MODUL VIII
REKURSIF FUNCTION, ANONYMOUS FUNCTION,
CLOSURE FUNCTION & FUNCTION AS PARAMETER

Pertemuan : 8
Waktu : 8 x 60 menit (Online)

1.1 Tujuan Modul III

Setelah mahasiswa mempelajari materi ini, diharapkan dapat :

1. Memahami penggunaan function sebagai parameter di golang.
2. Memahami penggunaan function rekursif di golang.
3. Memahami penggunaan function closure di golang.
4. Memahami penggunaan function anonymous di golang.

1.2 Landasan Teori

Setelah di materi sebelumnya kita belajar mengenai fungsi yang mengembalikan nilai balik berupa fungsi, kali ini topiknya tidak kalah unik, yaitu fungsi yang digunakan sebagai parameter.

Di Go, fungsi bisa dijadikan sebagai tipe data variabel. Dari situ sangat memungkinkan untuk menjadikannya sebagai parameter juga.

1.2.1 Penerapan Fungsi Sebagai Parameter

Cara membuat parameter fungsi adalah dengan langsung menuliskan skema fungsi nya sebagai tipe data. Contohnya bisa dilihat pada kode berikut.

```
package main

import "fmt"
import "strings"

func filter(data []string, callback func(string) bool) []string {
    var result []string
    for _, each := range data {
        if filtered := callback(each); filtered {
            result = append(result, each)
```

```
        }
    }
    return result
}
```

Parameter `callback` merupakan sebuah closure yang dideklarasikan bertipe `func(string) bool`. Closure tersebut dipanggil di tiap perulangan dalam fungsi `filter()`.

Fungsi `filter()` sendiri kita buat untuk filtering data array (yang datanya didapat dari parameter pertama), dengan kondisi filter bisa ditentukan sendiri. Di bawah ini adalah contoh pemanfaatan fungsi tersebut.

```
func main() {
    var data = []string{"wick", "jason", "ethan"}
    var dataContainsO = filter(data, func(each string)
bool {
        return strings.Contains(each, "o")
    })
    var dataLenght5 = filter(data, func(each string) bool
{
        return len(each) == 5
    })

    fmt.Println("data asli \t\t:", data)
    // data asli : [wick jason ethan]

    fmt.Println("filter ada huruf \"o\"\t:",
dataContainsO)
    // filter ada huruf "o" : [jason]

    fmt.Println("filter jumlah huruf \"5\"\t:",
dataLenght5)
    // filter jumlah huruf "5" : [jason ethan]
}
```

Ada cukup banyak hal yang terjadi didalam setiap pemanggilan fungsi `filter()` di atas. Berikut merupakan penjelasannya.

1. Data array (yang didapat dari parameter pertama) akan di-looping.
2. Di tiap perulangannya, closure `callback` dipanggil, dengan disisipkan data tiap elemen perulangan sebagai parameter.

3. Closure `callback` berisikan kondisi filtering, dengan hasil bertipe `bool` yang kemudian dijadikan nilai balik dikembalikan.
4. Di dalam fungsi `filter()` sendiri, ada proses seleksi kondisi (yang nilainya didapat dari hasil eksekusi closure `callback`). Ketika kondisinya bernilai `true`, maka data elemen yang sedang diulang dinyatakan lolos proses filtering.
5. Data yang lolos ditampung variabel `result`. Variabel tersebut dijadikan sebagai nilai balik fungsi `filter()`.

Pada `dataContainsO`, parameter kedua fungsi `filter()` berisikan statement untuk deteksi apakah terdapat substring `"o"` di dalam nilai variabel `each` (yang merupakan data tiap elemen), jika iya, maka kondisi filter bernilai `true`, dan sebaliknya.

pada contoh ke-2 (`dataLength5`), closure `callback` berisikan statement untuk deteksi jumlah karakter tiap elemen. Jika ada elemen yang jumlah karakternya adalah 5, berarti elemen tersebut lolos filter.

Memang butuh usaha ekstra untuk memahami pemanfaatan closure sebagai parameter fungsi. Tapi setelah paham, penerapan teknik ini pada kondisi yang tepat akan sangat membantu proses pembuatan aplikasi.

1.2.2 Alias Skema Closure

Kita sudah mempelajari bahwa closure bisa dimanfaatkan sebagai tipe parameter, contohnya seperti pada fungsi `filter()`. Pada fungsi tersebut kebetulan skema tipe parameter closure-nya tidak terlalu panjang, hanya ada satu buah parameter dan satu buah nilai balik.

Pada fungsi yang skema-nya cukup panjang, akan lebih baik jika menggunakan alias, apalagi ketika ada parameter fungsi lain yang juga menggunakan skema yang sama. Membuat alias fungsi berarti menjadikan skema fungsi tersebut menjadi tipe data baru. Caranya dengan menggunakan keyword `type`. Contoh:

```
type FilterCallback func(string) bool

func filter(data []string, callback FilterCallback) []string {
    // ...
```

```
}
```

Skema `func(string) bool` diubah menjadi tipe dengan nama `FilterCallback`. Tipe tersebut kemudian digunakan sebagai tipe data parameter `callback`.

1.2.2.1 Penggunaan Fungsi string.Contains()

Inti dari fungsi ini adalah untuk deteksi apakah sebuah substring adalah bagian dari string, jika iya maka akan bernilai `true`, dan sebaliknya. Contoh penggunaannya:

```
var result = strings.Contains("Golang", "ang")
// true
```

Variabel `result` bernilai `true` karena string `"ang"` merupakan bagian dari string `"Golang"`.

1.2.3 Fungsi Closure

Definisi Closure adalah sebuah fungsi yang bisa disimpan dalam variabel. Dengan menerapkan konsep tersebut, kita bisa membuat fungsi didalam fungsi, atau bahkan membuat fungsi yang mengembalikan fungsi.

Closure merupakan anonymous function atau fungsi tanpa nama. Biasa dimanfaatkan untuk membungkus suatu proses yang hanya dipakai sekali atau dipakai pada blok tertentu saja.

1.2.3.1 Closure Disimpan Sebagai Variabel

Sebuah fungsi tanpa nama bisa disimpan dalam variabel. Variabel yang menyimpan closure memiliki sifat seperti fungsi yang disimpannya. Di bawah ini adalah contoh program sederhana untuk mencari nilai terendah dan tertinggi dari suatu array. Logika pencarian dibungkus dalam closure yang ditampung oleh variabel `getMinMax`.

```
package main

import "fmt"

func main() {
    var getMinMax = func(n []int) (int, int) {
```

```
        var min, max int
        for i, e := range n {
            switch {
            case i == 0:
                max, min = e, e
            case e > max:
                max = e
            case e < min:
                min = e
            }
        }
        return min, max
    }

    var numbers = []int{2, 3, 4, 3, 4, 2, 3}
    var min, max = getMinMax(numbers)
    fmt.Printf("data : %v\nmin  : %v\nmax  : %v\n", numbers, min,
max)
}
```

Bisa dilihat pada kode di atas bagaimana sebuah closure dibuat dan dipanggil. Sedikit berbeda memang dibanding pembuatan fungsi biasa. Fungsi ditulis tanpa nama, lalu ditampung dalam variabel.

```
var getMinMax = func(n []int) (int, int) {
    // ...
}
```

Cara pemanggilannya, dengan menuliskan nama variabel tersebut sebagai fungsi, seperti pemanggilan fungsi biasa.

```
var min, max = getMinMax(numbers)
```

1.2.3.2 Immediately-Invoked Function Expression (IIFE)

Closure jenis ini dieksekusi langsung pada saat deklarasinya. Biasa digunakan untuk membungkus proses yang hanya dilakukan sekali, bisa mengembalikan nilai, bisa juga tidak.

Di bawah ini merupakan contoh sederhana penerapan metode IIFE untuk filtering data array.

```
package main

import "fmt"
```

```
func main() {
    var numbers = []int{2, 3, 0, 4, 3, 2, 0, 4, 2, 0, 3}

    var newNumbers = func(min int) []int {
        var r []int
        for _, e := range numbers {
            if e < min {
                continue
            }
            r = append(r, e)
        }
        return r
    }(3)

    fmt.Println("original number :", numbers)
    fmt.Println("filtered number :", newNumbers)
}
```

Ciri khas IIFE adalah adanya kurung parameter tepat setelah deklarasi closure berakhir. Jika ada parameter, bisa juga dituliskan dalam kurung parameternya.

```
var newNumbers = func(min int) []int {
    // ...
}(3)
```

Pada contoh diatas IIFE menghasilkan nilai balik yang kemudian ditampung `newNumber`. Perlu diperhatikan bahwa yang ditampung adalah **nilai kembaliannya** bukan body fungsi atau **closure**.

1.2.3.3 Closure Sebagai Nilai Kembalian

Salah satu keunikan closure lainnya adalah bisa dijadikan sebagai nilai balik fungsi, cukup aneh memang, tapi pada suatu kondisi teknik ini sangat membantu. Di bawah ini disediakan sebuah fungsi bernama `findMax()`, fungsi ini salah satu nilai kembaliannya berupa closure.

```
package main

import "fmt"

func findMax(numbers []int, max int) (int, func() []int) {
    var res []int
    for _, e := range numbers {
        if e <= max {
            res = append(res, e)
```

```
            }
        }
        return len(res), func() []int {
            return res
        }
    }
```

Nilai kembalian ke-2 pada fungsi di atas adalah closure dengan skema `func() []int`. Bisa dilihat di bagian akhir, ada fungsi tanpa nama yang dikembalikan.

```
return len(res), func() []int {
    return res
}
```

Sedikit tentang fungsi `findMax()`, fungsi ini digunakan untuk mencari banyaknya angka-angka yang nilainya di bawah atau sama dengan angka tertentu. Nilai kembalian pertama adalah jumlah angkanya. Nilai kembalian kedua berupa closure yang mengembalikan angka-angka yang dicari. Berikut merupakan contoh implementasi fungsi tersebut.

```
func main() {
    var max = 3
    var numbers = []int{2, 3, 0, 4, 3, 2, 0, 4, 2, 0, 3}
    var howMany, getNumbers = findMax(numbers, max)
    var theNumbers = getNumbers()

    fmt.Println("numbers\t:", numbers)
    fmt.Printf("find \t: %d\n\n", max)

    fmt.Println("found \t:", howMany)     // 9
    fmt.Println("value \t:", theNumbers) // [2 3 0 3 2 0 2 0 3]
}
```

1.2.4 Fungsi Anonymous

Fungsi anonim/Anonymous adalah fungsi yang dideklarasikan tanpa pengenal bernama untuk merujuknya. Fungsi anonim dapat menerima input dan mengembalikan output, sama seperti fungsi standar.Menetapkan fungsi ke variabel.

Fungsi anonim dapat digunakan untuk memuat fungsionalitas yang tidak perlu diberi nama dan mungkin untuk penggunaan jangka pendek.Contoh :

```
package main

import "fmt"

var (
      area = func(l int, b int) int {
            return l * b
      }
)

func main() {
      fmt.Println(area(20, 30))
}
```

Contoh

Melewati argumen ke fungsi anonim.

```
package main

import "fmt"

func main() {
      func(l int, b int) {
            fmt.Println(l * b)
      }(20, 30)
}
```

Contoh

Fungsi didefinisikan untuk menerima parameter dan mengembalikan nilai.

```
package main

import "fmt"

func main() {
      fmt.Printf("100 (°F) = %.2f (°C)\n",
```

```
            func(f float64) float64 {
                  return (f - 32.0) * (5.0 / 9.0)
            }(100),
      )
}
```

1.2.4 Fungsi Rekursif

Rekursif berarti suatu proses yang memanggil dirinya sendiri. Dalam rekursif sebenarnya terkandung pengertian prosedur atau fungsi. Rekursif merupakan teknik pemrograman yang penting, dan beberapa bahasa pemrograman mendukung keberadaan proses rekursif ini.

Pemanggilan prosedur atau fungsi ke dirinya sendiri bisa berarti proses yang mungkin berulang yang tidak bisa diketahui kapan akan berakhir. Dalam pemakaian sehari-hari, rekursif merupakan teknik pemrograman yang berdaya guna untuk digunakan pada pekerjaan pemrograman dengan mengekspresikannya ke dalam suku-suku dari program lain dengan menambahkan langkah-langkah sejenis.Contoh :

```
package main

import (
"fmt"
)

func getFactorial(n int) int {

      if n == 0 || n == 1 {
       return 1
      }

      return n * getFactorial(n-1)
}

func main() {
      val := getFactorial(5)
      fmt.Println(val)
}
```

1.3 Praktikum

1.3.1 Latihan Praktikum

1. Buatlah algoritma fungsi rekursif dan pemanggilnya untuk menghitung deret berikut :
    a. Sum = 1 + 3 + 5 + 7 + …
    b. Sum = 1 + 5 + 25 + 125 + 625 + …
    c. Sum = 1 + 6 + 36 + 216 + 1296 + ...
2. Buatlah algoritma fungsi rekursif dan pemanggilnya untuk membalikkan urutan huruf dalam sebuah kata. Contoh : huruf ‘MAKAN’ dibalik menjadi ‘NAKAM’ ! parameter input terhadap fungsi adalah kata !

1.3.2 Tugas Praktikum

1. Buatlah algoritma fungsi rekursif dan pemanggilnya untuk menghitung deret berikut :
    a. Sum = 20 + 10 + 6.66 + 5 + ...
    b. Sum = 100 + 50 + 33.33 + 25 + ...
    c. Sum = 1 + 0.5 + 0.33 + 0.25 + ...
    d. Sum = 1/2x + 2/4x + 3/9x + 4/16x + ...
    e. Sum = 2x/1 + 4x/2 + 9x/3 + 16x/4 + ...
2. Buatlah algoritma fungsi rekursif dengan menghitung huruf non kapital pada kalimat. Contoh : ‘Makan’ hasilnya 4.
3. Buatlah algoritma fungsi rekursif untuk menghitung angka pada suatu kalimat. Contoh : ‘JKT48’ hasilnya 2