Assignment 4

SETA

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

package main import "fmt"

//Interface declaration type shape interface { area() float64 perimeter() float64

}

//Struct declaration for rectangle type rectangle struct{ length, height float64

}

//Struct declaration for circle type circle struct{ radius float64

}

//Method declarations for rectangle func (r rectangle) area() float64 { return r.length \* r.height

}

func (r rectangle) perimeter() float64 { return 2 \* r.length + 2 \* r.height

}

//Method declarations for circle func (c circle) area() float64 { return 3.142 \* c.radius \* c.radius }

func (c circle) perimeter() float64 { return 2 \* 3.142 \* c.radius

}

func main() { r := rectangle{length: 10.0, height: 5.0} c := circle{radius: 5.0}

fmt.Println("Area of rectangle is ", r.area()) fmt.Println("Parameter of rectangle is ", r.perimeter()) fmt.Println("Area of circle is ", c.area())

fmt.Println("Perimeter of circle is ", c.perimeter())

}

2. Write a program in go language to print multiplication of two numbers using method package main import "fmt"

type data int

func (d1 data) multipy (d2 data) data { mult := d1 \* d2 return mult

}

func main() { value1 := data(23) value2 := data(20)

res := value1.multipy(value2) fmt.Println("Final Result : " ,res)

}

3. Write a program in go language to create structure author.

Write a method show() whose receiver is struct author.

package main import "fmt"

type author struct{

branch,name string particles,salary int

}

func (a author) show(){ fmt.Println("Author's Name : ", a.name) fmt.Println("Branch Name : ", a.branch) fmt.Println("Published Articles : ", a.particles) fmt.Println("Salary : ", a.salary)

}

func main(){

res := author{ name : "Tejas", branch : "BCA", particles : 03, salary: 734000,

}

res.show()

}

SETB

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

package main import "fmt" type student struct{

name string roll int

}

func (s \*student) show (sroll int,sname string){

(\*s).roll = sroll

(\*s).name = sname

}

func main() {

res := student{ name : "Ashish", roll: 1,

}

fmt.Println("Roll No : ", res.roll) fmt.Println("Student Name : ",res.name)

p := &res

p.show(2,"Tejas")

fmt.Println("Roll No : ", res.roll) fmt.Println("Student Name : ",res.name)

}

2. Write a program in go language to demonstrate working type switch in interface.

package main import ("fmt") func main() { var value interface{} = "2" switch t := value.(type){

case int64:

fmt.Println("Type is an integer:", t) case float64:

fmt.Println("Type is a float:", t) case string:

fmt.Println("Type is a string:", t)

case nil:

fmt.Println("Type is nil.") case bool:

fmt.Println("Type is a bool:", t)

default:

fmt.Println("Type is unknown!")

}

}

SETC

1. Write a program in go language to create an interface and display it’s values with the help of type assertion.

package main import "fmt" func main(){ var myInt interface{} = 123 k, ok := myInt.(int) if ok { fmt.Println("Success :", k) }

v, ok := myInt.(float64) if ok { fmt.Println(v) }else{

fmt.Println("Failed Without Panicking !")

}

}

2. Write a program in go language to demonstrate working embedded interfaces. package main import "fmt" // Interface 1 type AuthorDetails interface { details()

}

// Interface 2 type AuthorArticles interface { articles()

}

// Interface 3

// Interface 3 embedded with // interface 1 and 2 type FinalDetails interface {

AuthorDetails

AuthorArticles

}

// Structure type author struct {

a\_name string branch string college string year int salary int particles int tarticles int

}

// Implementing method of // the interface 1 func (a author) details() {

fmt.Printf("Author Name: %s", a.a\_name) fmt.Printf("\nBranch: %s and passing year: %d",

a.branch, a.year) fmt.Printf("\nCollege Name: %s", a.college) fmt.Printf("\nSalary: %d", a.salary) fmt.Printf("\nPublished articles: %d", a.particles)

}

// Implementing method of the interface 2 func (a author) articles() {

pendingarticles := a.tarticles - a.particles fmt.Printf("\nPending articles: %d", pendingarticles)

}

// Main value

func main() {

// Assigning values // to the structure values := author{ a\_name: "Shirwaikar", branch: "Computer science", college: "XYZ", year: 1990, salary: 80000, particles: 209, tarticles: 309,

}

// Accessing the methods of

// the interface 1 and 2 // Using FinalDetails interface var f FinalDetails = values

f.details()

f.articles()

}

# Assignment 5

SETA

2. WAP in GO program that executes 5 go routines simultaneously which generates numbers from 0 to 10, waiting between 0 and 250 ms after each go routine.

package main import ( "fmt"

"time"

)

func numbers() {

for i := 1; i <= 5; i++ { time.Sleep(250 \* time.Millisecond) fmt.Printf("%d ", i)

}

}

func alphabets() { for i := '0'; i <= '10'; i++ { time.Sleep(400 \* time.Millisecond) fmt.Printf("%c ", i)

}

}

func main() { go numbers() go alphabets() time.Sleep(3000 \* time.Millisecond) fmt.Println("main terminated")

}

3. Write a go program that creates a slice of integers, checks numbers from slice are even or odd and further sent to respective go routines through channel and display values received by go routines.

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) }

}

SETB

1. WAP in Go to create buffered channel, store few values in it and find channel capacity and length. Read values from channel and find modified length of a channel.

package main import (

"fmt"

)

func main() {

// create a buffered channel // with a capacity of 2. ch := make(chan string, 2) ch <- "geeksforgeeks" ch <- "geeksforgeeks world" fmt.Println(<-ch) fmt.Println(<-ch) fmt.Println("Capacity of Buffered Channel : ",cap(ch)) fmt.Println("Capacity of Buffered Channel : ",len(ch))

}

2. WAP in Go main go routine to read and write Fibonacci series to the channel 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 := 10

go func(n int) { for i := 0; i < n; i++ { fmt.Println(<-ch) // read from channel ch

}

quit <- false

}(n)

fibonacci(ch, quit)

}}

3. WAP in Go how to create channel and illustrate how to close a channel using for range loop and close function.

package main

import (

"fmt"

"sync"

)

var wg sync.WaitGroup = sync.WaitGroup{}

func main() {

ch := make(chan int, 50)

wg.Add(2)

go func(ch <-chan int) {

for {

if i, ok := <-ch; ok {

fmt.Println(i)

} else {

break

}

}

wg.Done()

}(ch)

go func(ch chan<- int) {

var i int i = 17 ch <- i

ch <- i + 18 ch <- 13 ch <- 19 close(ch) wg.Done() }(ch) wg.Wait()

}