

# Variabel, Tipe Data, Konversi

2

## Data, Konstanta, Defer, dan Operator

### Pada Golang

#### Objektif :

- Mahasiswa mampu memahami dan menggunakan variabel di Golang
- Mahasiswa mampu memahami dan menggunakan tipe data yang sesuai dengan penerapan pada Golang.
- Mahasiswa mampu memahami jenis-jenis dan menggunakan operator pada Golang.
- Mahasiswa mampu menggunakan konversi tipe yang sesuai dengan data pada Golang.
- Mahasiswa mampu memahami dan menggunakan konsep Defer pada Golang.

---

### Pendahuluan

Di dalam modul pendahuluan ini dijelaskan secara umum dan khusus mengenai variabel yang dapat dideklarasikan di dalam Golang. Jenis-jenis tipe data yang terdapat pada Golang dan cara menggunakannya dalam penerapan di dalam Golang. Operator yang digunakan tidak hanya sebagai symbol, tetapi juga dalam hitungan matematis dalam pengolahan data, dan data tersebut dapat dikonversi menjadi tipe data yang disesuaikan dengan kebutuhan pengguna. Penggunaan konstanta yang digunakan di dalam Golang dan mahasiswa dapat memahami perbedaan dengan variabel, serta konsep Defer yang dapat diterapkan dalam kode program Golang sebagai akhir dari perintah atau pernyataan di dalam Golang.

#### 2.1. VARIABEL

Variabel merupakan tempat menyimpan atau menampung sebuah nilai data. Variabel di dalam Golang diadopsi menjadi dua jenis penulisan, yaitu dapat dituliskan nama variabel diikuti dengan tipe data dan ada yang tidak. Golang memiliki beberapa jenis deklarasi variabel, yaitu deklarasi variabel dengan tipe

data, variabel tanpa tipe data, menggunakan *keyword* **var**, multi variabel, *underscore*, menggunakan *keyword* **new**, dan menggunakan *keyword* **make**.

### 2.1.1. Deklarasi Variabel Dengan Tipe Data

Pendeklarasian variabel dengan jenis ini adalah sebuah variabel harus diikuti atau disertai dengan jenis tipe data tertentu. Tipe data ini akan menentukan sebuah nilai data yang tersimpan di dalam sebuah variabel. Istilah tipe data yang digunakan dalam suatu variabel ini disebut dengan metode *manifest typing*.

Bentuk Umum:

```
var nama_variabel tipe_data
```

Adapun contoh dalam penerapan jenis deklarasi variabel ini sebagai berikut:

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var nama_depan, nama_belakang string
7     fmt.Print("Masukkan Nama Depan Anda: ")
8     fmt.Scan(&nama_depan)
9     fmt.Print("Masukkan Nama Belakang Anda: ")
10    fmt.Scan(&nama_belakang)
11    fmt.Println("Nama Lengkap Anda: " + nama_depan + " " + nama_belakang)
12 }
```

Gambar 2.1 Contoh Penerapan Variabel Lokal

Kata kunci **var** digunakan sebagai awal dalam mendeklarasi sebuah variabel dan diikuti dengan nama variabel, dan tipe data yang digunakan. Variabel yang digunakan yaitu **nama\_depan** dan **nama\_belakang** yang sudah dideklarasikan pada baris ke 6. Berikut ini hasil keluaran dari kode program gambar 2.1.

```
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh1.go
Masukkan Nama Depan Anda: Jodi
Masukkan Nama Belakang Anda: Bernadi
Nama Lengkap Anda: Jodi Bernadi
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.2 Hasil Keluaran Program Deklarasi Dengan Tipe Data

### 2.1.2. Deklarasi Variabel Tanpa Tipe Data

Dalam deklarasi jenis ini berlawanan dengan deklarasi tipe data. Jenis deklarasi ini tidak harus menggunakan tipe data yang ditulis setelah nama variabel. Deklarasi jenis ini tetap menggunakan tipe data yang dimana tipe datanya ditentukan sendiri oleh tipe data nilainya, jenis deklarasi ini disebut dengan metode *type inference*.

Bentuk Umum:

```
nama_variabel := nilai_data
```

Berikut ini contoh penggunaan deklarasi variabel tanpa tipe data:

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var nama_depan string
7     nama_belakang := "Bernadi"
8     fmt.Print("Masukkan Nama Depan Anda: ")
9     fmt.Scan(&nama_depan)
10    fmt.Println("Nama Lengkap Anda: " + nama_depan + " " + nama_belakang)
11 }
```

Gambar 2.3 Contoh Penerapan Deklarasi Variabel Tanpa Tipe Data

Variabel **nama\_belakang** menyimpan nilai data “Bernadi” yang merupakan tipe datanya adalah **String**. Tipe data dari variabel **nama\_belakang** ditentukan sendiri oleh nilai datanya yang menggunakan tanda petik dua (“”). Tanda **:=** hanya digunakan sekali di awal pada saat deklarasi saja yang menandakan bahwa variabel tersebut tidak diikuti oleh deklarasi tipe data. Berikut adalah hasil kelauran dari program gambar 2.3, seperti pada gambar 2.4.

```
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh2.go
Masukkan Nama Depan Anda: Jodi
Nama Lengkap Anda: Jodi Bernadi
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.4 Hasil Keluaran Program Deklarasi Variabel Tanpa Tipe Data

### 2.1.3. Deklarasi Variabel Menggunakan Keyword Var

Menggunakan kata kunci **var** dapat dilakukan dalam mendeklarasikan suatu variabel. Bentuk umum untuk menggunakan deklarsi variabel ini adalah:

```
var nama_variabel tipe_data      var  
  
nama_variabel tipe_data = nilai_data
```

Dapat diperhatikan bahwa kata kunci **var** dapat digunakan dengan cara 2 deklarasi yang berbeda. Jika menggunakan kata kunci **var** dan terdapat nilai data, maka menggunakan operator (**=**), bukan (**:=**).

```
1  package main  
2  
3  import "fmt"  
4  
5  func main() {  
6      var nama_depan string  
7      var nama_belakang = "Bernadi"  
8      fmt.Print("Masukkan Nama Depan Anda: ")  
9      fmt.Scan(&nama_depan)  
10     fmt.Println("Nama Lengkap Anda: " + nama_depan + " " + nama_belakang)  
11 }
```

Gambar 2.5 Contoh Penerapan Deklarasi Variabel Keyword **var**

Hasil keluaran dari kode program pada gambar 2.5 ketika masukkan untuk **nama\_depan** adalah “Budi”, yaitu:

```
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh3.go  
Masukkan Nama Depan Anda: Budi  
Nama Lengkap Anda: Budi Bernadi  
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.6 Hasil Keluaran Program Deklarasi Variabel Keyword **var**

#### 2.1.4. Deklarasi Variabel Multi Variabel

Deklarasi ini digunakan untuk mendukung Golang dalam mendeklarasikan variabel secara bersamaan. Deklarasi ini dapat dituliskan dalam satu baris perintah deklarasi variabel dan menggunakan tanda koma ( , ) sebagai pembatas.

Berikut ini bentuk umum dari cara deklarasi variabel jenis ini:

Bentuk umum 1:

```
var nama_var1, nama_var2, nama_varN tipe_data nama_var1,  
  
nama_var2, nama_varN = nilai1, nilai2, nilaiN
```

Bentuk umum 2: `var nama_var1, nama_var2, nama_varN tipe_data = nilai1, nilai2, nilaiN`

Bentuk umum 3:

`nama_var1, nama_var2, nama_varN := nilai1, nilai2, nilaiN`

Contoh program berikut ini mendeklarasikan dalam 3 bentuk umum yang sudah dideklarasikan.

Program dapat dilihat pada gambar 2.7 berikut ini:

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var N1, N2, N3 string
7     N1, N2, N3 = "Saya", "Cinta", "Indonesia"
8
9     var N4, N5, N6 float32 = 1, 2.77, 3
10
11     N7, N8, N9 := "Hello", true, 1
12
13     fmt.Println(N1, N2, N3)
14     fmt.Println(N4, N5, N6)
15     fmt.Println(N7, N8, N9)
16 }
```

Gambar 2.7 Contoh Penerapan Deklarasi Multi Variabel

Hasil keluaran dari kode program pada gambar 2.7, yaitu:

```
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh4.go
Saya Cinta Indonesia
1 2.77 3
Hello true 1
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.8 Hasil Keluaran Program Deklarasi Multi Variabel

### 2.1.5. Deklarasi Variabel Underscore

Deklarasi variabel jenis ini digunakan sebagai predefined variabel yang dapat digunakan untuk nilai yang tidak digunakan. Golang merupakan Bahasa pemrograman yang memiliki aturan unik bahwa tidak boleh ada satupun yang tidak digunakan. Semua variabel yang dideklarasikan harus digunakan, jika tidak maka program akan gagal dikompilasi. Deklarasi ini menggunakan tanda ( = ) untuk mengisi nilai data dan tidak perlu menggunakan ( := ) karena sifatnya yang predefined. Variabel jenis ini sering dimanfaatkan untuk menampung nilai balik fungsi yang tidak digunakan dan isi variabel ini tidak dapat ditampilkan.

Berikut ini contoh penerapan variabel Underscore ( \_ ):

```
_ = "Saya belajar Golang"
_ = "Yuk Belajar Pemrograman"
nama1, _ := "Golang", "itu Mudah"
```

Gambar 2.9 Contoh Penerapan Deklarasi Variabel Underscore

### 2.1.6. Deklarasi Variabel Menggunakan Keyword New

Deklarasi variabel dengan menggunakan kata kunci **new** digunakan untuk mencetak data **pointer** dengan tipe data tertentu dan nilai data dari default-nya menyesuaikan tipe datanya. Berikut ini contoh penerapan dalam variabel jenis ini:

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     nama := new(string)
7     fmt.Println(nama)
8     fmt.Println(*nama)
9 }
```

Gambar 2.10 Contoh Penerapan Deklarasi Variabel Menggunakan Keyword new

```
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh5.go
0xc4200781c0
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.11 Hasil Keluaran Program Deklarasi Variabel Menggunakan Keyword new

Variabel nama menampung data bertipe **pointer string**. Jika ditampilkan yang tampil adalah nilai alamat memori nilai tersebut (notasi heksadesimal) dan untuk menampilkan nilai aslinya, variabel tersebut harus menggunakan tanda asterisk ( \* ) karena harus di-**dereference** terlebih dahulu. *(dipelajari lebih lanjut pada bagian pointer dan dereference)*



### 2.1.7. Deklarasi Variabel Menggunakan Keyword Make

Deklarasi variabel menggunakan kata kunci **make** hanya bisa digunakan untuk pembuatan beberapa jenis variabel saja, yaitu :

- Channel
- Slice
- Map

Jenis ini akan dibahas pada bagian khusus di pertemuan selanjutnya.

## 2.2. TIPE DATA

Secara umum dalam Bahasa pemrograman untuk merepresentasikan dan mengolah data membutuhkan tipe data yang sesuai dengan masukkan data ke dalam sistem. Golang memiliki beberapa tipe data yang dapat dikelompokkan menjadi **tipe data numerik (decimal dan non-desimal)**, **string**, dan **Boolean**.

### 2.2.1. Tipe Data Numerik Non-Desimal

Tipe data numerik non-desimal merupakan tipe data yang digunakan untuk nilai data bilangan bulat. Tipe data numerik non-desimal juga disebut dengan *non floating point*. Di Golang, secara umum terdapat dua tipe data yang perlu diketahui, yaitu:

- **uint**, adalah tipe data untuk bilangan positif (bilangan cacah), dan
- **int**, adalah tipe data untuk bilangan bulat (bilangan negative dan positif)

Kedua tipe data di atas terbagi menjadi beberapa jenis tipe data lagi yang dibagi berdasarkan jangkauan cakupan nilainya. Jenis tipe data numerik non-decimal sebagai berikut:

Tabel 2.1 Tipe Data Numerik Non-Desimal

No	Tipe Data	Cakupan Nilai Terendah	Cakupan Nilai Tertinggi
1	uint8	0	255
2	uint16	0	65535
3	uint32	0	4294967295
4	uint64	0	18446744073709551615
5	uint	sama dengan <b>uint32</b> atau <b>uint64</b> (tergantung nilai)	
6	byte	sama dengan <b>uint8</b>	

7	int8	-128	127
8	int16	-32768	32767
9	int32	-2147483648	2147483647
10	int64	-9223372036854775808	9223372036854775807
11	int	sama dengan <b>int32</b> atau <b>int64</b> (tergantung nilai)	
12	rune	sama dengan <b>int32</b>	

Berikut ini contoh penggunaan tipe data numerik non-desimal:

```

1  package main
2
3  import "fmt"
4
5  func main() {
6      var bilangan_positif uint = 70
7      var bilangan_negatif = -777
8
9      fmt.Printf("Bilangan positif statis = %d\n", bilangan_positif)
10     fmt.Printf("Bilangan negatif statis = %d\n", bilangan_negatif)
11 }

```

Gambar 2.11 Penerapan Tipe Data Numerik Non-Desimal

Ketika dijalankan kode program pada gambar 2.11, maka akan tampil seperti berikut:

```

[Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh6.go
Bilangan positif statis = 70
Bilangan negatif statis = -777
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$

```

Gambar 2.12 Hasil Keluaran Program Tipe Data Numerik Non-Desimal

Pada tipe data variabel **bilangan\_positif** sudah dideklarasikan dengan tipe data **uint** dengan nilai awal 70, sedangkan tipe data variabel **bilangan\_negatif** menyesuaikan nilai datanya, yaitu -777 yang termasuk ke dalam tipe data **int16**. Perintah **%d** pada **fmt.Printf()** digunakan untuk memformat keluaran data numerik.



### 2.2.2. Tipe Data Numerik Desimal

Tipe data numerik decimal digunakan untuk bilangan yang memiliki angka dibelakang koma, atau bilangan pecahan. Tipe data jenis ini ada dua, yaitu **float32** dan **float64**. Perbedaan kedua tipe data ini terletak pada jangkauan nilai. Jangkauan nilai ini merujuk ke spesifikasi yang telah diatur dan dikeluarkan oleh **IEEE-754 32-bit floating-point numbers**.

Berikut ini contoh penggunaan dalam tipe data numerik decimal:

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var bilangandesimalpecahan = 7.7
7
8     fmt.Printf("Bilangan desimal pecahan: %f\n", bilangandesimalpecahan)
9     fmt.Printf("Bilangan desimal pecahan: %.3f\n", bilangandesimalpecahan)
10 }
```

Gambar 2.13 Penerapan Tipe Data Numerik Desimal

Ketika dijalankan kode program pada gambar 2.13, maka hasil keluarannya adalah:

```
[Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh7.go
Bilangan desimal pecahan: 7.700000
Bilangan desimal pecahan: 7.700
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.14 Hasil Keluaran Program Tipe Data Numerik Desimal

Perintah **%f** digunakan untuk mengkonversi data numerik decimal menjadi tipe data string. Digit decimal yang dihasilkan 6 digit angka. Untuk mengontrol jumlah bilangan angka dibelakang koma dapat menggunakan perintah **%.nf**, n dapat diganti menjadi bilangan, sesuai dengan kebutuhan pengguna.

### 2.2.3. Tipe Data bool (Boolean)

Tipe data Boolean ini dideklarasikan dengan kata kunci **bool**. Tipe data ini hanya memiliki dua nilai, yaitu **true** dan **false**. Tipe data ini secara umum digunakan dalam seleksi kondisi dan perulangan.

Berikut ini contoh penggunaan tipe data Boolean:

```

1  package main
2
3  import "fmt"
4
5  func main() {
6      var uji bool = true
7      fmt.Printf("Hasil dari var uji: %t \n", uji)
8  }

```

Gambar 2.15 Penerapan Tipe Data Boolean

Jika kode program pada gambar 2.15 dijalankan, maka hasil keluaran seperti pada gambar 2.16 berikut:

```

[Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh7.go
Bilangan desimal pecahan: 7.700000
Bilangan desimal pecahan: 7.700
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$

```

Gambar 2.16 Hasil Keluaran Penerapan Tipe Data Boolean

Perintah `%t` digunakan untuk memformat data dengan tipe data **bool** yang digunakan dalam fungsi `fmt.Printf()`.

#### 2.2.4. Tipe Data string

Dalam menggunakan tipe data **string**, nilai data diapit oleh tanda petik dua (") atau tanda *quote*. Selain itu, dapat juga menggunakan tanda *grave accent/backticks* ( ` ), tanda ini terletak di sebelah kiri tombol 1. Kelebihan string yang dideklarasikan dengan menggunakan *grave accent* adalah membuat semua karakter di dalamnya **termasuk ke dalam string**.

Berikut contoh penerapannya:

```

1  package main
2
3  import "fmt"
4
5  func main() {
6      var pesan1 string = "hello"
7      var pesan2 = `Saya senang belajar.
8      Golang`
9
10     fmt.Println(pesan1, pesan2)
11 }

```

Gambar 2.17 Penerapan Tipe Data String

Berikut ini hasil keluaran dari penerapan string dengan tanda *quote* dan *grave accent*.

```
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh9.go
hello Saya senang belajar.
Golang
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.18 Hasil Keluaran Tipe Data String

### 2.2.5. Nilai nil dan Nilai Default Tipe Data

**nil** bukan termasuk ke dalam tipe data, melainkan sebuah nilai. Variabel yang isi nilainya **nil**, berarti variabel tersebut memiliki nilai kosong.

Semua tipe data yang dibahas di atas memiliki nilai **default**. Artinya meskipun variabel dideklarasikan dengan tanpa nilai awal, aka nada nilai **default**-nya.

Tabel 2.2 Nilai Default Tipe Data

Tipe Data	Nilai Default
string	“ “ (string kosong)
bool	false
numerik non-desimal	0
numerik desimal	0.0

**nil** tidak dapat digunakan pada tipe data yang sudah dibahas di atas, karena seuanya memiliki nilai default pada saat deklarasi. Ada beberapa tipe data yang bisa di-set nilainya dengan **nil**, diantaranya **pointer**, **tipe data fungsi**, **slice**, **map**, **channel**, **interface kosong** atau **interface{}**.

## 2.3. KONVERSI DATA

Konversi data digunakan untuk mengubah tipe data dari suatu variabel yang mengandung nilai data menjadi tipe data yang dibutuhkan atau disesuaikan dengan program.

Berikut dijelaskan macam-macam konversi yang terdapat pada Golang:

### 2.3.1 Konversi Data Menggunakan strconv

**strconv** mengandung banyak fungsi yang sangat membantu untuk keperluan konversi data.

Berikut ini merupakan beberapa fungsi dalam *package* tersebut yang bisa dimanfaatkan.

**a. Fungsi strconv.Atoi()**

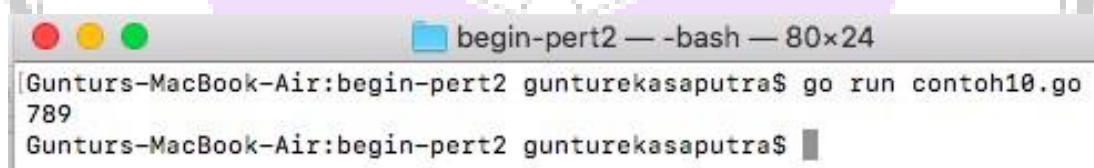
Fungsi ini dapat digunakan untuk mengkonversi data dari tipe **string** ke **int**. Mengembalikan 2 buah nilai balik, yaitu hasil konversi dan **error** (jika konversi sukses, maka **error** akan berisi **nil**).

Berikut ini contoh dalam penerapan fungsi strconv.Atoi():

```
1  package main
2
3  import "fmt"
4  import "strconv"
5
6  func main() {
7      var str = "789"
8      var num, err = strconv.Atoi(str)
9
10     if err == nil {
11         fmt.Println(num)
12     }
13 }
```

Gambar 2.19 Penerapan Fungsi strconv.Atoi()

Setelah dijalankan kode program pada gambar 2.19, maka akan menghasilkan seperti pada gambar 2.20.



```
begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh10.go
789
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.20 Hasil Keluaran Penerapan Fungsi strconv.Atoi()

**b. Fungsi strconv.Itoa()**

Fungsi ini merupakan kebalikan dari **strconv.Atoi** yang digunakan untuk mengkonversi **int** ke **string**.

Berikut ini contoh penerapan dari fungsi **strconv.Itoa()**:

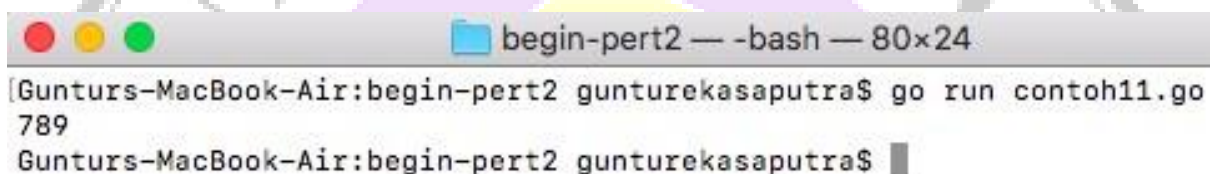
```

1  package main
2
3  import "fmt"
4  import "strconv"
5
6  func main() {
7      var bil = 789
8      var str = strconv.Itoa(bil)
9
10     fmt.Println(str)
11 }

```

Gambar 2.21 Penerapan Fungsi strconv.Itoa()

Jika dijalankan kode program pada gambar 2.21, maka akan menghasilkan hasil keluaran seperti pada gambar 2.22 berikut ini:



```

begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh11.go
789
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$

```

Gambar 2.22 Hasil Keluaran Penerapan Fungsi strconv.Itoa()

### c. Fungsi strconv.ParseInt()

Fungsi ini digunakan untuk mengkonversi tipe data **string** yang berbentuk numerik dengan basis tertentu ke tipe numerik non-desimal dengan lebar data bisa ditentukan.

Pada contoh seperti gambar 2.23, nilai data “789” dengan tipe string yang ditentukan basis numeriknya 10, akan dikonversi ke jenis tipe data **int64**.

Berikut ini penerapan pada fungsi strconv.ParseInt() pada gambar 2.23:

```

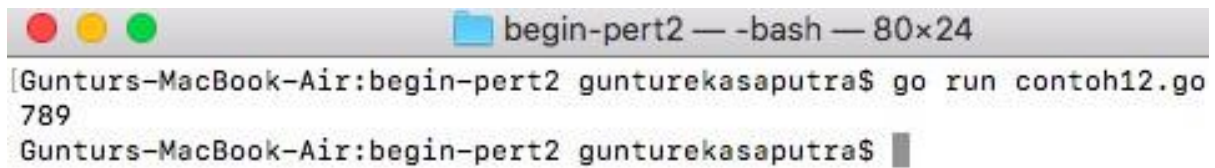
1  package main
2
3  import "fmt"
4  import "strconv"
5
6  func main() {
7      var str = "789"
8      var bil, err = strconv.ParseInt(str, 10, 64)
9      if err == nil {
10         fmt.Println(bil)
11     }
12 }

```

Gambar 2.23 Penerapan Fungsi strconv.ParseInt ()



Ketika dijalankan kode program pada gambar 2.23 akan tampak hasil keluaran seperti gambar 2.24.



```
begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh12.go
789
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.24 Hasil Keluaran Penerapan Fungsi `strconv.ParseInt()`

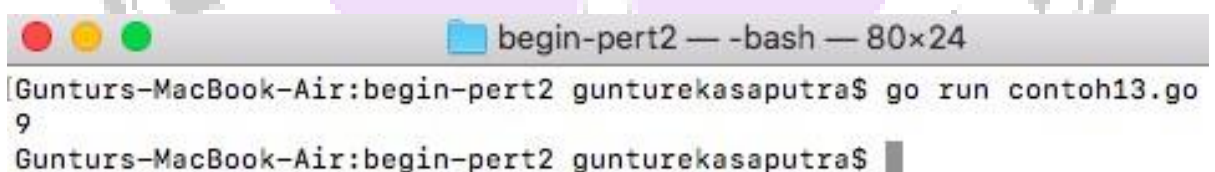
Fungsi ini dapat juga digunakan untuk mengkonversi tipe data dari string “1010” ditentukan basis numeriknya 2 (biner), akan di konversi ke jenis tipe data **int8**.



```
1 package main
2
3 import "fmt"
4 import "strconv"
5
6 func main() {
7     var str = "1001"
8     var bil, err = strconv.ParseInt(str, 2, 8)
9     if err == nil {
10         fmt.Println(bil)
11     }
12 }
```

Gambar 2.25 Penerapan Lain dari Fungsi `strconv.ParseInt()`

Berikut ini hasil kode program gambar 2.25 yang dijalankan pada gambar 2.26.



```
begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh13.go
9
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.26 Hasil Keluaran Penerapan Lain dari Fungsi `strconv.ParseInt()`

#### d. Fungsi `strconv.FormatInt()`

Fungsi ini digunakan untuk mengkonversi data numerik dengan tipe data **int64** ke **string** dengan menggunakan basis numerik bisa ditentukan sendiri.

Berikut ini contoh penerapan dalam fungsi `strconv.FormatInt()` pada gambar 2.27.



```

1  package main
2
3  import "fmt"
4  import "strconv"
5
6  func main() {
7      var bil = int64(17)
8      var str = strconv.FormatInt(bil, 8)
9      fmt.Println(str)
10 }

```

Gambar 2.27 Penerapan Fungsi strconv.FormatInt()

Ketika dijalankan kode program gambar 2.27, maka akan menghasilkan keluaran pada gambar 2.28.

```

[Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh14.go
21
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$

```

Gambar 2.28 Hasil Keluaran Fungsi strconv.FormatInt()

e. **Fungsi strconv.ParseFloat()**

Fungsi dapat digunakan untuk mengkonversi **string** ke **numerik decimal** dengan lebar data yang bisa ditentukan.

Berikut ini contoh penerapan fungsi strconv.ParseFloat() pada gambar 2.29:

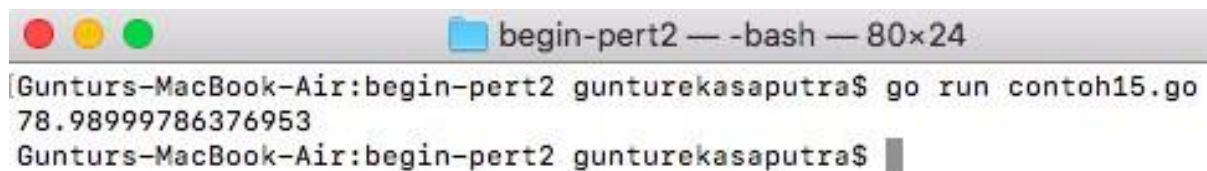
```

1  package main
2
3  import "fmt"
4  import "strconv"
5
6  func main() {
7      var str = "78.99"
8      var bil, err = strconv.ParseFloat(str, 32)
9      if err == nil {
10         fmt.Println(bil)
11     }
12 }

```

Gambar 2.29 Penerapan Fungsi strconv.ParseFloat()

String "78.99" akan dikonversi ke tipe data float dengan lebar **float32**. Hasil konversi dengan fungsi ini disesuaikan dengan standar IEEE Standard for Floating-Point Arithmetic. Berikut ini hasil keluaran dari kode program yang dijalankan pada gambar 2.29:



```
begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh15.go
78.98999786376953
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.30 Hasil Keluaran Fungsi strconv.ParseFloat()

**f. Fungsi strconv.FormatFloat()**

Fungsi ini digunakan untuk mengkonversi data bertipe **float64** ke **string** dengan format eksponen, lebar digit desimal, dan lebar tipe data bisa ditentukan.

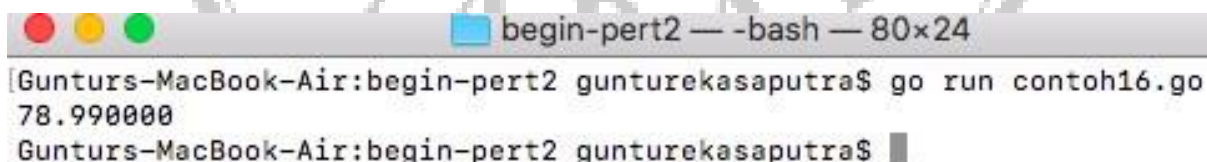
Berikut ini contoh konversi tipe data seperti pada kode program gambar 2.31:



```
1 package main
2
3 import "fmt"
4 import "strconv"
5
6 func main() {
7     var bil = float64(78.99)
8     var str = strconv.FormatFloat(bil, 'f', 6, 64)
9     fmt.Println(str)
10 }
```

Gambar 2.31 Penerapan Fungsi strconv.FormatFloat()

Kode program pada gambar 2.31, nilai data pada variabel **bil** adalah “78.99” yang bertipe **float64**, kemudian dikonversi ke dalam tipe data string dengan format eksponen **f** (tanpa eksponen), jangkauan digit decimal sebanyak 6 digit, dan lebar tipe data **float64**. Berikut ini ketika dijalankan kode program di atas seperti pada gambar 2.32:



```
begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh16.go
78.990000
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.32 Hasil Keluaran Fungsi strconv.FormatFloat()

Format eksponen pada Golang dapat disajikan dengan beberapa cara, seperti pada tabel 2.3.

Tabel 2.3 Format Eksponen Golang

Format Eksponen	Keterangan
b	-ddddp□ddd, a, eksponen biner (basis 2)
e	-d.dddde□dd, a, eksponen desimal (basis 10)
E	-d.ddddE□dd, a, eksponen desimal (basis 10)
f	-ddd.dddd, tanpa eksponen
g	Akan menggunakan format eksponen e untuk eksponen besar dan f untuk lainnya
G	Akan menggunakan format eksponen E untuk eksponen besar dan f untuk lainnya

**g. Fungsi strconv.ParseBool**

Fungsi ini digunakan untuk mengkonversi **string** ke tipe data **bool**. Berikut ini contoh penerapan fungsi strconv.ParseBool seperti pada gambar 2.33:

```

1  package main
2
3  import "fmt"
4  import "strconv"
5
6  func main() {
7      var str = "true"
8      var boolean, err = strconv.ParseBool(str)
9      if err == nil {
10         fmt.Println(boolean)
11     }
12 }

```

Gambar 2.33 Penerapan Fungsi strconv.ParseBool()

Berikut ini hasil keluaran ketika dijalankan kode program pada gambar 2.33, seperti pada gambar 2.34:

```

begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh17.go
true
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$

```

Gambar 2.34 Hasil Keluaran Fungsi strconv.ParseBool()

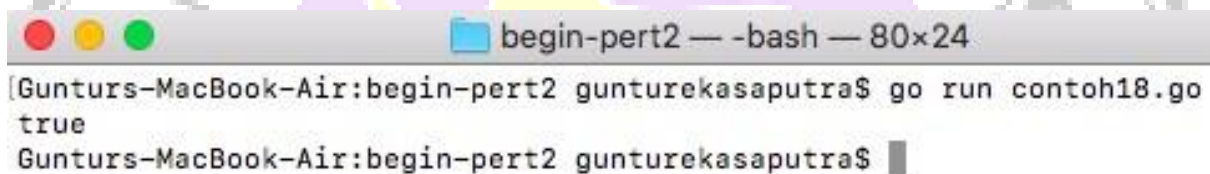
#### h. Fungsi `strconv.FormatBool()`

Fungsi ini digunakan untuk mengkonversi tipe data **bool** ke **string**. Contoh penerapan dengan fungsi ini, seperti pada gambar 2.35:

```
1 package main
2
3 import "fmt"
4 import "strconv"
5
6 func main() {
7     var boolean = true
8     var str = strconv.FormatBool(boolean)
9     fmt.Println(str)
10 }
```

Gambar 2.35 Penerapan Fungsi `strconv.FormatBool()`

Ketika dijalankan kode program pada gambar 2.35, maka hasil keluaran dapat terlihat seperti gambar 2.36.



```
begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh18.go
true
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.36 Hasil Keluaran Fungsi `strconv.FormatBool()`

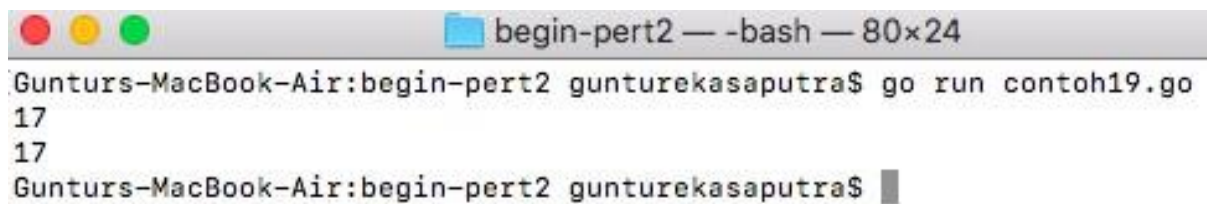
### 2.3.2 Konversi Data Menggunakan Casting

Konversi data dengan menggunakan metode Casting adalah konversi tipe data dengan menggunakan *keyword* atau kata kunci. Penggunaan konversi ini dengan memanggil tipe data sebagai fungsi dan menyisipkan nilai data yang akan dikonversi sebagai parameter.

Berikut ini contoh penerapan secara umum untuk menerapkan konversi data menggunakan Casting seperti pada gambar 2.37:

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var bil1 float64 = float64(17)
7     fmt.Println(bil1)
8
9     var bil2 int32 = int32(17.00)
10    fmt.Println(bil2)
11 }
```

Gambar 2.37 Penerapan Secara Umum Konversi Data Casting



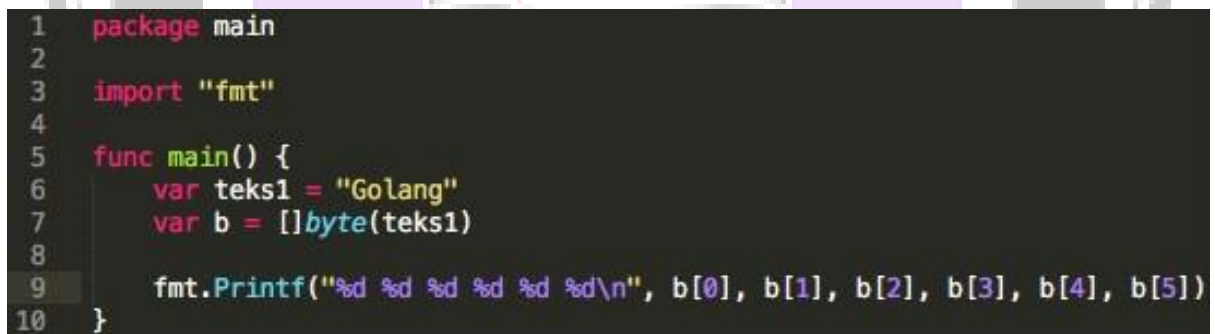
```
begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh19.go
17
17
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.38 Hasil Keluaran Konversi Data Casting

### Casting string ke byte

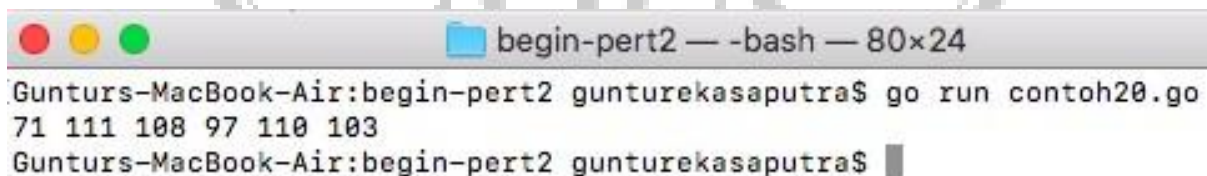
Tipe data string sebenarnya adalah slice/array **byte**. Pada Bahasa pemrograman Golang, sebuah karakter biasa (bukan Unicode) direpresentasikan oleh sebuah elemen slice byte. Nilai slice tersebut adalah data **int** yang (default-nya) berbasis desimal, yang merupakan kode ASCII dari karakter biasa tersebut.

Cara mendapatkan slice byte dari sebuah data string adalah dengan menggunakan Casting ke tipe [ ] **byte**. Tiap elemen **byte** memiliki isi data numerik dengan basis desimal. Berikut ini adalah contoh string dalam variabel **teks1** yang dikonversi ke dalam [ ] **byte**. Tiap elemen slice byte tersebut kemudian ditampilkan satu per satu, seperti pada gambar 2.39 dan hasil dari kode program tersebut tampak pada gambar 2.40 berikut ini:



```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var teks1 = "Golang"
7     var b = []byte(teks1)
8
9     fmt.Printf("%d %d %d %d %d %d\n", b[0], b[1], b[2], b[3], b[4], b[5])
10 }
```

Gambar 2.39 Penerapan String ke Byte



```
begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh20.go
71 111 108 97 110 103
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

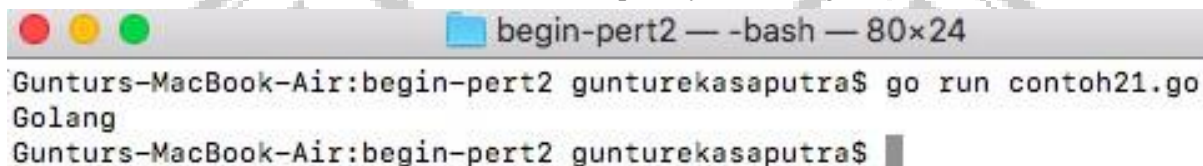
Gambar 2.40 Hasil Keluaran String ke Byte



Contoh berikutnya di bawah ini merupakan kebalikan dari contoh penerapan pada gambar 2.39. Sebuah `[] byte` akan dicari bentuk `string`-nya. Contoh penerapan tersebut tampak pada kode program di gambar 2.41 dan hasil keluarannya tampak pada gambar 2.42.

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var b = []byte{71, 111, 108, 97, 110, 103}
7     var s = string(b)
8
9     fmt.Printf("%s \n", s)
10 }
```

Gambar 2.41 Penerapan Byte ke String



```
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh21.go
Golang
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.42 Hasil Keluaran Byte ke String

Kode program pada gambar 2.41 merupakan beberapa kode byte string yang dituliskan sebagai rangkaian slice, yang ditampung oleh variabel `b`. Kemudian, nilai variabel tersebut dikonversi (cast) ke dalam tipe data `string` untuk kemudian ditampilkan dengan perintah `fmt.Printf("%s \n", s)`.

### 2.3.3 Konversi Data `interface{}` Menggunakan Teknik Type Assertions

Konversi data dengan teknik **type assertions** merupakan teknik casting data `interface{}` ke segala jenis tipe (dengan syarat datatersebut memang bisa di-casting).

Berikut merupakan contoh penerapannya. Disiapkan variabel `nilai_data` bertipe `map[string]interface{}` dengan *value* berbeda-beda tipe datanya.

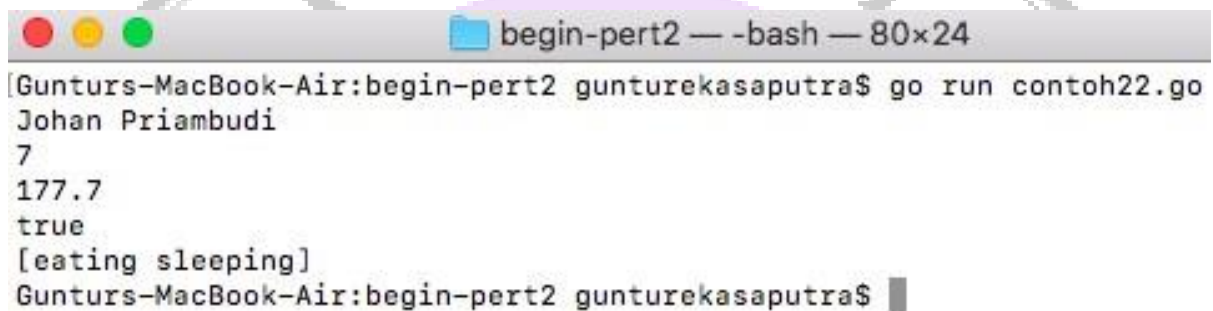


```

1 package main
2
3 import "fmt"
4
5 func main() {
6     var nilai_data = map[string]interface{}{
7         "nama": "Johan Priambudi",
8         "grade": 7,
9         "height": 177.7,
10        "isMale": true,
11        "hobbies": []string{"eating", "sleeping"},
12    }
13
14    fmt.Println(nilai_data["nama"].(string))
15    fmt.Println(nilai_data["grade"].(int))
16    fmt.Println(nilai_data["height"].(float64))
17    fmt.Println(nilai_data["isMale"].(bool))
18    fmt.Println(nilai_data["hobbies"].([]string))
19 }

```

Gambar 2.43 Penerapan Type Assertions



```

begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh22.go
Johan Priambudi
7
177.7
true
[eating sleeping]
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$

```

Gambar 2.44 Hasil Keluaran Type Assertions

Pernyataan `nilai_data["nama"].(string)` memiliki arti bahwa `nilai_data["nama"]` dicasting sebagai **string**.

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

## 2.4. KONSTANTA

**Konstanta** adalah variabel yang memiliki nilai tetap atau tidak bisa diubah. Inisialisasi nilai hanya dilakukan sekali pada awal program, setelah itu data tidak dapat diubah nilainya.

### 2.4.1 Penggunaan Konstanta

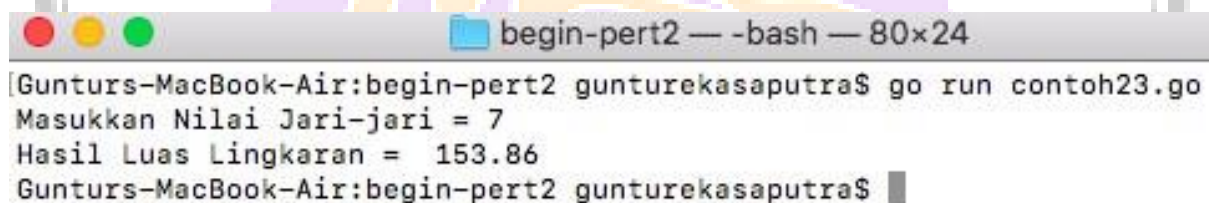
Penggunaan seperti dalam matematika adalah nilai phi (22/7) atau (3.14), kecepatan Chaya (299.792.458 m/s) adalah contoh nilai data yang tepat jika dideklarasikan sebagai konstanta daripada variabel karena nilainya sudah pasti dan tidak berubah.

Cara penerapan constanta sama seperti variabel pada umumnya, hanya dengan menggunakan kata kunci **const**. Contoh:

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     const phi float32 = 3.14
7     var jari, luas float32
8
9     fmt.Print("Masukkan Nilai Jari-jari = ")
10    fmt.Scan(&jari)
11
12    luas = phi * (jari * jari)
13
14    fmt.Println("Hasil Luas Lingkaran = ", luas)
15
16 }
```

Gambar 2.45 Penerapan Konstanta pada Golang

Jika dijalankan kode program pada gambar 2.45, maka akan tampil hasil keluaran seperti pada gambar 2.46.



```
begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh23.go
Masukkan Nilai Jari-jari = 7
Hasil Luas Lingkaran = 153.86
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.46 Hasil Keluaran Kode Program Penerapan Konstanta

#### 2.4.2 Penggunaan Fungsi **fmt.Print()**

Fungsi ini memiliki peran yang sama seperti fungsi **fmt.Println()**, pembedanya fungsi **fmt.Print()** tidak menghasilkan baris baru di akhir hasil keluarannya.

### 2.5. DEFER

**Defer** digunakan untuk mengakhirkan eksekusi sebuah pernyataan, sedangkan **Exit** digunakan untuk menghentikan program. Dua bahasan ini digabungkan agar hubungan antara keduanya dapat dengan lebih mudah dipahami.

### 2.5.1 Penerapan Kata Kunci defer

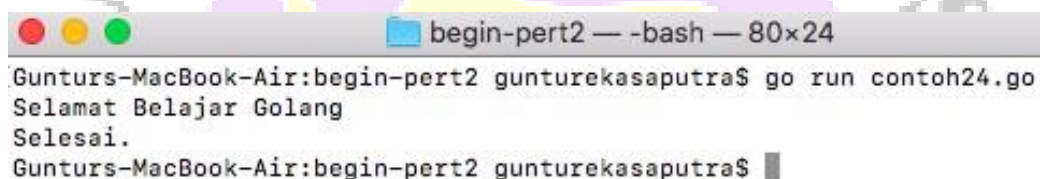
Ketika eksekusi program sudah sampai pada akhir blok fungsi, pernyataan yang di defer baru akan dijalankan.

Defer dapat ditempatkan di mana saja, awal maupun akhir blok program. Berikut contoh penerapan.

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     defer fmt.Println("Selesai.")
7     fmt.Println("Selamat Belajar Golang")
8 }
```

Gambar 2.47 Penerapan Kode Program Defer

Kata kunci **defer** digunakan untuk mengakhirkan pernyataan. Pada kode program gambar 2.47, **fmt.Println("Selesai.")** di-defer, hasilnya string "Selesai."



```
begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh24.go
Selamat Belajar Golang
Selesai.
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.48 Hasil Keluaran Penerapan Defer

### 2.5.2 Penerapan Fungsi os.Exit()

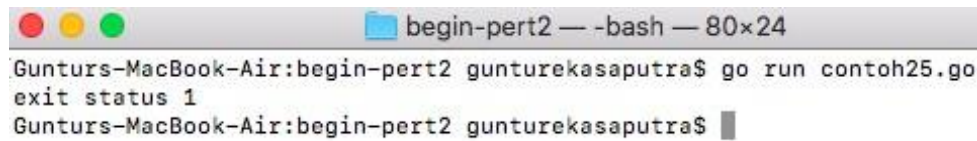
**Exit** digunakan untuk menghentikan program secara paksa pada saat itu juga. Semua pernyataan setelah **Exit** tidak akan dieksekusi, termasuk juga defer.

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.

```
1 package main
2
3 import "fmt"
4 import "os"
5
6 func main() {
7     defer fmt.Println("Selesai.")
8     os.Exit(1)
9     fmt.Println("Selamat Belajar Golang")
10 }
```

Gambar 2.49 Penerapan os.Exit()

Meskipun **defer fmt.Println("Selesai.")** ditempatkan sebelum **os.Exit()**, pernyataan tersebut tidak akan dieksekusi, karena fungsi program tersebut pada tengah kode program, dan dihentikan secara paksa.



```
begin-pert2 — -bash — 80x24
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$ go run contoh25.go
exit status 1
Gunturs-MacBook-Air:begin-pert2 gunturekasaputra$
```

Gambar 2.50 Hasil Keluaran Penerapan os.Exit()

## 2.6 OPERATOR

**Operator** merupakan symbol yang memiliki arti yang secara umum digunakan oleh bahasa pemrograman untuk operasi matematika. Secara umum, operator dibagi menjadi tiga kategori, yaitu Operator Aritmatika, Perbandingan, dan Logika.

### 2.6.1 Operator Aritmatika

Operator aritmatika adalah operator yang digunakan untuk operasi yang sifatnya mengoperasikan perhitungan. Golang mendukung beberapa operator aritmatika standar, seperti pada tabel 2.4.

Tabel 2.4 Operator Aritmatika

Tanda	Keterangan
+	Penjumlahan
-	Pengurangan
*	Perkalian
/	Pembagian
%	Modulus / Sisa hasil pembagian

### 2.6.2 Operator Perbandingan

Operator perbandingan digunakan untuk menentukan kebenaran suatu kondisi. Hasilnya berupa nilai Boolean, **true** atau **false**.

Berikut tabel 2.5 yang berisikan operator perbandingan yang digunakan di Golang.

Tabel 2.5 Operator Perbandingan

Tanda	Keterangan
==	Apakah nilai kiri <b>sama dengan</b> nilai kanan
!=	Apakah nilai kiri <b>tidak sama dengan</b> nilai kanan
<	Apakah nilai kiri <b>lebih kecil daripada</b> nilai kanan
<=	Apakah nilai kiri <b>lebih kecil atau sama dengan</b> nilai kanan
>	Apakah nilai kiri <b>lebih besar dari</b> nilai kanan
>=	Apakah nilai kiri <b>lebih besar atau sama dengan</b> nilai kanan

### 2.6.3 Operator Logika

Operator ini digunakan untuk mencari benar tidaknya kombinasi data bertipe **bool** (yang bisa berupa variabel bertipe **bool**, atau hasil dari operator perbandingan).

Beberapa operator logika standar yang bisa digunakan:

Tabel 2.6 Operator Perbandingan

Tanda	Keterangan
&&	Kiri <b>dan</b> Kanan
	Kiri <b>atau</b> Kanan
!	Negasi / nilai kebalikan