Go Programming Lab Manual

Contents

1 Introduction to Go

Go (Golang) is a statically typed, compiled language designed for simplicity and efficiency.

Key Features:

- **Speed:** Compiled to machine code.
- **Concurrency:** Built-in support via goroutines and channels.
- **Simplicity:** Clean syntax with minimal keywords.

2 Lab Setup

2.1 Install Go

Download from: https://go.dev/dl/

Add Go to your system PATH.

Verify installation:

```
go version
# Output: go version go1.21.0
```

2.2 Install VS Code & Extensions

Download VS Code: https://code.visualstudio.com

Recommended extensions:

- Go (by Google)
- Code Runner

3 Basic Syntax & Output

3.1 Exercise 1: Hello, World!

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, Go!")
}
```

Output: Hello, Go!

Task: Modify the program to print your name.

3.2 Exercise 2: Variables & Constants

```go
package main

import "fmt"

func main() {
    var x int = 5
    const y = 10
    x += 1
    fmt.Printf("x = %d, y = %d\n", x, y)
}
```

Output: x = 6, y = 10
Task: Declare a string variable and print it.

3.3 Exercise 3: Arithmetic Operations

```go
package main

import "fmt"

func main() {
    a := 10
    b := 3
    fmt.Printf("Sum: %d\n", a + b)
    fmt.Printf("Product: %d\n", a * b)
}
```

Output: Sum: 13
Product: 30
Task: Calculate the remainder of 15 divided by 4.

4 Control Flow

4.1 Exercise 4: Conditionals (if-else)

```go
package main

import "fmt"

func main() {
    n := -5
    if n > 0 {
        fmt.Println("Positive")
    } else if n < 0 {
        fmt.Println("Negative")
    } else {
        fmt.Println("Zero")
    }
}
```

Output: Negative
Task: Write a program to check if a number is even or odd.

4.2 Exercise 5: Loops (for)

```
package main

import "fmt"

func main() {
    for i := 1; i <= 3; i++ {
        fmt.Printf("Iteration %d\n", i)
    }

    counter := 1
    for counter <= 3 {
        fmt.Printf("Counter: %d\n", counter)
        counter++
    }
}
```

Output: Iteration 1
Iteration 2
Iteration 3
Counter: 1
Counter: 2
Counter: 3
Task: Print numbers from 10 to 1 in reverse.

5 Functions & Multiple Return Values

5.1 Exercise 6: Basic Functions

```
package main

import "fmt"

func square(x int) int {
    return x * x
}

func main() {
    fmt.Printf("Square of 5: %d\n", square(5))
}
```

Output: Square of 5: 25

Task: Write a function to calculate the factorial of a number.

5.2 Exercise 7: Multiple Return Values

```go
package main

import "fmt"

func swap(a, b string) (string, string) {
    return b, a
}

func main() {
    x, y := swap("hello", "world")
    fmt.Println(x, y)
}
```

Output: world hello

Task: Write a function that returns both the sum and product of two integers.

6 Data Structures

6.1 Exercise 8: Arrays & Slices

```go
package main

import "fmt"

func main() {
    arr := [3]int{10, 20, 30}
    slice := append(arr[:], 40)
    fmt.Printf("First element: %d\n", arr[0])
    fmt.Println("Slice:", slice)
}
```

Output: First element: 10
Slice: [10 20 30 40]

Task: Create a slice of strings and iterate over it.

6.2 Exercise 9: Maps & Structs

```go
package main

import "fmt"

```

```go
type Person struct {
    Name string
    Age  int
}

func main() {
    dict := map[string]string{"name": "Alice", "job": "Engineer"}
    fmt.Println("Job:", dict["job"])

    p := Person{Name: "Alice", Age: 30}
    fmt.Printf("Person: %s, Age: %d\n", p.Name, p.Age)
}
```

Output: Job: Engineer
Person: Alice, Age: 30

Task: Add a new key-value pair to the map.

7 Error Handling

7.1 Exercise 10: Error Checking

```go
package main

import (
    "fmt"
    "strconv"
)

func parseNumber(s string) (int, error) {
    return strconv.Atoi(s)
}

func main() {
    if num, err := parseNumber("123"); err == nil {
        fmt.Println("Number:", num)
    } else {
        fmt.Println("Error:", err)
    }
}
```

Output: Number: 123

Task: Modify the program to handle an invalid input like `"abc"`.

8 File I/O

8.1 Exercise 11: Read/Write Files

```go
package main

import (
    "os"
    "fmt"
)

func main() {
    os.WriteFile("data.txt", []byte("Go is efficient!"), 0644)
    data, _ := os.ReadFile("data.txt")
    fmt.Println("File content:", string(data))
}
```

Output: File content: Go is efficient!
Task: Append a new line to the file.

9 Concurrency

9.1 Exercise 12: Goroutines & Channels

```go
package main

import (
    "fmt"
    "time"
)

func printNumbers(ch chan int) {
    for i := 1; i <= 3; i++ {
        ch <- i
        time.Sleep(time.Second)
    }
    close(ch)
}

func main() {
    ch := make(chan int)
    go printNumbers(ch)
    for num := range ch {
        fmt.Println("Received:", num)
    }
}
```

Output: Received: 1
Received: 2
Received: 3
Task: Create two goroutines to calculate the sum of two arrays concurrently.

10 Libraries & Packages

10.1 Exercise 13: Using External Packages (e.g., Gin)

```
go mod init example.com/myproject
go get -u github.com/gin-gonic/gin
```

```go
package main

import "github.com/gin-gonic/gin"

func main() {
    r := gin.Default()
    r.GET("/", func(c *gin.Context) {
        c.String(200, "Hello from Gin!")
    })
    r.Run()
}
```

Output: Access `http://localhost:8080` to see the message.
Task: Create a GET endpoint that returns JSON data.

11 Debugging in VS Code

Set breakpoints in the editor. Press F5 to start debugging.

12 Next Steps

Explore Go documentation: `https://go.dev/doc/`

13 Troubleshooting

- Ensure Go binary is added to PATH.
- Use `go mod tidy` to resolve missing modules.
- Restart VS Code if extensions don't activate.

14 Practice Projects

14.1 1. Prime Number Checker

Objective: Implement a function to check if a number is prime.

- Create `isPrime(n int) bool` function
- Handle numbers 1 appropriately

- Use optimal prime-checking algorithm
- Test cases: 0, 2, 13, and 27

14.2 2. Slice Average Calculator

Objective: Process numeric data using slices and error handling.

- Create `average(numbers []float64) (float64, error)`
- Return error for empty slices
- Calculate mean of slice elements
- Test with `[5.2, 6.8, 9.1]` and empty slice

14.3 3. Employee Management System

Objective: Work with structs and slices.

- Define `Employee` struct with Name, Position, Salary
- Create slice of 5 employees
- Write `highestPaid(employees []Employee) Employee` function
- Print the highest earner's details

14.4 4. Robust File Appender

Objective: Implement safe file operations with error handling.

- Check if `"data.txt"` exists before writing
- Append new content instead of overwriting
- Handle file I/O errors explicitly
- Print success/failure messages

14.5 5. Concurrent Array Sum

Objective: Practice goroutines and channels.

- Create two arrays of numbers
- Sum each array in separate goroutines
- Use channels to return results
- Calculate and print total combined sum

14.6 6. JSON API Endpoint

Objective: Create web endpoints with Gin framework.

- Create `/user` endpoint returning JSON
- Response format:

  ```
  {
      "id": 123,
      "name": "Alice",
      "email": "alice@example.com"
  }
  ```

- Set proper Content-Type header

14.7 7. Bank Account Manager

Objective: Implement OOP-like behavior using struct methods.

- Create `BankAccount` struct with balance
- Methods: `Deposit(amount float64)`, `Withdraw(amount float64) error`
- Prevent overdrafts in Withdraw method
- Add `Balance() float64` accessor

14.8 8. Shape Interface

Objective: Understand interface implementation.

- Create `Shape` interface with `Area() float64` method
- Implement for `Circle` (radius) and `Rectangle` (width/height)
- Create slice of different shapes
- Calculate and print each shape's area