2 FUNCTION & METHOD

Objektif :

- **Mahasiswa mampu memahami dan menggunakan function pada Golang**
- **Mahasiswa mampu memahami dan menggunakan berbagai function yang sesuai dengan penerapan pada Golang.**
- **Mahasiswa mampu dan menggunakan method pada Golang.**

Pada bagian ini akan membahas tentang fungsi dan method pada Golang yang merupakan aspek penting dalam pemrograman. Definisi fungsi sendiri adalah sekumpulan blok kode yang dibungkus dengan nama tertentu. Setiap program pada Golang memiliki setidaknya satu fungsi, yaitu main ( ). Penerapan fungsi yang tepat akan menjadikan kode lebih modular dan juga *dry* (kependekan dari *don't repeat yourself*), karena tak perlu menuliskan banyak proses berkali-kali, cukup sekali saja dan tinggal panggil jika dibutuhkan. Fungsi juga dikenal sebagai method, sub-routine, atau procedure.

2.1 Penerapan Fungsi

Definisi fungsi dalam bahasa pemrograman Go terdiri dari *function header* dan *function body*. Berikut ini semua bagian dari suatu fungsi:

- **Func** − memulai deklarasi fungsi.
- **Function Name** − nama sebenarnya dari fungsi tersebut. Nama fungsi dan daftar parameter.
- **Parameters** − Parameter seperti tempat penampung. Saat suatu fungsi dipanggil, untuk meneruskan nilai ke parameter. Nilai ini disebut sebagai parameter atau argumen aktual. Daftar parameter mengacu pada jenis, urutan, dan jumlah parameter fungsi. Parameter bersifat opsional; artinya, suatu fungsi mungkin tidak mengandung parameter.
- **Return Type** − Suatu fungsi dapat mengembalikan daftar nilai. Return_types adalah daftar tipe data nilai yang dikembalikan fungsi. Beberapa fungsi melakukan operasi yang diinginkan tanpa mengembalikan nilai. Dalam hal ini, return_type adalah yang tidak diperlukan.

- **Function Body** − Ini berisi kumpulan pernyataan yang menentukan apa fungsi tidak.

Berikut adalah bentuk umum definisi fungsi dalam bahasa pemrograman Golang:

```
func function_name( [parameter list] ) [return_types]
{
   body of the function
}
```

Contoh penerapan perintah function pada Golang:

```
package main

import "fmt"

import "strings"

func main( ) {

var pesan= [] string{"belajar", "Function"}

printMessage( "Yuk", pesan)

}

func printMessage( message string, arr [] string) {

var nameString = strings. Join( arr, " ")

fmt. Println( message, nameString)

}
```

Pada kode di atas dibuat fungsi baru dengan nama ***printMessage*** yang memiliki 2 buah

parameter yaitu string ***message*** dan slice string ***arr*** . Di dalam p***rintMessage*** , nilai ***arr*** yang merupakan slice string digabungkan menjadi sebuah string dengan pembatas adalah karakter **spasi**. Penggabungan slice dapat dilakukan dengan memanfaatkan fungsi ***strings***. ***Join( )*** . Fungsi ini berada di dalam package ***strings***.

2.2 Fungsi Dengan Return Value / Nilai Balik

Sebuah fungsi dapat mengembalikan suatu nilai atau dapat didesain tidak mengembalikan apa-apa (void). Fungsi yang memiliki nilai kembalian, harus ditentukan tipe data nilai baliknya pada saat deklarasi. Program berikut merupakan contoh penerapan fungsi yang memiliki return value.

```
package main
import (
"fmt"
"math/rand"
"time"
)
func main( ) {
rand. Seed( time. Now( ) . Unix( ) )
var randomValue int
randomValue = randomWithRange( 2, 10)
fmt. Println( "random number: ", randomValue)
```

```
randomValue = randomWithRange( 2, 10)
fmt. Println( "random number: ", randomValue)
randomValue = randomWithRange( 2, 10)
fmt. Println( "random number: ", randomValue)
}
func randomWithRange( min, max int) int {
var value = rand. Int( ) % ( max - min + 1) + min
return value
}
```

Di dalam fungsi ***randomWithRange*** terdapat proses generate angka acak, yang angka tersebut kemudian digunakan sebagai nilai kembalian.

Cara menentukan tipe data nilai balik fungsi adalah dengan menuliskan tipe data yang diinginkan setelah kurung parameter. Bisa dilihat pada kode di atas, bahwa ***int*** merupakan tipe data nilai balik fungsi ***randomWithRange***.

```
func randomWithRange( min, max int) int
```

Sedangkan cara untuk mengembalikan nilainya adalah dengan menggunakan keyword ***return*** diikuti data yang ingin dikembalikan. Pada contoh di atas, ***return value*** artinya nilai variabel ***value*** dijadikan nilai kembalian fungsi. Eksekusi keyword ***return*** akan menjadikan proses dalam blok fungsi berhenti pada saat itu juga. Semua statement setelah keyword tersebut tidak akan dieksekusi.

Pada kode diatas juga terdapat fungsi ***rand. Seed()*** dimana fungsi ini diperlukan untuk memastikan bahwa angka random yang akan di-generate benar - benar acak. Untuk itu dapat digunakan angka apa saja sebagai nilai parameter fungsi ini (umumnya diisi ***time. Now( ) . Unix( )***).

2.1.1. Deklarasi Perameter Bertipe Sama

Dalam deklarasi ini khusus untuk fungsi yang tipe data parameternya sama, bisa ditulis dengan gaya yang unik. Tipe datanya dituliskan cukup sekali saja di akhir. Contohnya bisa dilihat pada kode berikut.

```
func nameOfFunc( paramA type, paramB type, paramC type) returnType
func nameOfFunc( paramA, paramB, paramC type) returnType
func randomWithRange( min int, max int) int
func randomWithRange( min, max int) int
```

2.3 Fungsi Variadic

Golang mengadopsi konsep variadic function atau pembuatan fungsi dengan parameter sejenis yang tak terbatas. Maksud tak terbatas disini adalah jumlah parameter yang disisipkan ketika pemanggilan fungsi bisa berapa saja. Parameter variadic memiliki sifat yang mirip dengan slice. Nilai parameter-parameter yang disisipkan memiliki tipe data yang sama, dan akan ditampung oleh sebuah variabel saja. Cara pengaksesan tiap datanya juga sama, dengan menggunakan indeks.

Deklarasi parameter variadic sama dengan cara deklarasi variabel biasa, pembedanya pada parameter jenis ini ditambahkan tanda 3 titik (. . .) setelah penulisan variabel (sebelum tipe data). Nantinya semua nilai yang disisipkan sebagai parameter akan ditampung oleh variabel tersebut.

Berikut merupakan contoh penerapannya:

```
package main
import "fmt"
func main( ) {
var avg = calculate( 2, 4, 3, 5, 4, 3, 3, 5, 5, 3)
var msg = fmt.Sprintf("Rata-rata: %.2f", avg)

fmt.Println( msg)
}
func calculate( numbers ...int) float64 {
var total int = 0
for _, number := range numbers {
total += number
}
var avg = float64( total) / float64( len( numbers) )
return avg
}
```

Fungsi ***calculate( )*** , parameter numbers dideklarasikan dengan disisipkan tanda 3 titik (. . .) sebelum penulisan tipe data-nya. Menandakan bahwa ***numbers*** adalah sebuah parameter variadic dengan tipe data ***int*** .

```
func calculate( numbers . . . int) float64 {
```

Pada pemanggilan fungsi disisipkan banyak parameter sesuai kebutuhan.

```
var avg = calculate( 2, 4, 3, 5, 4, 3, 3, 5, 5, 3)
```

Nilai tiap parameter bisa diakses seperti cara pengaksesan tiap elemen slice. Pada contoh di atas metode yang dipilih adalah ***for - range*** .

```
for _, number := range numbers
```

- Fungsi ***fmt.Sprintf()***

Fungsi ***fmt.Sprintf( )*** pada dasarnya sama dengan ***fmt.Printf( )***, hanya saja fungsi ini tidak menampilkan nilai, melainkan mengembalikan nilainya dalam bentuk string. Pada kasus di atas, nilai hasil ***fmt. Sprintf( )*** ditampung oleh variabel ***msg*** . Selain ***fmt.Sprintf( )*** , ada juga ***fmt.Sprint( )*** dan ***fmt.Sprintln( )*** .

- Fungsi ***float64()***

  Sebelumnya sudah dibahas bahwa ***float64*** merupakan tipe data. Tipe data jika ditulis sebagai fungsi (penandanya ada tanda kurungnya) berguna untuk casting. Casting sendiri adalah teknik untuk konversi tipe sebuah data ke tipe lain. Pada contoh di atas, variabel total yang tipenya adalah ***int***, dikonversi menjadi ***float64***, begitu juga **len(numbers)** yang menghasilkan ***int*** dikonversi ke ***float64*** . Variabel ***avg*** perlu dijadikan ***float64*** karena penghitungan rata-rata lebih sering menghasilkan nilai desimal. Operasi bilangan (perkalian, pembagian, dan lainnya) pada Golang hanya bisa dilakukan jika tipe datanya sejenis. Maka dari itulah perlu adanya casting ke tipe ***float64*** pada tiap operand.

- Fungsi Variadic Menggunakan Data Slice

  Slice bisa digunakan sebagai parameter variadic. Caranya cukup mudah, yaitu dengan menambahkan tanda 3 titik setelah nama variabel ketika memasukannya ke parameter.

  Contohnya bisa dilihat pada kode berikut.

```
var avg = calculate( 2, 4, 3, 5, 4, 3, 3, 5, 5, 3)
var msg = fmt.Sprintf("Rata-rata: %.2f", avg)

fmt.Println( msg)
```

Pada kode di atas, variabel ***numbers*** yang merupakan slice int, disisipkan ke fungsi ***calculate( )*** sebagai parameter variadic (bisa dilihat tanda 3 titik setelah penulisan variabel). Teknik ini sangat berguna ketika sebuah data slice ingin difungsikan sebagai

parameter variadic.

Perhatikan juga kode berikut ini. Intinya adalah sama, hanya caranya yang berbeda.

```
var numbers = [] int{2, 4, 3, 5, 4, 3, 3, 5, 5, 3}
var avg = calculate( numbers. . . )

// atau

var avg = calculate( 2, 4, 3, 5, 4, 3, 3, 5, 5, 3)
```

Pada deklarasi parameter fungsi variadic, tanda 3 titik (. . .) dituliskan sebelum tipe data parameter. Sedangkan pada pemanggilan fungsi dengan menyisipkan parameter array, tanda tersebut dituliskan dibelakang variabelnya.

2.4 Fungsi Closure

Definisi termudah Closure adalah sebuah fungsi yang bisa disimpan dalam variabel. Dengan menerapkan konsep tersebut, sangat mungkin untuk 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.

- **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)
}
```

Pada kode di atas bagiamana cara closure dibuat dan dipanggil. Sedikit berbeda memang dibanding pembuatan fungsi biasa. Fungsi ditulis tanpa nama, lalu ditampung dalam variable.

```
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)
```

2.5 Fungsi sebagai Parameter

Pada Golang fungsi bisa dijadikan sebagai tipe data variable dan dapat menjadikannya sebagai parameter juga. Cara membuat parameter fungsi adalah dengan langsung menuliskan skema fungsinya 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 digunakan 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{"gajah", "buaya", "ular"}
var dataContainsO = filter( data, func(each string) bool {
return strings. Contains( each, "u")
})
var dataLenght5 = filter( data, func(each string) bool {
return len( each) == 5
})
fmt. Println("data asli \t\t: ", data)
// data asli : [gajah buaya ular]
fmt. Println("filter ada huruf \"u\"\t:", dataContainsO)
// filter ada huruf "u" : [gajah]
fmt. Println("filter jumlah huruf \"5\"\t: ", dataLenght5)
// filter jumlah huruf "5" : [gajah buaya]
}
```

Ada cukup banyak hal yang terjadi didalam tiap 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 ***"u"*** 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.

2.6 Method

Method adalah fungsi yang hanya bisa di akses lewat variabel objek. Method merupakan bagian dari ***struct*** .Keunggulan method dibanding fungsi biasa adalah memiliki akses ke property struct hingga level *private*. Dan juga, dengan menggunakan method sebuah proses bisa di-enkapsulasi dengan baik. Cara menerapkan method sedikit berbeda dibanding penggunaan fungsi. Ketika deklarasi, ditentukan juga siapa pemilik method tersebut.

Contohnya bisa dilihat pada kode berikut:

```
package main

import "fmt"
import "strings"

type student struct {
name string
grade int

}
func ( s student) sayHello( ) {
fmt.Println( "halo", s.name)
}
func ( s student) getNameAt( i int) string {
return strings. Split( s.name, " ") [i- 1]
}
```

Cara deklarasi method sama seperti fungsi, hanya saja perlu ditambahkan deklarasi variable objek diantara keyword ***func*** dan nama fungsi. Struct yang digunakan akan menjadi pemilik method. ***func (s student) sayHello( )*** maksudnya adalah fungsi ***sayHello*** dideklarasikan sebagai method milik struct ***student*** . Pada contoh di atas struct ***student*** memiliki dua buah method, yaitu ***sayHello( )*** dan ***getNameAt( )*** .

Contoh pemanfaatan method diatas bisa dilihat pada kode berikut:

```
func main( ) {
var s1 = student{"Budi Putra", 17}
s1. sayHello( )
var name = s1.getNameAt( 2)
fmt. Println("nama panggilan : ", name)
}
```

Cara mengakses method sama seperti pengaksesan properti berupa variabel. Tinggal panggil saja methodnya.

```
s1.sayHello( )
var name = s1.getNameAt( 2)
```

Method memiliki sifat yang sama persis dengan fungsi biasa. Seperti bisa berparameter, memiliki nilai balik, dan lainnya. Dari segi sintaks, pembedanya hanya ketika pengaksesan dan deklarasi. Bisa dilihat di kode berikut, sekilas perbandingan penulisan fungsi dan method.

```
func sayHello( ) {
func (s student) sayHello( ) {

func getNameAt(i int) string {
func (s student) getNameAt(i int) string {
```

2.6 Method Pointer

Method pointer adalah method yang variabel objeknya dideklarasikan dalam bentuk pointer. Kelebihan method jenis ini adalah manipulasi data pointer pada property milik variabel tersebut bisa dilakukan.Pemanggilan method pointer sama seperti method biasa. Contohnya bisa dilihat di kode berikut.

```
func ( s *student) sayHello( ) {
fmt.Println( "halo", s.name)
}

func main( ) {
var s1 = student{"Budi Putra", 17}
s1.sayHello( )
```

```
}
```

Method pointer tetap bisa diakses lewat variabel objek biasa (bukan pointer) dengan cara yang nya masih sama. Contoh:

```
// pengaksesan method dari variabel objek biasa
var s1 = student{"Budi Putra", 17}

s1. sayHello( )
// pengaksesan method dari variabel objek pointer
var s2 = &student{"Ari Langit", 21}
s2.sayHello( )
```