Contents

Chapter 1

Chapter one

Go code is organised into packages, which are similar to libraries or modules in other languages. A package consists of one or more `.go` source files in a single directory that define what the packages does.

Each source file begins with a `package` declaration, here `package main` that states which package the file belongs to, followed by a list of other packages that it imports, and then the declarations of the program that are stored in that file.

the `fmt` package contains functions for printing formatted output and scanning input. `Println` is one of the basic output functions in `fmt`; it prints one or more values, seperated by spaces, with a newline character at the end so that the values appear as a single line of output.

Package `main` is special. It defines a standalone executable program, not a library. Within package main the *function* main is also special – it's where execution of the program begins. Whatever `main` does is what the program does. ofcourse, `main` will normally call upon functions in other packages to do much of the work, such as function `fmt.Println`.

Command Line Arguments

The Variable `os.Args` is a *slice* of strings. Slices are a fundamental notion in Go. A slice is a dynamically sized sequence `s` of array of elements where individual elements can be accessed by `s[i]` and a contiguous subsequence as `s[m:n]`

The number of elements is given by `len(s)`.

The first element of `os.Args`, `os.Args[0]`, is the name of the command itself; The other elements are arguments that were presented to the program when it started execution

```
for _, arg := range os.Args[1: ] {
    s += sep + arg
    sep = " "
}
fmt.Println(s)
```

here range produces two values, `index` and `value` of the element at that index so arg handles `value` and `_` handles the index

each time around the loop, the string `s` gets completely new contents. The `+=` statement makes a new string by concatenating the old string, a space character, and the next argument, then assigns the new string to s. The old contents of `s` are no longer in use, so they will be garbage-collected in due course.

If the amount of data involved is large, this could be costly. A simple and more efficient solution would be to use `Join` function from the `strings` package

```
func main() {
    fmt.Println(strings.Join(os.Args[1:], " "))
}
```

Variables

The version of strings above uses short variable declaration There are several other ways to declare a variable in go

```
s := ""
var s = ""
var s string
var s string = ""
```

Chapter 2

Finding Duplicate Lines

finding duplicate lines is partly inspired from `uniq` command, which looks for adjacent duplicate lines. The structures and packages used are models that can be easily adapted.

A *map* holds a set of key / value pairs and provides constant - time operations to store, retrieve, or test for an item in the set.

`a map is like a dictionary in python ?`

The key may be of any type whose values can be compared with `==`, strings being the most common example; The value may be of any type at all. In the example, the keys are strings and the values are `ints`. The built-in function `make` creates a new empty map;

The program uses a short variable declaration to create a new variable `input` that refers to a `bufio.Scanner`

```
input := bufio.NewScanner(os.Stdin)
```

The scanner reads from the program's standard input. Each call to `input.Scan()` reads the next line and removes the newline character from the end; the result can be retrieved by calling `input.Text()` The `Scan` function returns `true` if there is a line and `false` when there is no more input.

Chapter 3

Getting Started

- place the source files `file_name.go` in src directories

Chapter 4

GoNotes

Understanding Go Programming Language

types in go:

```
func add(x float64, y float64) float64 {
    return x + y
}
```

can also be written as

```
func add(x, y float64) float64 {
    return x + y
}

var num1, num2 float64 = 5.6, 9.5
// short circuit declaration
```

inorder to return multiple elements from a function

```
func return_multiple_items(a, b string) (string, string) {
    return a, b
}
```

Chapter 5

pointers in go

```
func main() {
    x := 15
    a := &x // memory address

    fmt.Println(a)

    // print the value of x
    fmt.Println(*a)

    *a = 5
    fmt.Println(x)
    fmt.Println(a)
}
```