**Google Go Programming Language**

*REPORT submitted in partial fulfillment of the requirements*

Submitted By

**Motamarri Jaya Naga Venkata Sai ( 208W1A12A0 )**

Under The Guidance Of

**M . Ramesh ( Assistant Professor )**

**Prakash Kaja ( CEO of Saadruso Company )**

****

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**V R SIDDHARTHA ENGINEERING COLLEGE**

**( AUTONOMOUS - AFFILIATED TO JNTU-K, KAKINADA )**

**Approved by AICTE & Accredited by NBA**

**KANURU, VIJAYAWADA-520007**

**ACADEMIC YEAR**

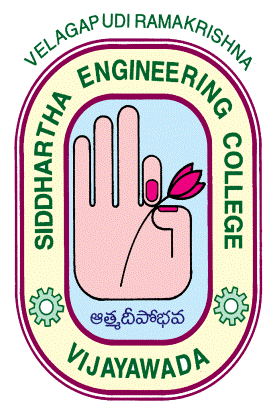
**(2022-23)**

**V.R.SIDDHARTHA ENGINEERING COLLEGE**

(Affiliated to JNTUK: Kakinada, Approved by AICTE, Autonomous)

(An ISO certified and NBA accredited institution)

Kanuru, Vijayawada – 520007



**CERTIFICATE**

This is to certify that this project report titled **“Go language”** is a bonafide record of work done by  **M . J . N. V. Sai ( 208W1A12A0 )** under my guidance and supervision is submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Information Technology, **V.R. Siddhartha Engineering College** (Autonomous under JNTUK) during the year 2022-23.

**( M . Ramesh )** ( **K . Prakash** **)**

Assistant Professor Chief Executive Officer

Dept. of Information Technology Saadruso Company

# **ACKNOWLEDGEMENT**

First and foremost, I sincerely salute our esteemed institution **V.R SIDDHARTHA ENGINEERING COLLEGE** for giving me this opportunity for fulfilling my project. I am grateful to our Principal **Dr. A.V.RATNA PRASAD**, for his encouragement and support all through the way of my project.

On the submission of this Project report, I would like to extend my honour to **Dr. M.Suneetha**, Head of the Department, IT for her constant motivation and support during the course of my work.

I feel glad to express my deep sense of gratefulness to my project guide **M . Ramesh, Assistant Professor**  for **his/her** guidance and assistance in completing this project successfully.

I would also like to convey my sincere indebtedness to all faculty members, including supporting staff of the Department, friends and family members who bestowed their great effort and guidance at appropriate times without which it would have been very difficulty on my part to finish the project work.

**Table of Contents**

[**ACKNOWLEDGEMENT** 3](#_Toc121040918)

[**CHAPTER – 1** 5](#_Toc121040919)

[1.1 Installation And Introduction to variables in the Google Go 5](#_Toc121040920)

[Programming Language 5](#_Toc121040921)

[**CHAPTER – 2** 7](#_Toc121040922)

[**2.1** Understanding the Different Types of DataTypes in the Google Go 7](#_Toc121040923)

[Programming language. 7](#_Toc121040924)

[**CHAPTER – 3** 10](#_Toc121040925)

[**3.1** Apply the Logical, Arithimatic, Control Structures in Google Go 10](#_Toc121040926)

[Programming Language. 10](#_Toc121040927)

[**CHAPTER – 4** 17](#_Toc121040928)

[**4.1**  Write a Basic Program On Arrays and Slices and Perform all 17](#_Toc121040929)

[Operations on The arrays in Go language. 17](#_Toc121040930)

[**CHAPTER – 5** 23](#_Toc121040931)

[**5.1** Apply Various types of functions in Go Language. 23](#_Toc121040932)

[**CHAPTER – 6** 29](#_Toc121040933)

[**6.1** implement the Below List of Programs in Go language. 29](#_Toc121040934)

[**CHAPTER – 7** 38](#_Toc121040935)

[**7.1** Implement maps associative datatype in go language. 38](#_Toc121040936)

[**CHAPTER – 8** 40](#_Toc121040937)

[**8.1** implement structures concepts and it’s methods and interface 40](#_Toc121040938)

[Concepts in go language. 40](#_Toc121040939)

# **CHAPTER – 1**

## Installation And Introduction to variables in the Google Go

## Programming Language

**Procedure :**

1. First, Visit the Website of google called “ <http://go.dev/dl/> “.
2. From there download the Microsoft windows . msi and then run it.
3. After giving the all access permissions for the software create a folder

With a name called “ go “ in the system default drive called C Drive

And then navigate to the Users Directory and then create a folder

A name called “ go “.

1. Now, copy the created folder path and then click the ⊞ + R at the same time and then type “ sysdm.cpl ,3 “ and then click on the environmental variables.
2. Then navigate to the bottom down variables block and click on New button and set the variable name to “ GOPATH “ and paste the copied one in the variable path. Then click on OK or SAVE.
3. Final step is go to the Coomand Prompt and type “ %GOPATH % “ and the created folder will open this results the the path setting in google go environment is success.

**Program :**

package main

import "fmt" // importing the inbuilt functions

func main( ){

fmt.Println("Different types of variable declarations.")

var k int

k = 10

// m = 10 is a wrong format

m := 10

fmt.Println("k and m values are : ", k, " ", m)

var b, c int = 1, 3

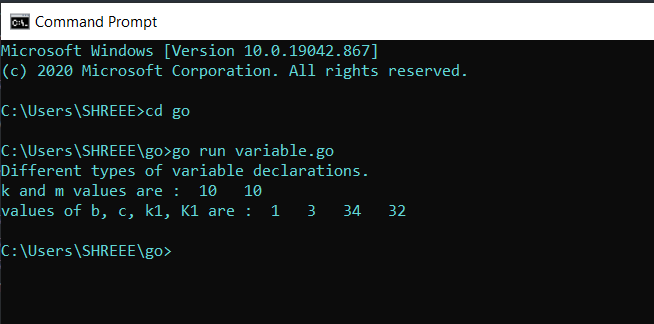
var k1 = 34

var K1 = 32

fmt.Println("values of b, c, k1, K1 are : ", b, " ", c, " ", k1, " ", K1)

}

**Output :**



**Result :** Successfully Executed the Program.

# **CHAPTER – 2**

## **2.1** Understanding the Different Types of DataTypes in the Google Go

## Programming language.

**Description :**

1. Datatypes plays an important and major role in any programming language and they play as base to all programming stratiges.
2. They store only a single type of data at only in a life time and allocated the memory id and type of the data stored in that memory part.
3. There are different types of datatypes in google go programming language and some the basic data types are below one’s :

* Boolean Data Type
* Integer Data Type
* String Data Type
* Float Data Type

**Program :**

package main

import "fmt"

func main(){

// Integer DataTypes

var x int = 500

var x1 = 200

x2 := 4500

fmt.Printf("Type: %T, value: %v\n",x,x) // %T and %v works on only Printf

Function..

fmt.Printf("Type: %T, value: %v\n",x1,x1)

fmt.Printf("Type: %T, value: %v\n",x2,x2)

// Float Datatypes

fmt.Println()

var y float32 = 12.56 // or you can use float64 type also

var y1 = 19.999

y2 := 10.15

fmt.Printf("Type: %T, value: %v\n",y,y)

fmt.Printf("Type: %T, value: %v\n",y1,y1)

fmt.Printf("Type: %T, value: %v\n",y2,y2)

// String Datatypes

fmt.Println()

var z string = "Hello World!"

var z1 = "Welcome To The Go lang World"

z3 := 'A'

z2 := "I am Faster than Python and Efficient Than Java"

fmt.Printf("Type: %T, value: %v\n",z,z)

fmt.Printf("Type: %T, value: %v\n",z1,z1)

fmt.Printf("Type: %T, value: %v\n",z2,z2)

fmt.Printf("Type: %T, value: %v\n",z3,z3)

// Boolean DataTypes

fmt.Println()

var a bool = true

var a1 = false

a2 := true

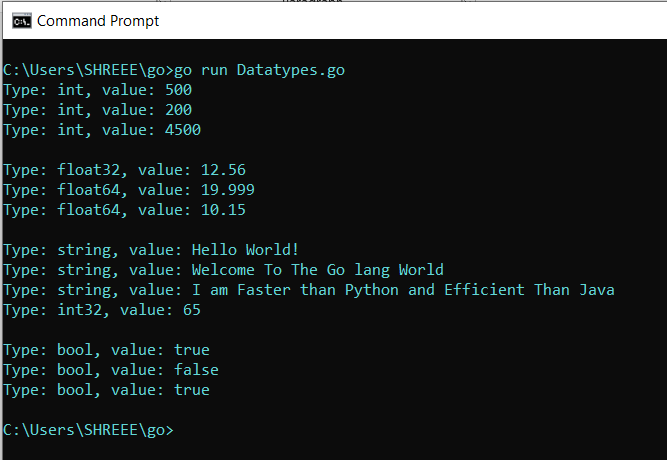
fmt.Printf("Type: %T, value: %v\n",a,a)

fmt.Printf("Type: %T, value: %v\n",a1,a1)

fmt.Printf("Type: %T, value: %v\n",a2,a2)

}

**Output :**



**Result :**  Sucessfully Executed the Program.

# **CHAPTER – 3**

## **3.1** Apply the Logical, Arithimatic, Control Structures in Google Go

## Programming Language.

**Description :**

1. Arithimatic operations are common in all operations they are addition, subtraction, multiplication, Division just we have to apply them on a variable in which they are holding a integer or float data.
2. Logical operators are AND(&&), OR(||), NOT(!) these operator will help us in finding the Boolean values as their result and they also help in building the logic in problem.
3. Control structures are very famous in any programming languages and some of the control structures are :

* If else Condition
* While condition
* For condition

**Program :**

//go build hello-world.go ==> it converts .go code to an encrypted binary code language

//dir

//./hello-world ==> this command used for running the file.go

package main

// By using paranthesis () after import we can import multiple inbuilt functions and packages

import (

"fmt"

"math"

)

const s string = "constant"

func main() {

// Concatination Of Strings

str1 := "go"

str2 := " language"

fmt.Println("String 1 : ", str1)

fmt.Println("String 2 : ", str2)

fmt.Println("Concatination of 2 Strings : ", str1 + str2)

fmt.Println()

// Performing Arthimatic Operations

num1 := 5

num2 := 10

num3 := 5.65

num4 := 8.9

fmt.Println("Number 1 : ", num1)

fmt.Println("Number 2 : ", num2)

fmt.Println("Addition of 2 Numbers are : ", num1 + num2)

fmt.Println("Subraction of 2 Numbers are : ", num1 - num2)

fmt.Println("Multiplication of 2 Numbers are : ", num1 \* num2)

fmt.Println("Division of 2 Float Numbers : ", num3 / num4)

fmt.Println()

// Performing Logical Operations

var b1 bool = true

var b2 bool = false

fmt.Println("Data Stored in Variable b1 is : ", b1)

fmt.Println("Data Stored in Variable b2 is : ", b2)

fmt.Println("Applying AND Opearation with different values : ", b1 && b2)

fmt.Println("Applying AND Opearation with Same Values : ", b1 && b1)

fmt.Println("Applying OR Opearation with different values: ", b1 || b2)

fmt.Println("Applying OR Opearation with Same values: ", b2 || b2)

fmt.Println("Applying NOT(!) Opearation : ", !b1)

fmt.Println()

// Performing Constatant Data Type

fmt.Println("Value Stored in S variable which is a Constatnt DataType : ", s)

//const n = 50000000 // explicit conversion.

const n = 50

const d = 3e20 / n

fmt.Println("the value of 3e20 is : ", 3e20)

fmt.Println("Converting the Biggest number : ", int64(d))

fmt.Println(" value of sin 50 is : ", math.Sin(n))

fmt.Println()

//Performing the Conditional and Control Structures

i := 1

fmt.Println("for loop with Single Condition printing 1 to 3 Numbers :")

for i <= 3 {

fmt.Println(i)

i = i + 1

}

fmt.Println("for loop with 3 conditions printing 7 to 9 Numbers : ")

for j := 7; j <= 9; j++ {

fmt.Println(j)

}

fmt.Println("for loop with no conditions : ")

for {

fmt.Println("loop")

break

}

fmt.Println("Printing the Odd numbers using For loop : ")

for n := 0; n <= 5; n++ {

if n%2 == 0 {

continue

}

fmt.Println(n)

}

// Checking the even odd Conditions

if 7%2 == 0 {

fmt.Println("7 is even")

} else {

fmt.Println("7 is odd")

}

// Checking the divisibility Conditions

if 8%4 == 0 {

fmt.Println("8 is divisible by 4")

}

// checking the Positive, Negative, Single digits

if num := 9; num < 0 {

fmt.Println(num, "is negative")

} else if num < 10 {

fmt.Println(num, "has 1 digit")

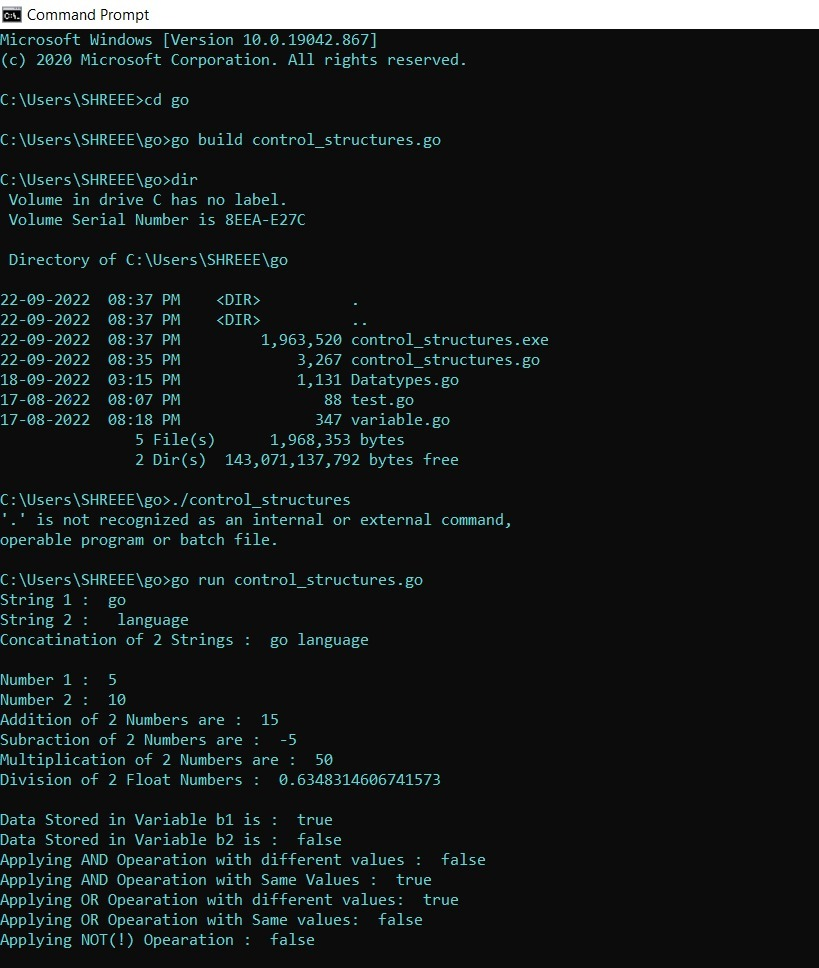
} else {

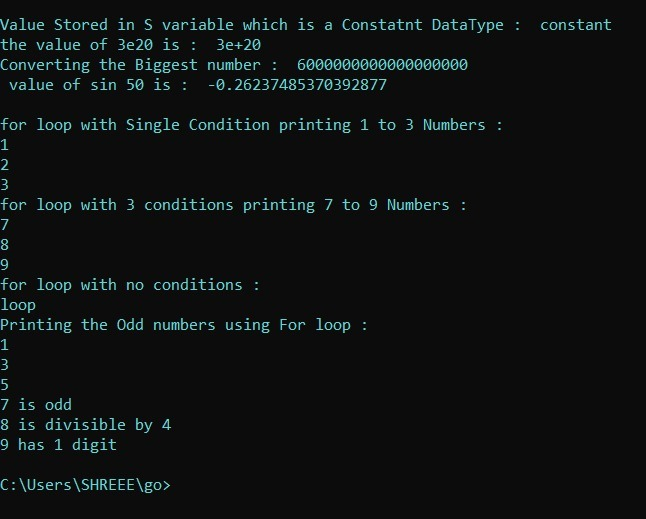
fmt.Println(num, "has multiple digits")

}

}

**Output :**





**Result :** Sucessfully Executed the Program.

# **CHAPTER – 4**

## **4.1** Write a Basic Program On Arrays and Slices and Perform all

## Operations on The arrays in Go language.

**Description :**

1. Arrays plays an important role in manipulating the and handling the huge amount of a data of an same type .
2. They store a single type of an data in a array which can have an explicated declared the array length or implicated declared array length.
3. [. . .] 🡺 this resembles the infinite length in that array.
4. Ex : var1 := [3]int{ 1,2,3 }

Var2 := [ . . . ]int{ 1,2,3,4,5,6,7,8}

1. The main disadvantage in the arrays is they can hold only same datatype elements in the array.

**Program - 1 :**

package main

import ("fmt")

func main(){

var arr1 = [3]int{1,2,3} // 3 indicates the length of an array

arr2 := [5]int{4,5,6,7,8}

arr3 := [...]string{"Volvo", "BMW", "Mercendes", "Aadhipurush", "SkyScrapper"} // [...] => indicates unlimited length in the array

fmt.Printf("Type of an array : %T, \n Values in the array are : %v", arr1, arr1)

fmt.Printf("Type of an array : %T, \n Values in the array are : %v", arr3, arr3)

// Performing the Slicing Operations

fmt.Println("\nElement present at the position 3 is in arr2 variable : ", arr2[3])

// Manipulating the elements in the array

fmt.Println("List of all Elements in arr2 are : ", arr2)

fmt.Println("Element present in position 2 is : ", arr2[2])

arr2[2] = 500

fmt.Println("Now Element present in position 2 is : ", arr2[2])

// Special Type in Array Declaration

array := [...]int{50:23, 12:100} // i.e; element 23 is placed at 50th position in array

// element 100 is placed at 12th position in the array

fmt.Printf("Type of an array : %T, \n Values in the array are : %v", array, array)

fmt.Println("\nLength of an array is : ", len(array))

fmt.Println("\n Capacity of an array is : ", cap(array))

// 2 dimensional Array

TwoD := [3][3]int{ {1,2,3}, {4,5,6}, {7,8,9} }

fmt.Printf("\n Type of an 2D array is : %T ", TwoD)

fmt.Println("\n Elements in 2D array are : ", TwoD)

}

**Program – 2:**

// \_Slices\_ are an important data type in Go, giving

// a more powerful interface to sequences than arrays.

package main

import "fmt"

func main() {

// Unlike arrays, slices are typed only by the

// elements they contain (not the number of elements).

// To create an empty slice with non-zero length, use

// the builtin `make`. Here we make a slice of

// `string`s of length `3` (initially zero-valued).

s := make([]string, 3)

fmt.Println("emp:", s)

// We can set and get just like with arrays.

s[0] = "a"

s[1] = "b"

s[2] = "c"

fmt.Println("set:", s)

fmt.Println("get:", s[2])

// `len` returns the length of the slice as expected.

fmt.Println("len:", len(s))

// In addition to these basic operations, slices

// support several more that make them richer than

// arrays. One is the builtin `append`, which

// returns a slice containing one or more new values.

// Note that we need to accept a return value from

// `append` as we may get a new slice value.

s = append(s, "d")

s = append(s, "e", "f")

fmt.Println("apd:", s)

// Slices can also be `copy`'d. Here we create an

// empty slice `c` of the same length as `s` and copy

// into `c` from `s`.

c := make([]string, len(s))

copy(c, s)

fmt.Println("cpy:", c)

// Slices support a "slice" operator with the syntax

// `slice[low:high]`. For example, this gets a slice

// of the elements `s[2]`, `s[3]`, and `s[4]`.

l := s[2:5]

fmt.Println("sl1:", l)

// This slices up to (but excluding) `s[5]`.

l = s[:5]

fmt.Println("sl2:", l)

// And this slices up from (and including) `s[2]`.

l = s[2:]

fmt.Println("sl3:", l)

// We can declare and initialize a variable for slice

// in a single line as well.

t := []string{"g", "h", "i"}

fmt.Println("dcl:", t)

// Slices can be composed into multi-dimensional data

// structures. The length of the inner slices can

// vary, unlike with multi-dimensional arrays.

twoD := make([][]int, 3)

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

innerLen := i + 1

twoD[i] = make([]int, innerLen)

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

twoD[i][j] = i + j

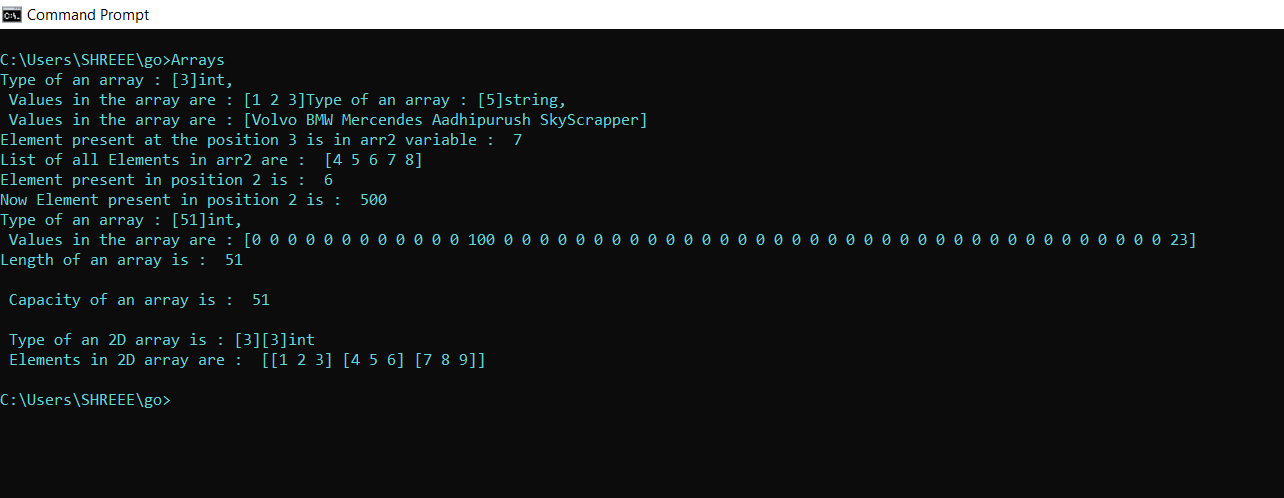
}

}

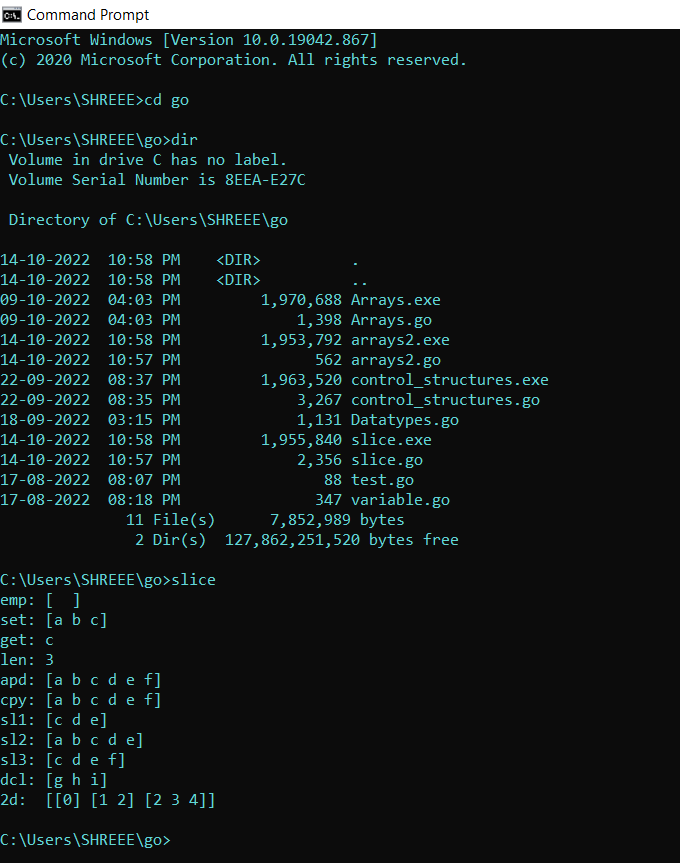
fmt.Println("2d: ", twoD)

}

**Output - 1:**



**Output – 2:**



**Result :** Sucessfully Executed the program.

# **CHAPTER – 5**

## **5.1** Apply Various types of functions in Go Language.

**Description :**

1. Functions play an important role in a developer life or any application.
2. They help in reducing the code.
3. Once we write a function we can reuse that function as many times we want in the application or program.
4. Functions will have the return type and no.of parameters should pass to it.
5. In go language the functions can return multiple values at time .
6. **Syntax :**

func name(variable datatype, variable datatype) function\_return\_type{

Body of the Function

... => means Ellipsis operator

}

**Program :**

package main

import "fmt"

import "strings"

func main(){

x := 10

y := 20

z := simple\_add(x,y)

fmt.Println(" Addition : ", z)

u1, u2, u3 := multiple\_return()

fmt.Println(" Book Number : ", u1)

fmt.Println(" Book Name : ", u2)

fmt.Println(" Book Cost : ", u3)

// Anonymous Function

Anon := func(a, b, c int) int{

fmt.Println()

fmt.Println(" This Is an Anonymous Function")

return a+b+c

}

fmt.Println("Arithmetic : ", Anon(1,2,3))

cs := vardiac\_con("VRSEC", "It", "Cse", "Ece", "Civil", "Mech")

fmt.Println("Concatination Of Strings : ", cs)

fact := recursive\_fact(10)

fmt.Println("Factorial of 10 is : ", fact)

defer defer\_end()

r1 := special(10, inc)

r2 := special(100, dec)

fmt.Println("r1 value : ", r1);

fmt.Println("r2 value : ", r2);

}

func simple\_add(a int, b int) int{

fmt.Println(" This Is a Simple Function ")

ans := a + b

return ans

}

func multiple\_return() (int, string, float64){

fmt.Println()

fmt.Println("This function Returns Multiple Values at a Same Time")

book\_no := 38

book\_name := "Simulated Reality"

book\_cost := 480.95

return book\_no, book\_name, book\_cost

}

func vardiac\_con(elements ...string) string{

fmt.Println()

fmt.Println(" This is an vardiac Function ")

concat := strings.Join(elements, " $ ")

return concat

}

func recursive\_fact(a int) int{

fmt.Println()

fmt.Println(" This is an Recursive Function")

if a == 0 || a == 1{

return 1

} else{

return a\*recursive\_fact(a-1)

}

}

func defer\_end() {

fmt.Println()

fmt.Println(" This is a Defer Function Call ")

fmt.Println("After The Main Method The Defer Statement Will Execute")

}

func inc(x int) int{

x++

return x

}

func dec(x int) int{

x--

return x

}

func special(x int, f func(int) int) int{

fmt.Println()

fmt.Println(" This is a Function as a parameter to Another function ")

r := f(x)

return r

}

/\*

func name(variable datatype, variable datatype) function\_return\_type{

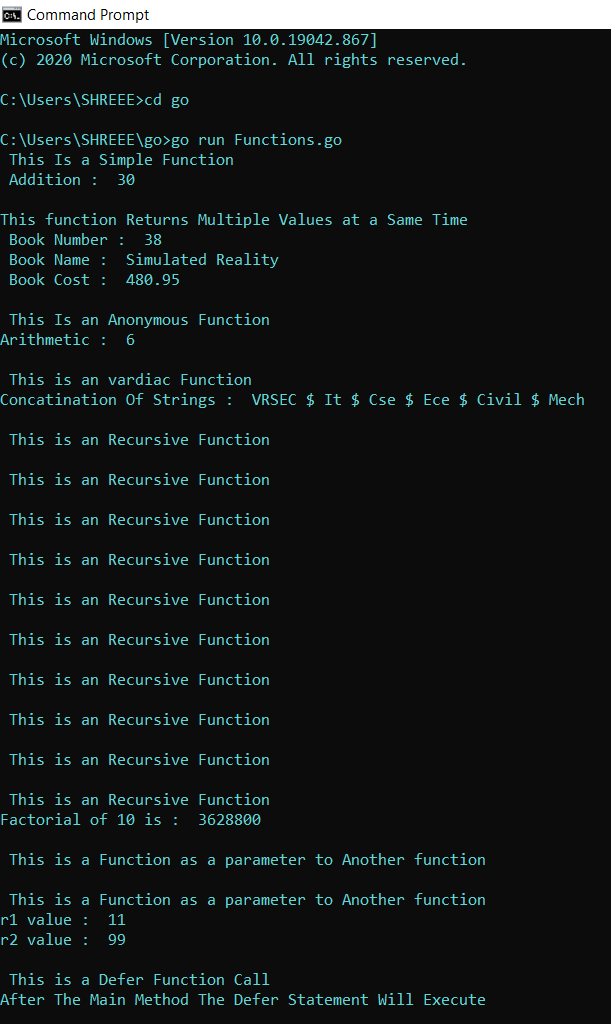
Body of the Function

... => means Ellipsis operator

}

\*/

**Output :**



**Result :** Sucessfully Executed the program.

# **CHAPTER – 6**

## **6.1** implement the Below List of Programs in Go language.

1. Write a go program weather the given string is palindrome or not.
2. Write a go program to display the given numbers in a Ascending order.
3. Write a Go program by using switch keyword to perform a arithmatic operations.
4. Write a Go program weather the given is Armstrong or not.
5. Write a go program by using a function keyword.

**Program – 1:**

package main

import "fmt"

func main(){

var ustr string

fmt.Print("Enter ant String as an Input : ")

fmt.Scanln(&ustr)

reversestr := ""

for i := len(ustr)-1; i >= 0; i--{

reversestr += string(ustr[i])

}

for i := range(ustr){

if ustr[i] != reversestr[i]{

fmt.Println(" Given String is not an palindrome string ")

break

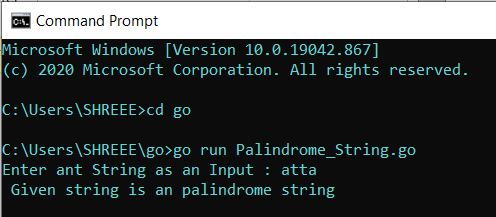
}

}

fmt.Println(" Given string is an palindrome string ")

}

**Output :**



**Program – 2:**

package main

import "fmt"

import "sort"

func main(){

array := [6]int{}

fmt.Println(" Enter the elements into the array ")

for i := 0; i < 6; i++{

fmt.Printf("Enter %vth element: ", i)

fmt.Scanln(&array[i])

}

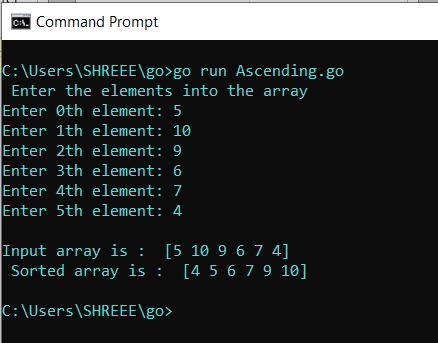
fmt.Println("\nInput array is : ", array)

sort.Ints(array[:])

fmt.Println(" Sorted array is : ", array)

}

**Output :**



**Program – 3:**

package main

import "fmt"

import "os"

func main(){

var a,b, choice int64

fmt.Print("Enter the 1st Number : ")

fmt.Scanln(&a)

fmt.Print("Enter the 2nd Number : ")

fmt.Scanln(&b)

for true{

fmt.Println(" \nEnter 1 for Addition ")

fmt.Println(" Enter 2 for Subraction ")

fmt.Println(" Enter 3 for Multiplication")

fmt.Println(" Enter 4 for Division")

fmt.Println(" Enter 5 for to exit the program")

fmt.Print("\n Enter your Choice : ")

fmt.Scanln(&choice)

switch choice{

case 1:

fmt.Println(" Addition of Given Numbers are : ", a+b)

case 2:

fmt.Println(" Subraction of Given Numbers are : ", a-b)

case 3:

fmt.Println(" Multiplication of Given Numbers are : ", a\*b)

case 4:

fmt.Println(" Division of Given Numbers are : ", a/b)

case 5:

os.Exit(0)

default:

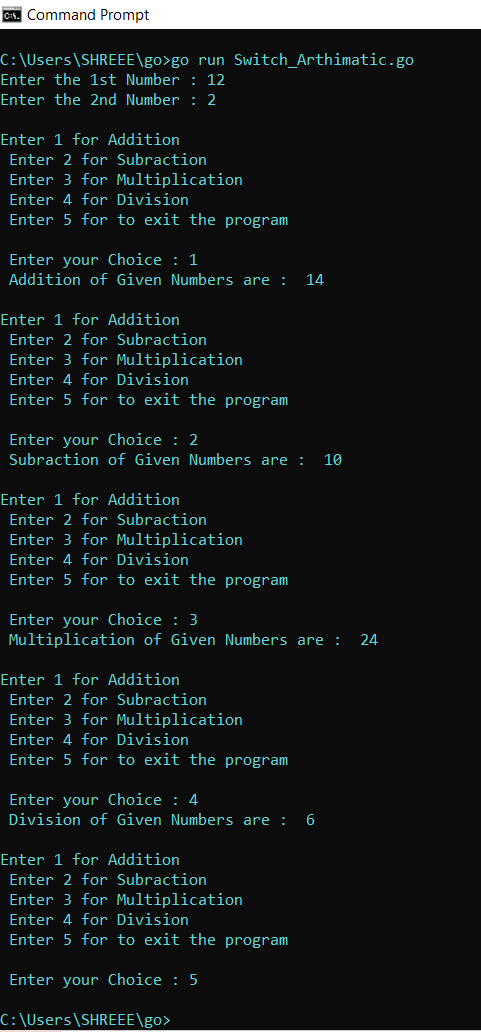
fmt.Println(" Invalid Input ")

}

}

}

**Output :**



**Program – 4:**

package main

import "fmt"

func main(){

var arm int

fmt.Print(" Enter Any Number : ")

fmt.Scanln(&arm)

temp := arm

sum := 0

for arm > 0{

rem := arm%10

sum += (rem\*rem\*rem)

arm = arm/10

}

if sum == temp{

fmt.Println(" Given Number is an Armstrong Number ")

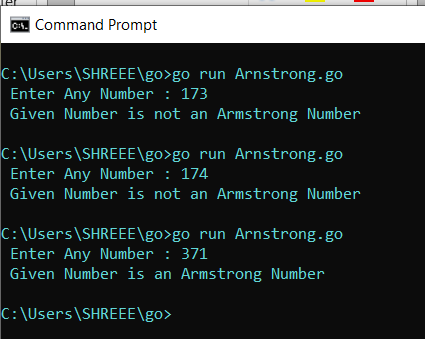
}else{

fmt.Println(" Given Number is not an Armstrong Number ")

}

}

**Output :**



**Program – 5:**

package main

import "fmt"

func Simple(x int, y int) int{

fmt.Println(" This is an Simple Function")

return x+y

}

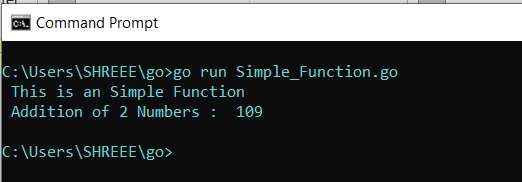
func main(){

result := Simple(10,99)

fmt.Println(" Addition of 2 Numbers : ", result)

}

**Output :**



**Result :** Successfully executed all programs.

# **CHAPTER – 7**

## **7.1** Implement maps associative datatype in go language.

**Description :**

1. In go language map datatype is a built in datatype and similar to the dictionary in python.
2. It consists of key – value pair and each pair is an item in map
3. All keys must be in same datatype and All values must be in same datatype in a map.
4. To create an empty map, use the builtin  function in go

make(map[key-type]val-type).

1. Set key/value pairs using typical  name[key] = val  syntax.
2. Get a value for a key with  name[key] .

**Program :**

package main

import "fmt"

func main() {

m := make(map[string]int) //creating a map with key value pair

// Adding Key value Pairs to Map

m["Age"] = 20

m["Roll"] = 100

fmt.Println("Initial Map :", m)

v1 := m["Age"]

fmt.Println("Value of Age in Map : ", v1)

fmt.Println("len of an Map :", len(m))

delete(m, "Roll")

fmt.Println("Deleting Roll key value pair in map :", m)

\_, prs := m["Roll"]

fmt.Println("Roll Value :", prs)

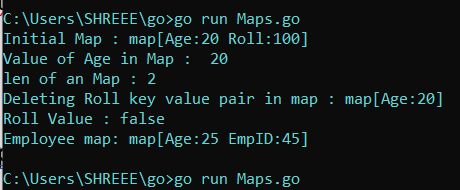
// Another way of creating a map

n := map[string]int{"EmpID": 45, "Age": 25}

fmt.Println("Employee map:", n)

}

**Output :**



**Result :** Sucessfully Executed the Program .

# **CHAPTER – 8**

## **8.1** implement structures concepts and it’s methods and interface

## Concepts in go language.

**Description :**

1. Structures in go are collection of fields and they help in grouping the datatypes together.
2. When ever you use the structure object in a function you return safely a pointer to the variable .

**Syntax :**

type structure – name struct {

variable datatype

variable datatype

}

// accessing the variable in a structure

Struct – object := structure – name { variable : value, variable : value }

Struct – object.variable = value

1. It can also handles the functions in go language.
2. We can use the structure variable inside the functions by using the below syntax it takes structure object as a parameter.

**Syntax :**

func (struct-object \*struct-name) function-name( ) return type {

// Body of the function

}

1. Interfaces are collection of method signatures together.
2. They are like a run time polymorphism in go language.
3. We can define the method signature in a interface and we can implement different logics under a same method signature.

**Syntax :**

type interface – name interface {

method – name returnType

method – name returnType

}

**Program – 1:**

package main

import "fmt"

type person struct {

Name string

Age int

}

func newPerson(name string) \*person {

p := person{Name: name}

p.Age = 42

return &p

}

func main() {

fmt.Println("directly give the values to the struct : ", person{"Bob", 20})

fmt.Println("giving the values by mapping the variable names in the struct : ", person{Name: "Alice", Age: 30})

fmt.Println("giving only one value to the struct : ", person{Name: "Fred"})

fmt.Println("Another way of giving values to the struct : ", &person{Name: "Ann", Age: 40})

fmt.Println("New Person Function : ", newPerson("Jon"))

s := person{Name: "Sean", Age: 50}

fmt.Println(" Initial Structure : ", s)

fmt.Println("Getting the Name value in struct using object : ", s.Name)

sp := &s

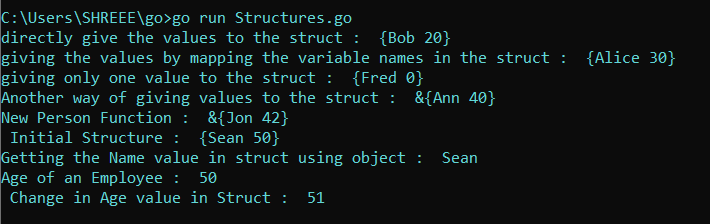
fmt.Println("Age of an Employee : " , sp.Age)

sp.Age = 51

fmt.Println(" Change in Age value in Struct : ", sp.Age)

}

**Output :**



**Program – 2:**

package main

import "fmt"

type rect struct {

width, height int

}

func (r \*rect) area() int {

return r.width \* r.height

}

func (r rect) perim() int {

return 2\*r.width + 2\*r.height

}

func main() {

r := rect{width: 10, height: 5}

fmt.Println("area of a Rectangle : ", r.area())

fmt.Println("perimeter of a Rectangle :", r.perim())

rp := &r

rp.width = 20

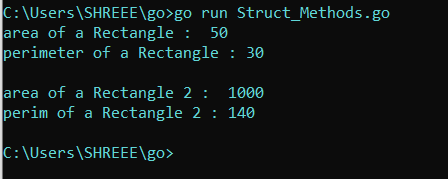
rp.height = 50

fmt.Println("\narea of a Rectangle 2 : ", rp.area())

fmt.Println("perim of a Rectangle 2 :", rp.perim())

}

**Output :**



**Program – 3 :**

package main

import (

"fmt"

"math"

)

type geometry interface {

area() float64

perim() float64

}

type rect struct {

width, height float64

}

type circle struct {

radius float64

}

func (r rect) area() float64 {

return r.width \* r.height

}

func (r rect) perim() float64 {

return 2\*r.width + 2\*r.height

}

func (c circle) area() float64 {

return math.Pi \* c.radius \* c.radius

}

func (c circle) perim() float64 {

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

}

func measure(g geometry) {

fmt.Println("Interface Object : ", g)

fmt.Println("Area of the Shape : ", g.area())

fmt.Println("Perimeter of the Shape : ", g.perim())

fmt.Println()

}

func main() {

r := rect{width: 3, height: 4}

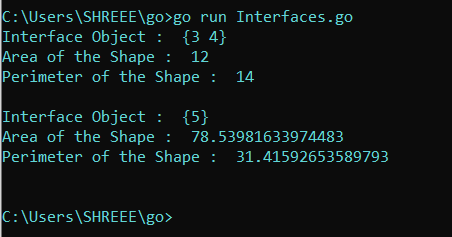
c := circle{radius: 5}

measure(r)

measure(c)

}

**Output :**



**Result :** Sucessfully executed all programs.