.Area(length, width)

fmt.Printf("Area of the rectangle with length %.2f and width %.2f: %.2f\n",
    length, width, area)

}
```

OR

B) Write a program in GO language that prints out the numbers from 0 to 10, waiting between 0 and 250 ms after each one using the delay function.

Solution:

```
package main

import (
	"fmt"
	"math/rand"
	"time"
)

func delay(min, max int) {
	time.Sleep(time.Duration(rand.Intn(max-min+1)+min) *
time.Millisecond)
}
func main() {
	rand.Seed(time.Now().UnixNano())
	for i := 0; i <= 10; i++ {
		fmt.Println(i)
		delay(0, 250) // Delay between 0 and 250 milliseconds
	}
}
```

****
****

Q1. A) Write a program in GO language to illustrate the concept of returning multiple values from a function. (Add, Subtract, Multiply, Divide)

Solution:

```
package main

import "fmt"

// Function to perform arithmetic operations and return multiple values
func arithmeticOperations(a, b float64) (float64, float64, float64, float64) {
    sum := a + b
    difference := a - b
    product := a * b
    // Check if the second number is not zero to avoid division by zero
    var division float64
    if b != 0 {
        division = a / b
    } else {
        division = 0
    }
    return sum, difference, product, division
}

func main() {
    var num1, num2 float64

    // Input two numbers from the user
    fmt.Print("Enter the first number: ")
    fmt.Scan(&num1)
    fmt.Print("Enter the second number: ")
    fmt.Scan(&num2)
```

```
    // Perform arithmetic operations and capture returned values
    sum, difference, product, division := arithmeticOperations(num1,
num2)

    // Print the results
    fmt.Printf("Sum: %.2f\n", sum)
    fmt.Printf("Difference: %.2f\n", difference)
    fmt.Printf("Product: %.2f\n", product)
    fmt.Printf("Division: %.2f\n", division)
}
```

OR

B) Write a program in GO language to add or append content at the end of a text file

Solution:

```
package main

import (
    "fmt"
    "os"
)

func appendToFile(fileName string, content string) error {
    // Open the file in append mode with write-only permission and create it if
it doesn't exist
    file, err := os.OpenFile(fileName,
os.O_APPEND|os.O_WRONLY|os.O_CREATE, 0644)
    if err != nil {
        return err
    }
    defer file.Close()
    // Write the content to the end of the file
    if _, err := file.WriteString(content); err != nil {
        return err
    }
    return nil
}
func main() {
    fileName := "example.txt"
    content := "This content will be appended to the end of the file.\n"
    err := appendToFile(fileName, content)
```

```
	if err != nil {
		fmt.Println("Error:", err)
		return
	}
	fmt.Println("Content has been successfully appended to the file:",
fileName)
}
```

****
****

Q1 A) Write a program in GO language to print a multiplication table of number using function.

Solution:

```
package main


import "fmt"


// Function to print multiplication table of a given number
func multiplicationTable(number, times int) {
    fmt.Printf("Multiplication Table of %d:\n", number)
    for i := 1; i <= times; i++ {
        fmt.Printf("%d x %d = %d\n", number, i, number*i)
    }
}


func main() {
    var number, times int


    // Input the number and the number of times to multiply
    fmt.Print("Enter the number: ")
    fmt.Scan(&number)
    fmt.Print("Enter the number of times: ")
    fmt.Scan(&times)
```

```
    // Print the multiplication table
    multiplicationTable(number, times)
}
```

OR

B) Write a program in GO language using a user defined package calculator that performs one calculator operation as per the user'schoice.

Solution:

```
// calculator.go
package calculator
import "errors"
// Operation represents a calculator operation
type Operation int
const (
Addition Operation = iota
Subtraction
Multiplication
Division
)
// Calculate performs the specified operation on two numbers
func Calculate(operation Operation, num1, num2 float64) (float64, error) {
switch operation {
case Addition:
return num1 + num2, nil
case Subtraction:
return num1 - num2, nil
case Multiplication:
return num1 * num2, nil
case Division:
if num2 == 0 {
return 0, errors.New("division by zero")
}
return num1 / num2, nil
default:
return 0, errors.New("unsupported operation")
}
}
```

```
// main.go
package main
import (
"fmt"
"calculator" // Import the user-defined package
)
func main() {
num1 := 10.0
num2 := 5.0
operation := calculator.Addition // Change this to the desired operation
result, err := calculator.Calculate(operation, num1, num2)
if err != nil {
fmt.Println("Error:", err)
return
}
fmt.Printf("Result of %.2f %s %.2f: %.2f\n", num1,
getOperationSymbol(operation), num2, result)
}
func getOperationSymbol(operation calculator.Operation) string {
switch operation {
case calculator.Addition:
return "+"
case calculator.Subtraction:
return "-"
case calculator.Multiplication:
return "*"
case calculator.Division:
return "/"
default:
return "?"
}
}
```

****

****

Q1 A) Write a program in GO language to illustrate the function returning multiple values(add, subtract).

Solution:

```
package main

import "fmt"

// Function to perform addition and subtraction and return multiple values
func addAndSubtract(a, b int) (int, int) {
    sum := a + b
    difference := a - b
    return sum, difference
}

func main() {
    var num1, num2 int

    // Input two numbers from the user
    fmt.Print("Enter the first number: ")
    fmt.Scan(&num1)
    fmt.Print("Enter the second number: ")
    fmt.Scan(&num2)

    // Call the function and capture the returned values
    sum, difference := addAndSubtract(num1, num2)

    // Print the returned values
    fmt.Printf("Sum: %d\n", sum)
    fmt.Printf("Difference: %d\n", difference)
}
```

OR

B) Write a program in the GO language program to open a file in READ only mode.

Solution:

```
package main

import (
        "fmt"
        "os"
)
```

```
func main() {
	// Specify the file path
	filePath := "example.txt"
	// Open the file in read-only mode
	file, err := os.OpenFile(filePath, os.O_RDONLY, 0644)
	if err != nil {
		fmt.Println("Error:", err)
		return
	}
	defer file.Close()
	fmt.Println("File", filePath, "opened successfully in read-only mode.")
}
```

****
****

Q1. A) Write a program in Go language to add or append content at the end of a text file.

Solution:

```
package main


import (
	"fmt"
	"os"
)


func appendToFile(fileName, content string) error {
	// Open the file in append mode with write-only permission
	file, err := os.OpenFile(fileName, os.O_APPEND|os.O_WRONLY, 0644)
	if err != nil {
		return err
	}
	defer file.Close()
	// Write the content at the end of the file
	if _, err := file.WriteString(content); err != nil {
		return err
	}
	return nil
}
```

```
func main() {

	fileName := "example.txt"

	content := "\nThis is the content to be appended."

	err := appendToFile(fileName, content)

	if err != nil {

		fmt.Println("Error:", err)

		return

	}

	fmt.Println("Content has been successfully appended to the file:",
fileName)

}
```

OR

B) Write a program in Go language how to create a channel and illustrate how to close a channel using for range loop and close function.

Solution:

```
package main

import (
    "fmt"
)

func sender(ch chan<- int) {
    for i := 0; i < 5; i++ {
        ch <- i
    }
    close(ch) // Close the channel when done sending values
}
func main() {
    // Create an integer channel
    ch := make(chan int)
    // Start a goroutine to send values to the channel
    go sender(ch)
    // Use for range loop to receive values from the channel until it's closed
    for num := range ch {
```

```
            fmt.Println("Received:", num)
        }
        // Creating a new channel
        ch2 := make(chan int)
        // Start a goroutine to send values to the channel
        go sender(ch2)
        // Manually close the channel using the close function
        close(ch2)
        // Use for range loop to receive values from the channel until it's closed
        for num := range ch2 {
            fmt.Println("Received from ch2:", num)
        }
    }
```

****

****