MODUL III
KONVERSI TIPE DATA ,CONSTANT VARIABLE & TYPE DECLARATION

Pertemuan : 3
Waktu : 8 x 60 menit (Online)

1.1 Tujuan Modul III

Setelah mahasiswa mempelajari materi ini, diharapkan dapat :

1. Memahami konversi tipe data di golang.
2. Memahami penggunaan constant variable di golang.
3. Memahami penggunaan type declaration di golang.

1.2 Landasan Teori

1.2.1 Konversi Tipe Data

1.2.1.1 Konversi Menggunakan strconv

Package `strconv` berisi banyak fungsi yang sangat membantu kita untuk melakukan konversi. Berikut merupakan beberapa fungsi yang dalam package tersebut.

- **Fungsi `strconv.Atoi()`**

  Fungsi ini digunakan untuk konversi data dari tipe `string` ke `int`. `strconv.Atoi()` menghasilkan 2 buah nilai kembalian, yaitu hasil konversi dan `error` (jika konversi sukses, maka `error` berisi `nil`).

```
package main
import "fmt"
import "strconv"
func main() {
    var str = "124"
    var num, err = strconv.Atoi(str)
    if err == nil {
        fmt.Println(num) // 124
```

```
    }
}
```

- **Fungsi `strconv.Itoa()`**

  Merupakan kebalikan dari `strconv.Atoi`, berguna untuk konversi `int` ke `string`.

```
var num = 124
var str = strconv.Itoa(num)

fmt.Println(str) // "124”
```

- **Fungsi `strconv.ParseInt()`**

  Digunakan untuk konversi `string` berbentuk numerik dengan basis tertentu ke tipe numerik non-desimal dengan lebar data bisa ditentukan.

  Pada contoh berikut, string `"124"` dikonversi ke tipe numerik dengan ketentuan basis yang digunakan `10` dan lebar datanya mengikuti tipe `int64` (lihat parameter ketiga).

```
var str = "124"
var num, err = strconv.ParseInt(str, 10, 64)

if err == nil {
    fmt.Println(num) // 124
}
```

  Contoh lainnya, string `"1010"` dikonversi ke basis 2 (biner) dengan tipe data hasil adalah `int8`.

```
var str = "1010"
var num, err = strconv.ParseInt(str, 2, 8)

if err == nil {
    fmt.Println(num) // 10
}
```

- **Fungsi `strconv.FormatInt()`**

Berguna untuk konversi data numerik `int64` ke `string` dengan basis numerik bisa ditentukan sendiri.

```
var num = int64(24)

var str = strconv.FormatInt(num, 8)


fmt.Println(str) // 30
```

- **Fungsi `strconv.ParseFloat()`**

  Digunakan untuk konversi `string` ke numerik desimal dengan lebar data bisa ditentukan.

  ```
  var str = "24.12"
  var num, err = strconv.ParseFloat(str, 32)

  if err == nil {
      fmt.Println(num) // 24.1200008392334
  }
  ```

  Pada contoh di atas, string `"24.12"` dikonversi ke float dengan lebar tipe data `float32`. Hasil konversi `strconv.ParseFloat` adalah sesuai dengan standar IEEE Standard for Floating-Point Arithmetic.

- **Fungsi `strconv.FormatFloat()`**

  Berguna untuk konversi data bertipe `float64` ke `string` dengan format eksponen, lebar digit desimal, dan lebar tipe data bisa ditentukan.

  ```
  var num = float64(24.12)
  var str = strconv.FormatFloat(num, 'f', 6, 64)

  fmt.Println(str) // 24.120000
  ```

  Pada kode di atas, Data `24.12` yang bertipe `float64` dikonversi ke string dengan format eksponen `f` atau tanpa eksponen, lebar digit desimal 6 digit, dan lebar tipe data `float64`.

- **Fungsi `strconv.ParseBool()`**

  Digunakan untuk konversi `string` ke `bool`.

```
var str = "true"
var bul, err = strconv.ParseBool(str)

if err == nil {
    fmt.Println(bul) // true
}
```

- **Fungsi `strconv.FormatBool()`**

  Digunakan untuk konversi `bool` ke `string`.

```
var bul = true
var str = strconv.FormatBool(bul)

fmt.Println(str) // true
```

1.2.1.2 Konversi Data Menggunakan Teknik Casting

Keyword tipe data bisa digunakan untuk casting, atau konversi antar tipe data. Cara penggunaannya adalah dengan menuliskan tipe data tujuan casting sebagai fungsi, lalu menyisipkan data yang akan dikonversi sebagai parameter fungsi tersebut.

```
var a float64 = float64(24)
fmt.Println(a) // 24

var b int32 = int32(24.00)
fmt.Println(b) // 24
```

1.2.1.3 Casting string ↔ byte

String sebenarnya adalah slice/array `byte`. Di Go sebuah karakter biasa (bukan unicode) direpresentasikan oleh sebuah elemen slice byte. Tiap elemen slice berisi data `int` dengan basis desimal, yang merupakan kode ASCII dari karakter dalam string.

Cara mendapatkan slice byte dari sebuah data string adalah dengan meng-casting-nya ke tipe `[]byte`.

```
var text1 = "halo"
var b = []byte(text1)

fmt.Printf("%d %d %d %d \n", b[0], b[1], b[2], b[3])
// 104 97 108 111
```

Pada contoh di atas, string dalam variabel `text1` dikonversi ke `[]byte`. Tiap elemen slice byte tersebut kemudian ditampilkan satu-per-satu.Contoh berikut ini merupakan kebalikan dari contoh di atas, data bertipe `[]byte` akan dicari bentuk `string`-nya.

```
var byte1 = []byte{104, 97, 108, 111}
var s = string(byte1)

fmt.Printf("%s \n", s)
// halo
```

Pada contoh di-atas, beberapa kode byte dituliskan dalam bentuk slice, ditampung variabel `byte1`. Lalu, nilai variabel tersebut di-cast ke `string`, untuk kemudian ditampilkan.Selain itu, setiap karakter string juga bisa di-casting ke bentuk `int`, hasilnya adalah sama yaitu data byte dalam bentuk numerik basis desimal, dengan ketentuan literal string yang digunakan adalah tanda petik satu (`'`).

Juga berlaku sebaliknya, data numerik jika di-casting ke bentuk string dideteksi sebagai kode ASCII dari karakter yang akan dihasilkan.

```
var c int64 = int64('h')
fmt.Println(c) // 104

var d string = string(104)
fmt.Println(d) // h
```

1.2.1.3 Type Assertions Pada Interface Kosong

Type assertions merupakan teknik untuk mengambil tipe data konkret dari data yang terbungkus dalam `interface{}`. Jadi bisa

disimpulkan bahwa teknik type assertions hanya bisa dilakukan pada data bertipe `interface{}`.

Variabel `data` disiapkan bertipe `map[string]interface{}`, map tersebut berisikan beberapa item dengan tipe data value-nya berbeda satu sama lain, sementara tipe data untuk key-nya sama yaitu `string`.

```
var data = map[string]interface{}{
    "nama":    "john wick",
    "grade":   2,
    "height":  156.5,
    "isMale":  true,
    "hobbies": []string{"eating", "sleeping"},
}

fmt.Println(data["nama"].(string))
fmt.Println(data["grade"].(int))
fmt.Println(data["height"].(float64))
fmt.Println(data["isMale"].(bool))
fmt.Println(data["hobbies"].([]string))
```

Statement `data["nama"].(string)` maksudnya adalah, nilai `data["nama"]` yang bertipe `interface{}` diambil nilai konkritnya dalam bentuk string `string`.

Pada kode di atas, tidak akan terjadi panic error, karena semua operasi type assertion adalah dilakukan menggunakan tipe data yang sudah sesuai dengan tipe data nilai aslinya. Seperti `data["nama"]` yang merupakan `string` pasti bisa di-asertasi ke tipe `string`.

Coba lakukan asertasi ke tipe yang tidak sesuai dengan tipe nilai aslinya, seperti `data["nama"].(int)`, pasti akan men-trigger panic error.

Nah, dari penjelasan diatas, terlihat bahwa kita harus tahu terlebih dahulu apa tipe data asli dari data yang tersimpan dalam interface. Jika misal tidak tahu, maka bisa gunakan teknik di bawah ini untuk pengecekan sukses tidaknya proses asertasi.

Tipe asli data pada variabel `interface{}` bisa diketahui dengan cara meng-casting ke tipe `type`, namun casting ini hanya bisa dilakukan pada `switch`.

```
for _, val := range data {
    switch val.(type) {
    case string:
        fmt.Println(val.(string))
    case int:
        fmt.Println(val.(int))
    case float64:
        fmt.Println(val.(float64))
    case bool:
        fmt.Println(val.(bool))
    case []string:
        fmt.Println(val.([]string))
    default:
        fmt.Println(val.(int))
    }
}
```

Kombinasi `switch` - `case` bisa dimanfaatkan untuk deteksi tipe konkret data yang bertipe `interface{}`, contoh penerapannya seperti pada kode di atas.

1.2.2 Constant Variable

Konstanta adalah jenis variabel yang nilainya tidak bisa diubah. Inisialisasi nilai hanya dilakukan sekali di awal, setelah itu variabel tidak bisa diubah nilainya.

1.2.2.1 Penggunaan Konstanta

Data seperti **pi** (22/7), kecepatan cahaya (299.792.458 m/s), adalah contoh data yang tepat jika dideklarasikan sebagai konstanta daripada variabel, karena nilainya sudah pasti dan tidak berubah.

Cara penerapan konstanta sama seperti deklarasi variabel biasa, selebihnya tinggal ganti keyword `var` dengan `const`.

```
const firstName string = "john"

fmt.Print("halo ", firstName, "!\n")
```

Teknik type inference bisa diterapkan pada konstanta, caranya yaitu cukup dengan menghilangkan tipe data pada saat deklarasi.

```
const lastName = "wick"

fmt.Print("nice to meet you ", lastName, "!\n")
```

1.2.3 Type Declaration

Type declaration mengikat pengidentifikasi, *nama tipe* , ke tipe . Type declaration datang dalam dua bentuk: alias declarations dan type definitions.

```
TypeDecl = "type" ( TypeSpec | "(" { TypeSpec ";" } ")" ) .

TypeSpec = AliasDecl | TypeDef .
```

1.2.3.1 Alias Declarations

Alias declaration mengikat pengidentifikasi ke tipe yang diberikan.

```
AliasDecl = identifier "=" Type .
```

Dalam lingkup pengidentifikasi, ini berfungsi sebagai alias untuk tipe.

```
type (
	nodeList = []*Node
	Polar    = polar
)
```

1.2.3.2 Type Definition

Type definition membuat tipe baru yang berbeda dengan tipe dan operasi dasar yang sama dengan tipe yang diberikan, dan mengikat pengidentifikasi ke dalamnya.

```
TypeDef = identifier Type .
```

Tipe baru ini bisa disebut tipe yang ditentukan. Ini berbeda dari jenis lainnya, termasuk jenis pembuatannya.

```
type (
     Point struct{ x, y float64 }
     polar Point
)

type TreeNode struct {
     left, right *TreeNode
     value *Comparable
}

type Block interface {
     BlockSize() int
     Encrypt(src, dst []byte)
     Decrypt(src, dst []byte)
}
```

Jenis yang ditentukan mungkin memiliki metode yang terkait dengannya. Itu tidak mewarisi metode apa pun yang terikat pada tipe yang diberikan, tetapi kumpulan metode dari tipe antarmuka atau elemen dari tipe komposit tetap tidak berubah:

```
type Mutex struct
func (m *Mutex) Lock()
func (m *Mutex) Unlock()

type NewMutex Mutex

type PtrMutex *Mutex

type PrintableMutex struct {
     Mutex
```

```
}

type MyBlock Block
```

Type definitions dapat digunakan untuk mendefinisikan tipe boolean, numerik, atau string yang berbeda dan mengaitkan metode dengannya:

```
type TimeZone int
const (
     EST TimeZone = -(5 + iota)
     CST
     MST
     PST
)
func (tz TimeZone) String() string {
     return fmt.Sprintf("GMT%+dh", tz)
}
```

1.3 Praktikum

1.3.1 Latihan Praktikum

1. Research Operasi Matematika & Perbandingan di Golang
2. Research if Conditional dan looping di Golang

1.3.2 Tugas Praktikum

1. Buat sebuah program untuk mengetahui jumlah karakter yang di input dan karakter yang sama.
2. Buat sebuah program Konversi suhu ruangan dari celcius ke fahrenheit dan kelvin.
3. Buat sebuah program deret ganjil dan genap dengan jumlah deret harus sesuai dengan inputan user.
4. Buat program input output berupa konversi dari integer(numerik) menjadi format currency (Rupiah).

5. Buat program untuk menghitung Luas dan keliling dari persegi panjang.Untuk Panjang dan Lebar nya sendiri itu di dapat dari inputan user