MODUL IX
DEFER , EXIT, ERROR, PANIC dan RECOVER

Pertemuan : 9
Waktu : 8 x 60 Menit (Online)

1.1 Tujuan Modul IX

Setelah mahasiswa mempelajari materi ini, diharapkan dapat :

1. Memahami fungsi Defer dan Exit
2. Memahami fungsi Error, Panic, dan Recover

1.2 Landasan Teori

1.2.1 Defer & Exit

Defer digunakan untuk mengakhirkan eksekusi sebuah statement tepat sebelum blok fungsi selesai. Sedangkan Exit digunakan untuk menghentikan program secara paksa (ingat, menghentikan program, tidak seperti return yang hanya menghentikan blok kode).

1.2.1.1 Penerapan Defer

Seperti yang sudah dijelaskan secara singkat di atas, bahwa defer digunakan untuk mengakhirkan eksekusi baris kode dalam skope blok fungsi. Ketika eksekusi blok sudah hampir selesai, statement yang di-defer dijalankan. Defer bisa ditempatkan di mana saja, awal maupun akhir blok. Tetapi tidak mempengaruhi kapan waktu dieksekusinya, akan selalu dieksekusi di akhir.

```
package main

import "fmt"

func main() {

    defer fmt.Println("Aku diatas, tapi muncul dibawah :(")

    fmt.Println("Hello World!")

}
```

Contoh output dari kode diatas adalah :

```
Hello World!
Aku diatas, tapi muncul dibawah :(


...Program finished with exit code 0
Press ENTER to exit console.
```

Keyword `defer` di atas akan mengakhirkan ekseusi `fmt.Println("Aku diatas, tapi muncul dibawah :(")`, efeknya pesan `"Aku diatas, tapi muncul dibawah :("` akan muncul setelah `"Hello World!"`.

1.2.1.2 Kombinasi Defer dan IIFE

Contoh implementasi :

```
func main() {
    number := 3

    if number == 3 {
        fmt.Println("halo 1")
        defer fmt.Println("halo 3")
    }

    fmt.Println("halo 2")
}
```

Output:

```
halo 1
halo 2
halo 3
```

Pada contoh di atas `halo 3` akan tetap di print setelah `halo 2` meskipun statement defer dipergunakan dalam blok seleksi kondisi `if`. Hal ini karena defer eksekusinya terjadi pada akhir blok fungsi (dalam contoh di atas `main()`), bukan pada akhir blok `if`.

Agar `halo 3` bisa dimunculkan di akhir blok `if`, maka harus dibungkus dengan IIFE. Contoh:

```
func main() {
    number := 3

    if number == 3 {
        fmt.Println("halo 1")
        func() {
            defer fmt.Println("halo 3")
        }()
    }

    fmt.Println("halo 2")
}
```

Output:

```
halo 1
halo 3
halo 2
```

Bisa dilihat `halo 3` muncul sebelum `halo 2`, karena dalam blok seleksi kondisi `if` eksekusi defer terjadi dalam blok fungsi anonymous (IIFE).

1.2.1.3 Penerapan Fungsi os.Exit()

Exit digunakan untuk menghentikan program secara paksa pada saat itu juga. Semua statement setelah exit tidak akan di eksekusi, termasuk juga defer.

Fungsi `os.Exit()` berada dalam package `os`. Fungsi ini memiliki sebuah parameter bertipe numerik yang wajib diisi. Angka yang dimasukkan akan muncul sebagai **exit status** ketika program berhenti.

```
package main


import "fmt"

import "os"


func main() {

    defer fmt.Println("Hello:)")

    os.Exit(1)

    fmt.Println("Welcome to Golang")

}
```

Meskipun `defer fmt.Println("halo")` ditempatkan sebelum `os.Exit()`, statement tersebut tidak akan dieksekusi, karena di-tengah fungsi program dihentikan secara paksa.

1.2.2 Error, Panic, dan Recover

Error merupakan topik yang sangat penting dalam pemrograman Go. Di bagian ini kita akan belajar mengenai pemanfaatan error dan cara membuat custom error sendiri. Selain itu, kita juga akan belajar tentang penggunaan panic untuk memunculkan panic error, dan recover untuk mengatasinya.

1.2.2.1 Pemanfaatan Error

`error` merupakan sebuah tipe. Error memiliki 1 buah property berupa method `Error()`, method ini mengembalikan detail pesan error dalam string. Error termasuk tipe yang isinya bisa `nil`.
Di Go, banyak sekali fungsi yang mengembalikan nilai balik lebih dari satu. Biasanya, salah satu kembalian adalah bertipe `error`. Contohnya seperti pada fungsi `strconv.Atoi()`. Fungsi tersebut digunakan untuk konversi data string menjadi numerik. Fungsi ini mengembalikan 2 nilai balik. Nilai balik pertama adalah hasil konversi, dan nilai balik kedua adalah `error`. Ketika konversi berjalan mulus, nilai balik kedua akan bernilai `nil`. Sedangkan ketika konversi gagal, penyebabnya bisa langsung diketahui dari error yang dikembalikan.

Dibawah ini merupakan contoh program sederhana untuk deteksi inputan dari user, apakah numerik atau bukan. Dari sini kita akan belajar mengenai pemanfaatan error.

```
package main

import (
    "fmt"
    "strconv"
)

func main() {
    var input string
    fmt.Print("Type some number: ")
    fmt.Scanln(&input)

    var number int
    var err error
    number, err = strconv.Atoi(input)

    if err == nil {
        fmt.Println(number, "is number")
    } else {
        fmt.Println(input, "is not number")
        fmt.Println(err.Error())
    }
}
```

Jalankan program, maka muncul tulisan `"Type some number: "`. Ketik angka bebas, jika sudah maka enter.

Statement `fmt.Scanln(&input)` dipergunakan untuk men-capture inputan yang diketik oleh user sebelum dia menekan enter, lalu menyimpannya sebagai string ke variabel `input`.

Selanjutnya variabel tersebut dikonversi ke tipe numerik menggunakan `strconv.Atoi()`. Fungsi tersebut mengembalikan 2 data, ditampung oleh `number` dan `err`.

Data pertama (`number`) berisi hasil konversi. Dan data kedua `err`, berisi informasi errornya (jika memang terjadi error ketika proses konversi).

Setelah itu dilakukan pengecekkan, ketika tidak ada error, `number` ditampilkan. Dan jika ada error, `input` ditampilkan beserta pesan errornya.

Pesan error bisa didapat dari method `Error()` milik tipe `error`.

1.2.2.2 Membuat Custom Error

Selain memanfaatkan error hasil kembalian suatu fungsi internal yang tersedia, kita juga bisa membuat objek error sendiri dengan menggunakan fungsi `errors.New()` (harus import package `errors` terlebih dahulu).

Pada contoh berikut ditunjukan bagaimana cara membuat custom error. Pertama siapkan fungsi dengan nama `validate()`, yang nantinya digunakan untuk pengecekan input, apakah inputan kosong atau tidak. Ketika kosong, maka error baru akan dibuat.

```
package main

import (
    "errors"
    "fmt"
    "strings"
)

func validate(input string) (bool, error) {
    if strings.TrimSpace(input) == "" {
        return false, errors.New("cannot be empty")
    }
    return true, nil
}
```

Selanjutnya di fungsi main, buat proses sederhana untuk capture inputan user. Manfaatkan fungsi `validate()` untuk mengecek inputannya.

```
func main() {
    var name string
    fmt.Print("Type your name: ")
    fmt.Scanln(&name)

    if valid, err := validate(name); valid {
        fmt.Println("halo", name)
    } else {
        fmt.Println(err.Error())
    }
}
```

Fungsi `validate()` mengembalikan 2 data. Data pertama adalah nilai `bool` yang menandakan inputan apakah valid atau tidak. Data ke-2 adalah pesan error-nya (jika inputan tidak valid).

Fungsi `strings.TrimSpace()` digunakan untuk menghilangkan karakter spasi sebelum dan sesudah string. Ini dibutuhkan karena user bisa saja menginputkan spasi lalu enter.

Ketika inputan tidak valid, maka error baru dibuat dengan memanfaatkan fungsi `errors.New()`. Selain itu objek error juga bisa dibuat lewat fungsi `fmt.Errorf()`.

1.2.2.3 Penggunaan Panic

Panic digunakan untuk menampilkan *stack trace* error sekaligus menghentikan flow goroutine (karena `main()` juga merupakan goroutine, maka behaviour yang sama juga berlaku). Setelah ada panic, proses akan terhenti, apapun setelah tidak di-eksekusi kecuali proses yang sudah di-defer sebelumnya (akan muncul sebelum panic error).

Panic menampilkan pesan error di console, sama seperti `fmt.Println()`. Informasi error yang ditampilkan adalah stack trace error, jadi sangat mendetail dan heboh.

Kembali ke koding, pada program yang telah kita buat tadi, ubah `fmt.Println()` yang berada di dalam blok kondisi `else` pada fungsi main menjadi `panic()`, lalu tambahkan `fmt.Println()` setelahnya.

```
func main() {
    var name string
    fmt.Print("Type your name: ")
    fmt.Scanln(&name)

    if valid, err := validate(name); valid {
        fmt.Println("halo", name)
    } else {
        panic(err.Error())
        fmt.Println("end")
    }
}
```

1.2.2.4 Penggunaan Recover

Recover berguna untuk meng-handle panic error. Pada saat panic error muncul, recover men-take-over goroutine yang sedang panic (pesan panic tidak akan muncul).

Ok, mari kita modif sedikit fungsi di-atas untuk mempraktekkan bagaimana cara penggunaan recover. Tambahkan fungsi `catch()`, dalam fungsi ini terdapat statement `recover()` yang dia akan mengembalikan pesan panic error yang seharusnya muncul.

Untuk menggunakan recover, fungsi/closure/IIFE dimana `recover()` berada harus dieksekusi dengan cara di-defer.

```
func catch() {
    if r := recover(); r != nil {
        fmt.Println("Error occured", r)
    } else {
        fmt.Println("Application running perfectly")
    }
}

func main() {
    defer catch()

    var name string
    fmt.Print("Type your name: ")
    fmt.Scanln(&name)
```

```
    if valid, err := validate(name); valid {
        fmt.Println("halo", name)
    } else {
        panic(err.Error())
        fmt.Println("end")
    }
}
```

1.2.2.5 Pemanfaatan Recover pada IIFE

Contoh penerapan recover pada IIFE:

```
func main() {
    defer func() {
        if r := recover(); r != nil {
            fmt.Println("Panic occured", r)
        } else {
            fmt.Println("Application running perfectly")
        }
    }()
```

Pada kode di atas, bisa dilihat di dalam perulangan terdapat sebuah IIFE untuk recover panic dan juga ada kode untuk men-trigger panic error secara paksa. Ketika panic error terjadi, maka idealnya perulangan terhenti, tetapi pada contoh di atas tidak, dikarenakan operasi dalam perulangan sudah di bungkus dalam IIFE dan seperti yang kita tau sifat panic error adalah menghentikan proses secara paksa dalam scope blok fungsi.

```
    panic("some error happen")
}
```

Dalam real-world development, ada kalanya recover dibutuhkan tidak dalam blok fungsi terluar, tetapi dalam blok fungsi yg lebih spesifik.

Silakan perhatikan contoh kode recover perulangan berikut. Umumnya, jika terjadi panic error, maka proses proses dalam scope blok fungsi akan terjenti, mengakibatkan perulangan juga akan terhenti secara paksa. Pada contoh berikut kita coba terapkan cara handle panic error tanpa menghentikan perulangan itu sendiri.

```
func main() {
    data := []string{"superman", "aquaman", "wonder woman"}
```

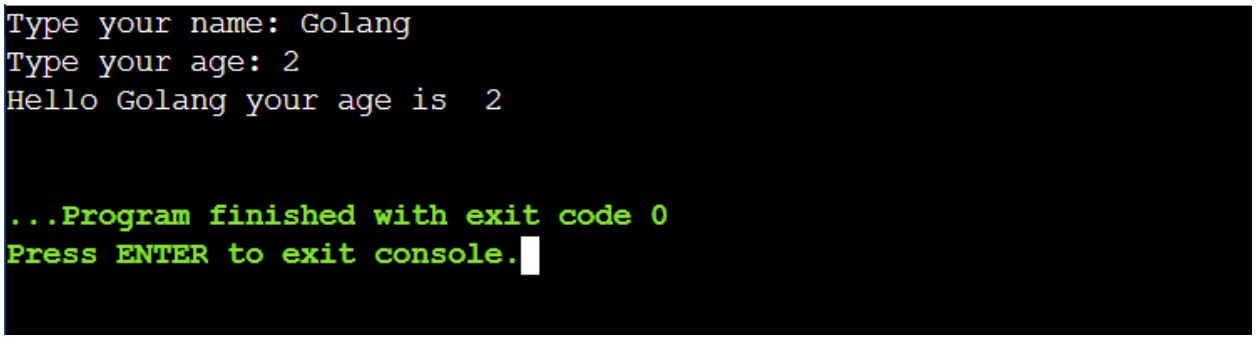
```
    for _, each := range data {

        func() {

            // recover untuk IIFE dalam perulangan
            defer func() {
                if r := recover(); r != nil {
                    fmt.Println("Panic occured on looping", each, "|
message:", r)
                } else {
                    fmt.Println("Application running perfectly")
                }
            }()

            panic("some error happen")
        }()

    }
}
```

1.3 Praktikum

1.3.1 Latihan Praktikum

```
package main

import (
    "errors"
    "fmt"
    "strings"
    "strconv"
)

func validate(name string, age string) (bool, error) {
    if strings.TrimSpace(age) == "" {
        return true, errors.New("Name cannot be empty")
    }

    _, err := strconv.Atoi(name)
    if err != nil {
        return false, errors.New("Age must be number")
    }

    return true, nil
}

func main() {
    var name, age string

    fmt.Print("Type your name: ")
    fmt.Scanln(&name)

    fmt.Print("Type your age: ")
    fmt.Scanln(&ages)

    if valid, err := validate(name); valid {
        fmt.Println("Hello", name, "your age is ", age)
    } else {
        fmt.Println(err.Error())
    }
}
```

Untuk latihan praktikum hari ini, silahkan untuk membenarkan program diatas sehingga berjalan seperti screenshot dibawah ini.



1.3.2 Tugas Praktikum

Coming Soon