**GO SLIP SOLUTION:**

**Slip 1:**

Que1 - A=>

package main

import "fmt"

func main() {

var num1, num2 float64

var choice rune

fmt.Println("Enter first number:")

fmt.Scanln(&num1)

fmt.Println("Enter second number:")

fmt.Scanln(&num2)

fmt.Println("Enter the arithmetic operation (+, -, \*, /):")

fmt.Scanf("%c", &choice)

var result float64

switch choice {

case '+': result = num1 + num2

case '-': result = num1 - num2

case '\*': result = num1 \* num2

case '/': if num2 != 0 {

result = num1 / num } else {

fmt.Println("Error: Division by zero")

return }

default: fmt.Println("Invalid operation")

return

}

fmt.Printf("Result: %f\n", result)

}

Que1- B =>

package main

import "fmt"

type Student struct {

RollNo int

Name string

Marks [3]float64

Total float64

Average float64

}

func main() {

var n int

fmt.Println("Enter the number of students:")

fmt.Scanln(&n)

students := make([]Student, n)

for i := 0; i < n; i++ {

fmt.Printf("\nEnter details for student %d:\n", i+1)

fmt.Println("Roll number:")

fmt.Scanln(&students[i].RollNo)

fmt.Println("Name:")

fmt.Scanln(&students[i].Name)

fmt.Println("Marks in three subjects:")

for j := 0; j < 3; j++ {

fmt.Printf("Mark %d: ", j+1)

fmt.Scanln(&students[i].Marks[j])

}

students[i].Total, students[i].Average = calculateTotalAndAverage(students[i].Marks)

}

fmt.Println("\nStudent details with total and average marks:")

for i, student := range students {

fmt.Printf("\nStudent %d\n", i+1)

fmt.Printf("Roll number: %d\n", student.RollNo)

fmt.Printf("Name: %s\n", student.Name)

fmt.Printf("Total marks: %.2f\n", student.Total)

fmt.Printf("Average marks: %.2f\n", student.Average)

}

}

func calculateTotalAndAverage(marks [3]float64) (float64, float64) {

total := marks[0] + marks[1] + marks[2]

average := total / 3.0

return total, average

}

**SLIP 2:**

QUE1- A=>

package main

import "fmt"

func fibonacci(n int) []int {

fib := make([]int, n)

fib[0], fib[1] = 0, 1

for i := 2; i < n; i++ {

fib[i] = fib[i-1] + fib[i-2]

}

return fib

}

func main() {

var n int

fmt.Println("Enter the number of terms in Fibonacci series:")

fmt.Scanln(&n)

if n <= 0 {

fmt.Println("Please enter a positive number.")

return

}

fibSeries := fibonacci(n)

fmt.Printf("Fibonacci series of %d terms: ", n)

for \_, num := range fibSeries {

fmt.Printf("%d ", num)

}

fmt.Println()

}

QUE1- B =>

package main

import (

"fmt"

"os"

)

func main() {

// Get file information

fileInfo, err := os.Stat("example.txt")

if err != nil {

fmt.Println("Error:", err)

return

}

// Print file information

fmt.Println("File Name:", fileInfo.Name())

fmt.Println("Size:", fileInfo.Size(), "bytes")

fmt.Println("Mode:", fileInfo.Mode())

fmt.Println("Last Modified:", fileInfo.ModTime())

fmt.Println("Is Directory:", fileInfo.IsDir())

}

Save this code in a file, for example, **file\_info.go**. Then, compile and run it using the following commands in your terminal:

**go build file\_info.go**

**./file\_info**

This program will print information about the file named **example.txt** in the current directory (where you run the program), including its name, size, mode, last modified time, and whether it is a directory or not. If the file doesn't exist or there's an error while getting its information, it will print the error message.

**SLIP 3:**

QUE1- A=>

package main

import (

"fmt"

)

func isPalindrome(num int) bool {

original := num

reversed := 0

for num > 0 {

remainder := num % 10

reversed = reversed\*10 + remainder

num /= 10

}

return original == reversed

}

func main() {

var number int

fmt.Println("Enter a number to check if it's a palindrome:")

fmt.Scanln(&number)

if isPalindrome(number) {

fmt.Println("The number is a palindrome.")

} else {

fmt.Println("The number is not a palindrome.")

}

}

QUE1 -B=>

package main

import "fmt"

type Employee struct {

Eno int

Ename string

Salary float64 }

func main() {

var n int

fmt.Println("Enter the number of employees:")

fmt.Scanln(&n)

employees := make([]Employee, n)

maxSalary := 0.0

for i := 0; i < n; i++ {

fmt.Printf("\nEnter details for employee %d:\n", i+1)

fmt.Println("Employee number:")

fmt.Scanln(&employees[i].Eno)

fmt.Println("Employee name:")

fmt.Scanln(&employees[i].Ename)

fmt.Println("Salary:")

fmt.Scanln(&employees[i].Salary)

if employees[i].Salary > maxSalary {

maxSalary = employees[i].Salary }

}

fmt.Println("\nEmployee(s) with maximum salary:")

for \_, emp := range employees {

if emp.Salary == maxSalary {

fmt.Printf("Employee number: %d, Name: %s, Salary: %.2f\n", emp.Eno, emp.Ename, emp.Salary) } } }

**SLIP 4:**

QUE1- A =>

package main

import "fmt"

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: sum the last digit with the sum of digits of the remaining number

return num%10 + recursiveSumOfDigits(num/10)

}

func main() {

var number int

fmt.Println("Enter a number:")

fmt.Scanln(&number)

sum := recursiveSumOfDigits(number)

fmt.Printf("Recursive sum of digits of %d is %d\n", number, sum)

}

QUE2- B=>

package main

import "fmt"

func bubbleSort(arr []int) {

n := len(arr)

for i := 0; i < n-1; i++ {

for j := 0; j < n-i-1; j++ {

if arr[j] > arr[j+1] {

// Swap arr[j] and arr[j+1]

arr[j], arr[j+1] = arr[j+1], arr[j]

}

}

}

}

func main() {

var size int

fmt.Println("Enter the size of the array:")

fmt.Scanln(&size)

arr := make([]int, size)

fmt.Println("Enter the elements of the array:")

for i := 0; i < size; i++ {

fmt.Printf("Element %d: ", i+1)

fmt.Scanln(&arr[i])

}

fmt.Println("Original array:", arr)

// Sorting the array

bubbleSort(arr)

fmt.Println("Sorted array in ascending order:", arr)

}

**SLIP 5:**

QUE1- A=>

package main

import (

"fmt"

"os"

)

func main() {

// Open file for writing. If the file doesn't exist, create it.

file, err := os.Create("example.txt")

if err != nil {

fmt.Println("Error:", err)

return

}

defer file.Close()

// Write data to the file

\_, err = file.WriteString("Hello, world!\nThis is a text file created using Go.")

if err != nil {

fmt.Println("Error:", err)

return

fmt.Println("Text file created successfully.")

}

Save this code in a file, for example, **create\_text\_file.go**. Then, compile and run it using the following commands in your terminal:

**go build create\_text\_file.go**

**./create\_text\_file**

This program will create a file named example.txt in the current directory (where you run the program), and write the specified content into it. If the file already exists, it will be overwritten.

QUE1- B=>

package main import "fmt"

type Employee struct {

Eno int

Ename string

Salary float64

}

func main() {

var n int

fmt.Println("Enter the number of employees:")

fmt.Scanln(&n)

employees := make([]Employee, n)

minSalary := float64(0)

// Accepting employee information

for i := 0; i < n; i++ {

fmt.Printf("\nEnter details for employee %d:\n", i+1)

fmt.Println("Employee number:")

fmt.Scanln(&employees[i].Eno)

fmt.Println("Employee name:")

fmt.Scanln(&employees[i].Ename)

fmt.Println("Salary:")

fmt.Scanln(&employees[i].Salary)

// Initialize minSalary with the first employee's salary

if i == 0 || employees[i].Salary < minSalary {

minSalary = employees[i].Salary

}

}

// Displaying records of employees with minimum salary

fmt.Println("\nEmployee(s) with minimum salary:")

for \_, emp := range employees {

if emp.Salary == minSalary {

fmt.Printf("Employee number: %d, Name: %s, Salary: %.2f\n", emp.Eno, emp.Ename, emp.Salary)

}

} }

**SLIP 6:**

QUE1- A =>

package main

import "fmt"

func main() {

var rows1, cols1, rows2, cols2 int

// Accept dimensions of the first matrix

fmt.Println("Enter the number of rows of the first matrix:")

fmt.Scanln(&rows1)

fmt.Println("Enter the number of columns of the first matrix:")

fmt.Scanln(&cols1)

// Accept dimensions of the second matrix

fmt.Println("Enter the number of rows of the second matrix:")

fmt.Scanln(&rows2)

fmt.Println("Enter the number of columns of the second matrix:")

fmt.Scanln(&cols2)

// Check if multiplication is possible

if cols1 != rows2 {

fmt.Println("Multiplication is not possible. Number of columns of the first matrix should be equal to the number of rows of the second matrix.")

return

}

// Initialize matrices

matrix1 := make([][]int, rows1)

matrix2 := make([][]int, rows2)

fmt.Println("Enter elements of the first matrix:")

for i := 0; i < rows1; i++ {

matrix1[i] = make([]int, cols1)

for j := 0; j < cols1; j++ {

fmt.Printf("Element [%d][%d]: ", i+1, j+1)

fmt.Scanln(&matrix1[i][j])

}

}

fmt.Println("Enter elements of the second matrix:")

for i := 0; i < rows2; i++ {

matrix2[i] = make([]int, cols2)

for j := 0; j < cols2; j++ {

fmt.Printf("Element [%d][%d]: ", i+1, j+1)

fmt.Scanln(&matrix2[i][j])

}

}

// Perform matrix multiplication

result := multiplyMatrices(matrix1, matrix2)

// Display the result

fmt.Println("Result of matrix multiplication:")

for i := 0; i < rows1; i++ {

for j := 0; j < cols2; j++ {

fmt.Printf("%d\t", result[i][j])

}

fmt.Println()

}

}

func multiplyMatrices(matrix1, matrix2 [][]int) [][]int {

rows1, cols1 := len(matrix1), len(matrix1[0])

\_, cols2 := len(matrix2), len(matrix2[0])

result := make([][]int, rows1)

// Initialize result matrix

for i := range result {

result[i] = make([]int, cols2)

// Perform matrix multiplication

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

}

QUE1- B=>

package main

import "fmt"

// Function to copy elements from source array to destination array

func copyArray(src []int, dest []int) {

// Check if the source and destination arrays have the same length

if len(src) != len(dest) {

fmt.Println("Arrays have different lengths. Copy operation aborted.")

return

}

// Copy elements from source to destination array

for i := 0; i < len(src); i++ {

dest[i] = src[i]

}

}

func main() {

// Source array

source := []int{1, 2, 3, 4, 5}

// Destination array with the same length as the source array

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

// Copy elements from source to destination array using the method

copyArray(source, destination)

// Display the contents of the destination array

fmt.Println("Contents of the destination array after copying:")

fmt.Println(destination)

}

**SLIP 7:**

QUE1- A=>

package main

import "fmt"

func main() {

var rows, cols in

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)

for i := range matrix {

matrix[i] = make([]int, cols)

}

// Input matrix elements

fmt.Println("Enter matrix elements:")

for i := 0; i < rows; i++ {

for j := 0; j < cols; j++ {

fmt.Printf("Enter element [%d][%d]: ", i, j)

fmt.Scan(&matrix[i][j])

}

}

// Display original matrix

fmt.Println("Original Matrix:")

displayMatrix(matrix)

// Display transpose matrix

fmt.Println("Transpose Matrix:")

transposeMatrix := transpose(matrix)

displayMatrix(transposeMatrix)

}

// Function to compute transpose of a matrix

func transpose(matrix [][]int) [][]int {

rows := len(matrix)

cols := len(matrix[0])

transposeMatrix := make([][]int, cols)

for i := range transposeMatrix {

transposeMatrix[i] = make([]int, rows)

}

for i := 0; i < rows; i++ {

for j := 0; j < cols; j++ {

transposeMatrix[j][i] = matrix[i][j]

}

}

return transposeMatrix

}

// Function to display a matrix

func displayMatrix(matrix [][]int) {

for i := 0; i < len(matrix); i++ {

for j := 0; j < len(matrix[0]); j++ {

fmt.Printf("%d\t", matrix[i][j])

}

fmt.Println()

}

}

QUE1- B=>

package main

import "fmt"

// Define a struct called student

type student struct {

Name string

Age int

Grade float64

}

// Method to display student information

func (s \*student) show() {

fmt.Printf("Name: %s\n", s.Name)

fmt.Printf("Age: %d\n", s.Age)

fmt.Printf("Grade: %.2f\n", s.Grade)

}

func main() {

// Creating a student object

s := student{

Name: "John",

Age: 18,

Grade: 87.5,

}

// Calling the show() method

s.show()

}

**SLIP 8:**

QUE1- A=>

package main

import "fmt"

// Define a struct called book

type book struct {

BookID int

Title string

Author string

Price float64

}

func main() {

var n int

// Prompt user to enter the number of books

fmt.Print("Enter the number of books: ")

fmt.Scan(&n)

// Create a slice to store the book details

books := make([]book, n)

// Input book details

for i := 0; i < n; i++ {

fmt.Printf("Enter details for book %d:\n", i+1)

fmt.Print("BookID: ")

fmt.Scan(&books[i].BookID)

fmt.Print("Title: ")

fmt.Scan(&books[i].Title)

fmt.Print("Author: ")

fmt.Scan(&books[i].Author)

fmt.Print("Price: ")

fmt.Scan(&books[i].Price)

}

// Display book details

fmt.Println("\nBook Details:")

for i, bk := range books {

fmt.Printf("Book %d:\n", i+1)

fmt.Printf("BookID: %d\n", bk.BookID)

fmt.Printf("Title: %s\n", bk.Title)

fmt.Printf("Author: %s\n", bk.Author)

fmt.Printf("Price: %.2f\n", bk.Price)

fmt.Println()

}

}

QUE1- B=>

package main

import (

"fmt"

"math"

)

type Shape interface {

Area() float64

Perimeter() float64

}

// Circle type represents a circle

type Circle struct {

Radius float64

}

// Area calculates the area of a circle

func (c Circle) Area() float64 {

return math.Pi \* c.Radius \* c.Radius

}

// Perimeter calculates the perimeter of a circle

func (c Circle) Perimeter() float64 {

return 2 \* math.Pi \* c.Radius

}

// Rectangle type represents a rectangle

type Rectangle struct {

Width float64

Height float64

}

// Area calculates the area of a rectangle

func (r Rectangle) Area() float64 {

return r.Width \* r.Height

}

// Perimeter calculates the perimeter of a rectangle

func (r Rectangle) Perimeter() float64 {

return 2\*r.Width + 2\*r.Height

func main() {

// Create a circle

circle := Circle{Radius: 5}

// Create a rectangle

rectangle := Rectangle{Width: 4, Height: 3}

// Calculate and print area and perimeter of the circle

fmt.Printf("Circle\nArea: %.2f\nPerimeter: %.2f\n", circle.Area(), circle.Perimeter())

// Calculate and print area and perimeter of the rectangle

fmt.Printf("\nRectangle\nArea: %.2f\nPerimeter: %.2f\n", rectangle.Area(), rectangle.Perimeter())

}

**SLIP 9:**

QUE1- A=>

package main

import (

"fmt"

)

func isPalindrome(num int) bool {

original := num

reversed := 0

for num > 0 {

remainder := num % 10

reversed = reversed\*10 + remainder

num /= 10

}

return original == reversed

}

func main() {

var number int

fmt.Println("Enter a number to check if it's a palindrome:")

fmt.Scanln(&number)

if isPalindrome(number) {

fmt.Println("The number is a palindrome.")

} else {

fmt.Println("The number is not a palindrome.")

}

}

QUE1- B=>

package main

import "fmt"

// Shape interface defines methods for calculating area and volume

type Shape interface {

Area() float64

Volume() float64

}

// Square type represents a square

type Square struct {

SideLength float64

}

// Area calculates the area of a square

func (s Square) Area() float64 {

return s.SideLength \* s.SideLength

}

// Volume returns 0 for a square, as it's a 2D shape

func (s Square) Volume() float64 {

return 0

}

// Rectangle type represents a rectangle

type Rectangle struct {

Width float64

Height float64

}

// Area calculates the area of a rectangle

func (r Rectangle) Area() float64 {

return r.Width \* r.Height

}

// Volume returns 0 for a rectangle, as it's a 2D shape

func (r Rectangle) Volume() float64 {

return 0

}

func main() {

square := Square{SideLength: 5}

rectangle := Rectangle{Width: 4, Height: 3}

fmt.Printf("Square\nArea: %.2f\n", square.Area())

fmt.Printf("Volume: %.2f\n", square.Volume())

fmt.Printf("\nRectangle\nArea: %.2f\n", rectangle.Area())

fmt.Printf("Volume: %.2f\n", rectangle.Volume())

}

**SLIP 10:**

**QUE1- A=>**

package main

import "fmt"

func main() {

// create an empty interface

var a interface {}

// store integer to an empty interface

a = 10

// type assertion

interfaceValue := a.(int)

fmt.Println(interfaceValue)

}

QUE1 -B=>

package main

import "fmt"

func fibonacci(ch chan int, quit chan bool) {

x, y := 0, 1

for {

select {

case ch <- x: // write to channel ch

x, y = y, x+y

case <-quit:

fmt.Println("quit")

return

}

}

}

func main() {

ch := make(chan int)

quit := make(chan bool)

n := 5

go func(n int) {

for i := 0; i < n; i++ {

fmt.Println(<-ch) // read from channel ch

}

quit <- false

}(n)

fibonacci(ch, quit)

}

**SLIP 11:**

**QUE1- A=>**

package main

import "fmt"

func main() {

var num int

fmt.Println("Enter a number:")

fmt.Scanln(&num)

if num >= 10 && num <= 99 {

fmt.Println("The entered number is a two-digit number.")

} else {

fmt.Println("The entered number is not a two-digit number.")

}

}

**QUE1- B=>**

package main

import "fmt"

func main() {

// Create a buffered channel with a capacity of 3

ch := make(chan int, 3)

// Store values in the channel

ch <- 10

ch <- 20

ch <- 30

// Find the channel capacity and length

capacity := cap(ch)

length := len(ch)

fmt.Println("Initial Channel Capacity:", capacity)

fmt.Println("Initial Channel Length:", length)

// Read values from the channel

for i := 0; i < length; i++ {

value := <-ch

fmt.Println("Read value from channel:", value)

}

// Find the modified length of the channel

length = len(ch)

fmt.Println("Modified Channel Length:", length)

}

SLIP 12:

QUE1- A=>

package main

import "fmt"

func swap(x \*int, y \*int) {

temp := \*x

\*x = \*y

\*y = temp

}

func main() {

num1 := 10

num2 := 20

fmt.Println("Before swapping:")

fmt.Println("num1:", num1)

fmt.Println("num2:", num2)

swap(&num1, &num2)

fmt.Println("\nAfter swapping:")

fmt.Println("num1:", num1)

fmt.Println("num2:", num2)

}

QUE1- B=>

package main

import "fmt"

func main() {

var intSlice = []int{91, 42, 23, 14, 15, 76, 87, 28, 19, 95}

chOdd := make(chan int)

chEven := make(chan int)

go odd(chOdd)

go even(chEven)

for \_, value := range intSlice {

if value%2 != 0 {

chOdd <- value

} else {

chEven <- value

}

}

}

func odd(ch <-chan int) {

for v := range ch {

fmt.Println("ODD :", v)

}

}

func even(ch <-chan int) {

for v := range ch {

fmt.Println("EVEN:", v)

}

}

SLIP 13:

QUE1- A=>

package main

import "fmt"

func main() {

evenSum, oddSum := 0, 0

// Calculate sum of even and odd numbers separately

for i := 1; i <= 100; i++ {

if i%2 == 0 {

evenSum += i

} else {

oddSum += i

}

}

// Print sum of even and odd numbers

fmt.Println("Sum of even numbers between 1 to 100:", evenSum)

fmt.Println("Sum of odd numbers between 1 to 100:", oddSum)

}

QUE1 -B=>

package main

import (

"fmt"

"testing"

)

// Square returns the square of a given number

func Square(x int) int {

return x \* x

}

func BenchmarkSquare(b \*testing.B) {

// Benchmark the Square function

for i := 0; i < b.N; i++ {

Square(10) // Change the argument as needed

}

}

func main() {

// Test the Square function

num := 5

fmt.Printf("Square of %d is: %d\n", num, Square(num))

}

SLIP 14:

QUE1- A=>

package main

import "fmt"

func main() {

// Creating a slice

slice1 := []int{1, 2, 3, 4, 5}

// Appending elements to the slice

slice1 = append(slice1, 6, 7, 8)

fmt.Println("Slice after appending:", slice1)

// Removing an element from the slice

indexToRemove := 2

slice1 = append(slice1[:indexToRemove], slice1[indexToRemove+1:]...)

fmt.Println("Slice after removing element at index", indexToRemove, ":", slice1)

// Copying a slice

slice2 := make([]int, len(slice1))

copy(slice2, slice1)

fmt.Println("Copied slice:", slice2)

}

QUE1- B=>

package main

import "fmt"

func calcSquares(number int, squareop chan int) {

sum := 0

for number != 0 {

digit := number % 10

sum += digit \* digit

number /= 10

}

squareop <- sum

}

func calcCubes(number int, cubeop chan int) {

sum := 0

for number != 0 {

digit := number % 10

sum += digit \* digit \* digit

number /= 10

}

cubeop <- sum

}

func main() {

number := 589

sqrch := make(chan int)

cubech := make(chan int)

go calcSquares(number, sqrch)

go calcCubes(number, cubech)

squares, cubes := <-sqrch, <-cubech

fmt.Println("Final output", squares + cubes)

}

QUE15:

QUE1- A=>

package main

import "fmt"

func swap(x, y int) (int, int) {

return y, x

}

func main() {

a, b := swap(10, 20)

fmt.Println("Swapped values:", a, b)

}

QUE1- B=>

package main

import (

"fmt"

"os" )

func main() {

// Open file for reading only

file, err := os.Open("example.txt")

if err != nil {

fmt.Println("Error:", err)

return

}

defer file.Close()

// Read content from the file

data := make([]byte, 100) // Adjust the buffer size according to your requirement

count, err := file.Read(data)

if err != nil {

fmt.Println("Error:", err)

return

}

fmt.Printf("Read %d bytes: %s\n", count, data[:count])

}

Save this code in a file, for example, read\_only\_file.go. Then, compile and run it using the following commands in your terminal:

go build read\_only\_file.go

./read\_only\_file

This program will open the file named example.txt in the current directory (where you run the program) in read-only mode. It then reads the content of the file and prints it to the console. If the file doesn't exist or there's an error while reading, it will print the error message.

SLIP 16:

QUE1 -A=>

create a directory structure for our package and main program:

project/

│

├── rectangle/

│ └── rectangle.go

│

└── main.go

// rectangle.go

package rectangle

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

func Area(length, width float64) float64 {

return length \* width

}

here's main.go, where we'll import our package and use it:

// main.go

package main

import (

"fmt"

"project/rectangle"

)

func main() {

length := 5.0

width := 3.0

area := rectangle.Area(length, width)

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

}

Now, compile and run the main.go file: Now, compile and run the main.go file:

go run main.go

QUE1- B=>

package main

import (

"fmt"

"time"

"math/rand"

)

func f(n int) {

for i := 0; i < 10; i++ {

fmt.Println(n, ":", i)

amt := time.Duration(rand.Intn(250))

time.Sleep(time.Millisecond \* amt)

}

}

func main() {

for i := 0; i < 10; i++ {

go f(i)

}

var input string

fmt.Scanln(&input) }

SLIP 17:

QUE1- A=>

package main

import "fmt"

// add returns the sum of two numbers

func add(a, b float64) float64 {

return a + b

}

// subtract returns the difference between two numbers

func subtract(a, b float64) float64 {

return a - b

}

// multiply returns the product of two numbers

func multiply(a, b float64) float64 {

return a \* b

}

// divide returns the result of dividing two numbers

// It also returns a boolean indicating whether division was successful or not (to avoid division by zero)

func divide(a, b float64) (float64, bool) {

if b == 0 {

return 0, false // Division by zero

}

return a / b, true

}

func main() {

num1, num2 := 10.5, 3.5

// Add

sum := add(num1, num2)

fmt.Printf("Sum: %.2f\n", sum)

// Subtract

difference := subtract(num1, num2)

fmt.Printf("Difference: %.2f\n", difference)

// Multiply

product := multiply(num1, num2)

fmt.Printf("Product: %.2f\n", product)

quotient, success := divide(num1, num2)

if success {

fmt.Printf("Quotient: %.2f\n", quotient)

} else {

fmt.Println("Division by zero error")

}

}

QUE1- B=>

package main

import (

"fmt"

"os"

)

func main() {

// Open file for appending. If the file doesn't exist, create it.

file, err := os.OpenFile("example.txt", os.O\_APPEND|os.O\_WRONLY|os.O\_CREATE, 0644)

if err != nil {

fmt.Println("Error:", err)

return

}

defer file.Close()

// Content to append

content := "\nThis content is appended to the end of the file."

// Write content to the file

\_, err = file.WriteString(content)

if err != nil {

fmt.Println("Error:", err)

return

}

fmt.Println("Content appended to the text file successfully.")

}

Save this code in a file, for example, append\_text\_file.go. Then, compile and run it using the following commands in your terminal:

go build append\_text\_file.go

./append\_text\_file

This program will append the specified content to the end of the file named example.txt in the current directory (where you run the program). If the file doesn't exist, it will be created.

SLIP 18:

QUE1- A=>

package main

import "fmt"

func printMultiplicationTable(number int) {

fmt.Printf("Multiplication table of %d:\n", number)

for i := 1; i <= 10; i++ {

fmt.Printf("%d x %d = %d\n", number, i, number\*i)

}

}

func main() {

number := 5 // Number for which the multiplication table is to be printed

printMultiplicationTable(number)

}

QUE 1- B=>

Create a directory structure for our package and main program:

calculator/

│ calculator.go

│

└── main.go

// calculator.go

package calculator

import "errors"

// Add returns the sum of two numbers

func Add(a, b float64) float64 {

return a + b

}

// Subtract returns the difference between two numbers

func Subtract(a, b float64) float64 {

return a - b

}

// Multiply returns the product of two numbers

func Multiply(a, b float64) float64 {

return a \* b

}

// Divide returns the division of two numbers. Returns an error if b is zero.

func Divide(a, b float64) (float64, error) {

if b == 0 {

return 0, errors.New("cannot divide by zero")

}

return a / b, nil

}

// main.go

package main

import (

"fmt"

"calculator"

)

func main() {

var a, b float64

var choice int

fmt.Println("Enter two numbers:")

fmt.Scan(&a, &b)

fmt.Println("Choose operation:")

fmt.Println("1. Addition")

fmt.Println("2. Subtraction")

fmt.Println("3. Multiplication")

fmt.Println("4. Division")

fmt.Scan(&choice)

var result float64

var err error

switch choice {

case 1:

result = calculator.Add(a, b)

case 2:

result = calculator.Subtract(a, b)

case 3:

result = calculator.Multiply(a, b)

case 4:

result, err = calculator.Divide(a, b)

if err != nil {

fmt.Println("Error:", err)

return

}

default:

fmt.Println("Invalid choice")

return

}

fmt.Println("Result:", result)

}

Now, compile and run the main.go file:=> go run main.go

This will prompt you to enter two numbers and choose an operation. Based on your choice, it will perform the operation using the functions defined in the calculator package and print the result.

SLIP 19:

QUE1- A=>

package main

import "fmt"

// addAndSubtract returns the sum and difference of two numbers

func addAndSubtract(a, b float64) (float64, float64) {

sum := a + b

difference := a - b

return sum, difference

}

func main() {

num1, num2 := 10.5, 3.5

sum, difference := addAndSubtract(num1, num2)

fmt.Printf("Sum: %.2f\n", sum)

fmt.Printf("Difference: %.2f\n", difference)

}

QUE1- B=>

package main

import (

"fmt"

"os"

)

func main() {

// Open file for reading only

file, err := os.Open("example.txt")

if err != nil {

fmt.Println("Error:", err)

return

}

defer file.Close()

// Read content from the file

data := make([]byte, 100) // Adjust the buffer size according to your requirement

count, err := file.Read(data)

if err != nil {

fmt.Println("Error:", err)

return

}

fmt.Printf("Read %d bytes: %s\n", count, data[:count])

}

Save this code in a file, for example, read\_only\_file.go. Then, compile and run it using the following commands in your terminal:

go build read\_only\_file.go

./read\_only\_file

This program will open the file named example.txt in the current directory (where you run the program) in read-only mode. It then reads the content of the file and prints it to the console. If the file doesn't exist or there's an error while reading, it will print the error message.

SLIP 20:

QUE1- A=>

package main

import (

"fmt"

"os"

)

func main() {

// Open file for appending. If the file doesn't exist, create it.

file, err := os.OpenFile("example.txt", os.O\_APPEND|os.O\_WRONLY|os.O\_CREATE, 0644)

if err != nil {

fmt.Println("Error:", err)

return

}

defer file.Close()

// Content to append

content := "\nThis content is appended to the end of the file."

// Write content to the file

\_, err = file.WriteString(content)

if err != nil {

fmt.Println("Error:", err)

return

}

fmt.Println("Content appended to the text file successfully.")

Save this code in a file, for example, append\_text\_file.go. Then, compile and run it using the following commands in your terminal: => go build append\_text\_file.go

./append\_text\_file

This program will append the specified content to the end of the file named example.txt in the current directory (where you run the program). If the file doesn't exist, it will be created.

QUE1- B=>

package main

import "fmt"

func main() {

// Create a channel of integers

ch := make(chan int)

// Producer: Send integers to the channel

go func() {

for i := 1; i <= 5; i++ {

ch <- i // Send integers 1 to 5 to the channel

}

close(ch) // Close the channel after sending all values

}()

// Consumer: Receive integers from the channel

for num := range ch {

fmt.Println("Received:", num)

}

// Check if the channel is closed

if \_, ok := <-ch; !ok {

fmt.Println("Channel is closed")

}

}