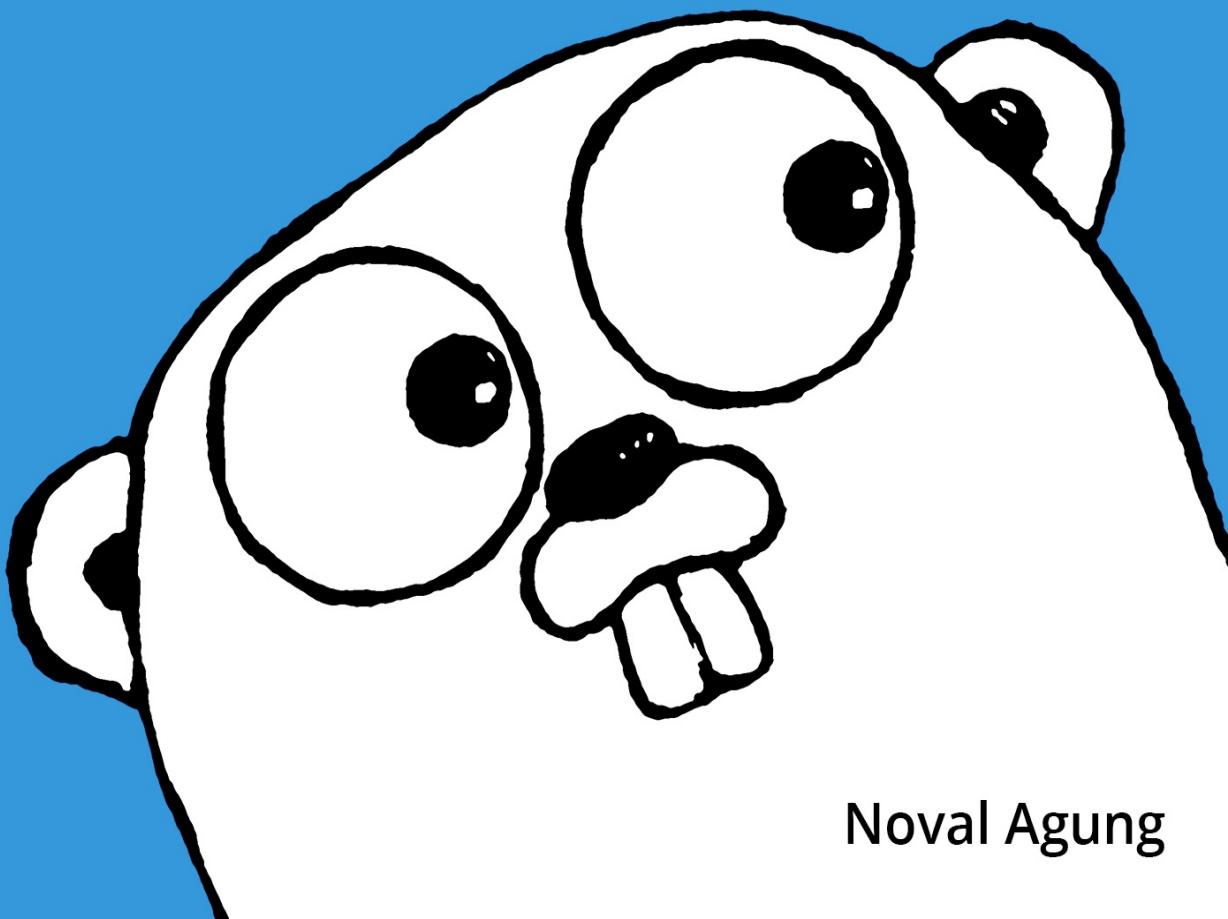


DASAR PEMROGRAMAN

GOLANG



Noval Agung

Table of Contents

Introduction	1.1
1. Berkenalan Dengan Golang	1.2
2. Instalasi Golang	1.3
3. GOPATH Dan Workspace	1.4
4. Instalasi Editor	1.5
5. Command	1.6
6. Program Pertama: Hello World	1.7
7. Komentar	1.8
8. Variabel	1.9
9. Tipe Data	1.10
10. Konstanta	1.11
11. Operator	1.12
12. Seleksi Kondisi	1.13
13. Perulangan	1.14
14. Array	1.15
15. Slice	1.16
16. Map	1.17
17. Fungsi	1.18
18. Fungsi Multiple Return	1.19
19. Fungsi Variadic	1.20
20. Fungsi Closure	1.21
21. Fungsi Sebagai parameter	1.22
22. Pointer	1.23
23. Struct	1.24
24. Method	1.25
25. Property Public & Private	1.26
26. Interface	1.27
27. Interface Kosong	1.28
28. Reflect	1.29
29. Goroutine	1.30

30. Channel	1.31
31. Buffered Channel	1.32
32. Channel - Select	1.33
33. Channel - Range & Close	1.34
34. Channel - Timeout	1.35
35. Defer & Exit	1.36
36. Error, Panic, & Recover	1.37
37. Layout Format String	1.38
38. Time, Parsing Time, & Format Time	1.39
39. Timer	1.40
40. Konversi Antar Tipe Data	1.41
41. Fungsi String	1.42
42. Regex	1.43
43. Encode - Decode Base64	1.44
44. Hash Sha1	1.45
45. Arguments & Flag	1.46
46. Exec	1.47
47. File	1.48
48. Web	1.49
49. URL Parsing	1.50
50. JSON	1.51
51. Web JSON API	1.52
52. HTTP Request	1.53
53. SQL	1.54
54. MongoDB	1.55
55. Unit Test	1.56
56. WaitGroup	1.57
57. Mutex	1.58

Dasar Pemrograman Golang

Golang (atau Go) adalah bahasa pemrograman baru, yang mulai dilirik oleh para developer karena kelebihan-kelebihan yang dimilikinya. Sudah banyak perusahaan besar yang menggunakan bahasa ini untuk produk-produk mereka hingga di level production.

Buku ini merupakan salah satu dari sekian banyak referensi yang bisa digunakan untuk belajar pemrograman Golang. Hal yang disampaikan disini benar-benar dasar, dengan pembelajaran mulai dari 0. Diharapkan dengan adanya buku ini kawan-kawan bisa lebih paham dalam memahami seperti apa sih Golang itu.

Ebook Dasar Pemrograman Golang gratis untuk disebarluaskan secara bebas, selama tidak melanggar aturan lisensi [GNU LGPL 2.1](#).

Source code contoh-contoh program bisa diunduh di <https://github.com/novalagung/dasarpemrogramangolang>. Dianjurkan untuk tidak copy-paste dari source code dalam belajar, usahakan untuk menulis sendiri kode program, agar cepat terbiasa dengan bahasa Golang.

Versi Buku: 1.2018.07.13

Buku ini bisa di-download dalam bentuk PDF, [link download](#). Untuk mendapatkan konten buku yang paling update, silakan baca online, atau download ulang PDF menggunakan link di atas. Konten pada versi terbaru lebih update.

Bantu developer lain untuk mengenal dan belajar Golang, dengan cara [tweet buku ini](#) atau [share ke facebook](#).

Buku ini dibuat oleh **Noval Agung Prayogo**. Untuk pertanyaan, kritik, dan saran, silakan drop email ke caknopal@gmail.com.



1. Berkenalan Dengan Golang

Golang (atau biasa disebut dengan **Go**) adalah bahasa pemrograman baru yang dikembangkan di **Google** oleh **Robert Griesemer**, **Rob Pike**, dan **Ken Thompson** pada tahun 2007 dan mulai diperkenalkan di publik tahun 2009.

Penciptaan bahasa Golang didasari bahasa **C** dan **C++**, oleh karena itu gaya sintaks-nya mirip.

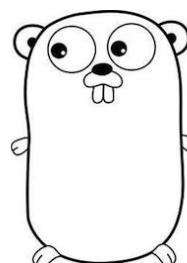
1.1. Kelebihan Golang

Golang memiliki kelebihan dibanding bahasa lainnya, beberapa di antaranya:

- Mendukung konkurensi di level bahasa dengan pengaplikasian cukup mudah
- Mendukung pemrosesan data dengan banyak prosesor dalam waktu yang bersamaan (*parallel processing*)
- Memiliki garbage collector
- Proses kompilasi sangat cepat
- Bukan bahasa pemrograman yang hierarkial, menjadikan developer tidak perlu *ribet* memikirkan segmen OOP-nya
- Package/modul yang disediakan terbilang lengkap. Karena bahasa ini open source, banyak sekali developer yang juga mengembangkan modul-modul lain yang bisa dimanfaatkan

Sudah banyak industri dan perusahaan yg menggunakan Golang sampai level production, termasuk diantaranya adalah Google sendiri, dan tempat dimana saya bekerja ☺

Di buku ini kita akan belajar tentang dasar pemrograman Golang mulai dari 0.



2. Instalasi Golang

Hal pertama yang perlu dilakukan sebelum bisa menggunakan Golang adalah meng-install-nya terlebih dahulu. Panduan instalasi sebenarnya sudah disediakan di situs official Golang <http://golang.org/doc/install#install>.

Disini penulis mencoba meringkas petunjuk instalasi di link tersebut, agar lebih mudah untuk diikuti terutama untuk pembaca yang baru belajar.

Golang yang digunakan adalah versi **1.8.3**. Direkomendasikan menggunakan versi tersebut, atau versi lain minimal **1.4.2** ke atas.

Link untuk download installer golang: <https://golang.org/dl/>. Anda bisa langsung unduh dari URL tersebut lalu lakukan instalasi sendiri, atau bisa mengikuti petunjuk di bab ini.

2.1. Instalasi Golang di Windows

1. Download terlebih dahulu installer-nya. Pilih sesuai jenis bit yang digunakan.
 - 32bit => [go1.8.3.windows-386.msi](#)
 - 64bit => [go1.8.3.windows-amd64.msi](#)
2. Setelah ter-download, jalankan installer, klik **next** sampai proses instalasi selesai. By default jika anda tidak merubah path pada saat instalasi, Golang akan terinstall di terinstal di `c:\go`. Path tersebut secara otomatis didaftarkan dalam **path variable**.
3. Buka **Command Prompt / CMD**, eksekusi perintah untuk mengecek versi Golang.

```
$ go version
```

4. Jika output adalah sama dengan Golang yang ter-install, menandakan instalasi berhasil.

Sering terjadi command `go version` tidak bisa dijalankan meskipun instalasi sukses. Solusinya bisa dengan restart CMD (close CMD, lalu buka kembali). Setelah itu coba jalankan sekali lagi command tersebut.

2.2. Instalasi Golang di Mac OS

Cara termudah instalasi Golang di **Mac OS** adalah menggunakan [homebrew](#).

2. Instalasi Golang

1. Install terlebih dahulu Homebrew (jika belum ada), jalankan perintah tersebut di **terminal**.

```
$ ruby -e "$(curl -fsSL http://git.io/pv0l)"
```

2. Install Golang menggunakan command `brew`.

```
$ brew install go
```

3. Tambahkan path ke environment variable.

```
$ echo "export PATH=$PATH:/usr/local/go/bin" >> ~/.bash_profile  
$ source ~/.bash_profile
```

4. Jalankan command untuk mengecek versi Golang.

```
$ go version
```

5. Jika output adalah sama dengan Golang yang ter-install, menandakan instalasi berhasil.

2.3. Instalasi Golang di Linux

1. Download archive berikut, pilih sesuai jenis bit komputer anda.

- 32bit => [go1.8.3.linux-386.msi](#)
- 64bit => [go1.8.3.linux-amd64.msi](#)

Download bisa dilakukan lewat CLI, menggunakan `wget` atau `curl`.

```
$ wget https://storage.googleapis.com/golang/go1.8.3.linux-amd64.tar.gz
```

2. Buka **terminal**, extract archive tersebut ke `/usr/local`.

```
$ tar -C /usr/local -xzf go1.8.3.linux-amd64.tar.gz
```

3. Setelah itu export path-nya, gunakan command di bawah ini.

```
$ echo "export PATH=$PATH:/usr/local/go/bin" >> ~/.bashrc  
$ source ~/.bashrc
```

4. Selanjutnya, eksekusi perintah berikut untuk mengetes apakah Golang sudah terinstal dengan benar.

```
$ go version
```

5. Jika output adalah sama dengan Golang yang ter-install, menandakan instalasi berhasil.

2.4. Variabel `GOROOT`

By default, setelah proses instalasi Golang selesai, secara otomatis akan muncul environment variabel bernama `GOROOT`. Isi dari environment variabel ini adalah path/folder/lokasi dimana golang di install.

Sebagai contoh di Windows, ketika golang di-install di `c:\go`, maka path tersebut akan menjadi isi dari `GOROOT`.

3. GOPATH Dan Workspace

Ada beberapa hal yang perlu disiapkan sebelum bisa masuk ke sesi pembuatan aplikasi menggunakan Golang, yaitu setup workspace untuk Project yang akan dibuat. Dan di bab ini kita akan belajar bagaimana caranya.

3.1. Variabel GOPATH

GOPATH adalah variabel yang digunakan oleh Golang sebagai rujukan lokasi dimana semua folder project disimpan. Gopath berisikan 3 buah sub folder: `src`, `bin`, dan `pkg`.

Project di Golang harus ditempatkan dalam `$GOPATH/src`. Sebagai contoh anda ingin membuat project dengan nama `belajar`, maka harus dibuatkan sebuah folder dengan nama `belajar`, ditempatkan dalam `src` (`$GOPATH/src/belajar`).

Path separator yang digunakan sebagai contoh di buku ini adalah slash `/`. Khusus pengguna Windows, path separator adalah backslash `\`.

3.2. Setup Workspace

Lokasi folder yang akan dijadikan sebagai workspace bisa ditentukan sendiri. Anda bisa menggunakan alamat folder mana saja, bebas, tapi jangan gunakan path dimana golang di-install (jangan sama dengan `GOROOT`). Lokasi tersebut harus didaftarkan dalam path variable dengan nama `GOPATH`. Sebagai contoh, penulis memilih path `$HOME/Documents/go`, maka saya daftarkan alamat tersebut. Caranya:

- Bagi pengguna **Windows**, tambahkan path folder tersebut ke **path variable** dengan nama `GOPATH`. Setelah variabel terdaftar, cek apakah path sudah terdaftar dengan benar.

Sering terjadi `GOPATH` tidak dikenali meskipun variabel sudah didaftarkan. Jika hal seperti ini terjadi, restart CMD, lalu coba lagi.

- Bagi pengguna Mac OS, export path ke `~/.bash_profile`. Untuk Linux, export ke `~/.bashrc`

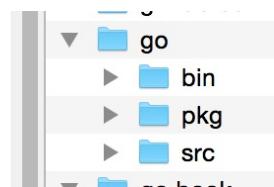
```
$ echo "export GOPATH=$HOME/Documents/go" >> ~/.bash_profile
$ source ~/.bash_profile
```

Cek apakah path sudah terdaftar dengan benar.

```
[novalagung:~ $ source ~/.bash_profile
[novalagung:~ $ echo $GOPATH
/Users/novalagung/Documents/go
novalagung:~ $ ]
```

Setelah `GOPATH` berhasil dikenali, perlu disiapkan 3 buah sub folder didalamnya, dengan kriteria sebagai berikut:

- Folder `src`, adalah path dimana project go lang disimpan
- Folder `pkg`, berisi file hasil kompilasi
- Folder `bin`, berisi file executable hasil build



Struktur diatas merupakan struktur standar workspace Golang. Jadi pastikan penamaan dan hirarki folder adalah sama.

4. Instalasi Editor

Proses pembuatan aplikasi menggunakan Golang akan lebih maksimal jika didukung oleh editor atau **IDE** yang pas. Ada cukup banyak pilihan bagus yang bisa dipertimbangkan, diantaranya: Brackets, IntelliJ, Netbeans, Atom, Brackets, Visual Studio Code, Sublime Text, dan lainnya.

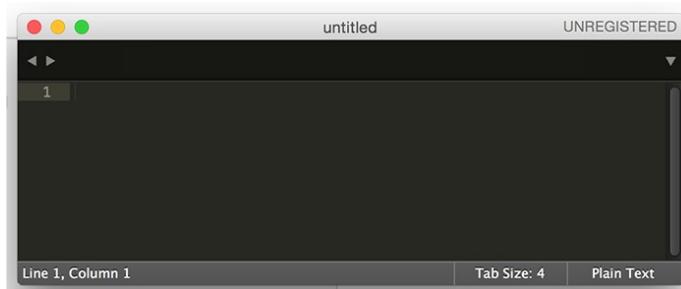
Pada saat menulis buku ini, editor yang penulis gunakan adalah **Sublime Text 3**. Editor ini ringan, mudah didapat, dan memiliki cukup banyak plugin. Pembaca bisa memilih editor yang sama dengan yang digunakan di buku ini, atau editor lainnya, bebas.

Bagi yang memilih Sublime Text, penulis sarankan untuk meng-install plugin bernama **GoSublime**. Plugin ini menyediakan banyak sekali fitur yang sangat membantu proses pengembangan aplikasi menggunakan Golang. Diantaranya seperti *code completion*, *lint* (deteksi kesalahan di level sintaks), *code formatting* (perapian kode otomatis), dan lainnya.

Di bab ini akan dijelaskan bagaimana cara instalasi editor Sublime Text, Package Control, dan plugin GoSublime.

4.1. Instalasi Editor Sublime Text

1. Download **Sublime Text versi 3** di <http://www.sublimetext.com/3>, pilih sesuai dengan sistem operasi yang digunakan.
2. Jalankan installer.
3. Setelah selesai, jalankan aplikasi.



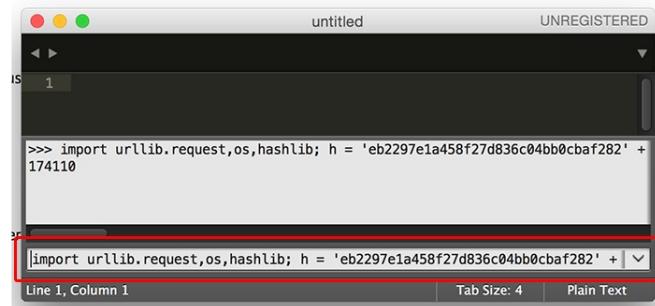
4.2. Instalasi Package Control

Package control merupakan aplikasi 3rd party untuk Sublime Text, digunakan untuk mempermudah instalasi plugin. Default-nya Sublime tidak menyediakan aplikasi ini, kita perlu meng-install-nya sendiri. Silakan ikuti petunjuk berikut untuk cara instalasinya.

1. Buka situs <https://packagecontrol.io/installation>, **copy** script yang ada di tab Sublime Text 3 (tab bagian kiri).

```
SUBLIME TEXT 3 SUBLIME TEXT 2
import urllib.request,os,hashlib; h =
'eb2297e1a458f27d836c04bb0cbaf282' +
'd0e7a3098092775ccb37ca9d6b2e4b7d'; pf = 'Package
Control.sublime-package'; ipp =
sublime.installed_packages_path();
urllib.request.install_opener(
urllib.request.build_opener(
urllib.request.ProxyHandler())); by =
urllib.request.urlopen( 'http://packagecontrol.io/' +
pf.replace(' ', '%20')).read(); dh =
hashlib.sha256(by).hexdigest(); print('Error
validating download (got %s instead of %s), please try
manual install' % (dh, h)) if dh != h else
open(os.path.join( ipp, pf), 'wb').write(by)
```

2. Jalankan aplikasi Sublime Text, klik menu **View > Show Console**, lalu **paste** script yang sudah di-copy tadi, ke inputan kecil di bagian bawah editor. Tekan Enter.

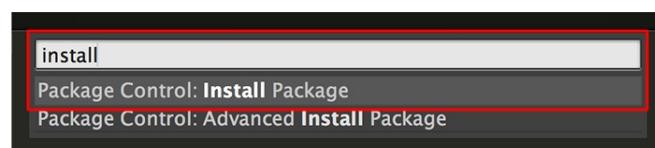


3. Tunggu hingga proses selesai. Perhatikan karakter sama dengan (=) di bagian kiri bawah editor yang bergerak-gerak. Jika karakter tersebut menghilang, menandakan bahwa proses instalasi selesai.
4. Setelah selesai, tutup aplikasi, lalu buka kembali. Package Control sudah berhasil di-install.

4.3. Instalasi Plugin GoSublime

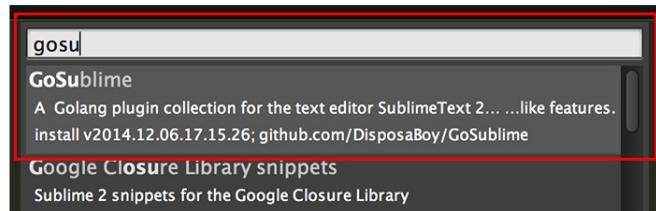
Dengan memanfaatkan Package Control, instalasi plugin akan menjadi lebih mudah. Berikut merupakan langkah instalasi plugin GoSublime.

1. Buka Sublime, tekan **ctrl+shift+p** (atau **cmd+shift+p** untuk pengguna Mac OS), akan muncul sebuah input dialog. Ketikan disana `install`, lalu enter.



4. Instalasi Editor

2. Akan muncul lagi input dialog lainnya, ketikkan `GoSublime`, lalu enter. Tunggu hingga proses selesai (acuan instalasi selesai adalah karakter sama dengan (=) di bagian kiri bawah editor yang sebelumnya bergerak-gerak).



3. Setelah selesai, restart Sublime, plugin GoSublime sudah berhasil ter-install.

5. Command

Pengembangan aplikasi Golang tak jauh dari hal-hal yang berbau CLI atau **Command Line Interface**. Proses kompilasi, testing, eksekusi program, semua dilakukan lewat command line.

Golang menyediakan command `go`, di bab ini kita akan belajar mengenai pemanfaatannya.

5.1. Command `go run`

Command `go run` digunakan untuk eksekusi file program (file ber-ekstensi `.go`). Cara penggunaannya dengan menuliskan command tersebut diikut argumen nama file.

Berikut adalah contoh penerapan `go run` untuk eksekusi file program `bab5.go` yang tersimpan di path `$GOPATH/src/belajar-golang`.

```
$ cd $GOPATH/src/belajar-golang
$ go run bab5.go
```

```
[novalagung:~ $ cd $GOPATH/src/belajar-golang
[novalagung:belajar-golang $ go run bab5.go
Hello World
novalagung:belajar-golang $ ]
```

Command `go run` hanya bisa digunakan pada file yang package-nya adalah **main**. Lebih jelasnya dibahas pada bab selanjutnya (bab 6).

Jika ada banyak file yang ber-package `main`, dan file-file tersebut di-import di file utama, maka eksekusinya adalah dengan menyisipkan semua file sebagai argument `go run` (lebih jelasnya akan dibahas pada bab 25). Contohnya bisa dilihat pada kode berikut.

```
$ go run bab5.go library.go
```

```
[novalagung:belajar-golang $ go run bab5.go library.go
Hello World
novalagung:belajar-golang $ ]
```

Atau bisa dengan menggunakan `*.go`, tanpa tidak perlu menuliskan nama-nama file program yang ada.

```
$ go run *.go
```

5.2. Command go test

Golang menyediakan package `testing`, berguna untuk keperluan unit testing. File yang akan di-test harus ber-suffix `_test.go`.

Berikut adalah contoh penggunaan command `go test` untuk testing file `bab5_test.go`.

```
$ go test bab5_test.go
```

```
[novalagung:belajar-golang $ go test bab5_test.go
ok      command-line-arguments 0.011s
novalagung:belajar-golang $ ]
```

5.3. Command go build

Command ini digunakan untuk mengkompilasi file program.

Sebenarnya ketika eksekusi program menggunakan `go run`, terjadi proses kompilasi juga, file hasil kompilasi akan disimpan pada folder temporary untuk selanjutnya langsung dieksekusi.

Berbeda dengan `go build`, command ini menghasilkan file executable pada folder yang sedang aktif. Contohnya bisa dilihat pada kode berikut.

```
[novalagung:belajar-golang $ go build bab5.go
[novalagung:belajar-golang $ ./bab5
Hello World
novalagung:belajar-golang $ ]]
```

Pada contoh di atas, file `bab5.go` di-build, menghasilkan file baru pada folder yang sama, yaitu `bab5`, yang kemudian dieksekusi.

Pada pengguna Windows, file executable ber-ekstensi `.exe`.

5.4. Command go install

Command `go install` memiliki fungsi yang sama dengan `go build`, hanya saja setelah proses kompilasi selesai, dilanjutkan ke proses instalasi program yang bersangkutan.

Target eksekusi harus berupa folder proyek (bukan file `.go`), dan path folder tersebut dituliskan relatif terhadap `$GOPATH/src`. Contoh:

```
$ go install github.com/novalagung/godong
```

`go install` menghasilkan output berbeda untuk package `main` dan non-main.

- Pada package **non-main**, menghasilkan file berekstensi `.a` tersimpan dalam folder `$GOPATH/pkg .`
- Pada package **main**, menghasilkan file *executable* tersimpan dalam folder `$GOPATH/bin .`

Berikut merupakan contoh penerapan `go install`.

```
[novalagung:godong $ go install github.com/novalagung/godong
[novalagung:godong $ go install github.com/novalagung/godong/godong_test
[novalagung:godong $ ls $GOPATH/pkg/darwin_amd64/github.com/novalagung/
godong      godong.a
[novalagung:godong $ ls $GOPATH/bin
godong_test
[novalagung:godong $ $GOPATH/bin/godong_test
route /dashboard/about-us
-> Dashboard.Action_AboutUs
route /dashboard/data-analytic/get-data
-> Dashboard.Action_DataAnalytic_GetData
route /dashboard/home
-> Dashboard.Action_Home
```

Pada kode di atas bisa dilihat command `go install` dieksekusi 2 kali.

1. Pada package non-main, `github.com/novalagung/godong` . Hasil instalasi adalah file berekstensi `.a` tersimpan pada folder `$GOPATH/pkg .`
2. Pada package main, `github.com/novalagung/godong/godong_test` . Hasil instalasi adalah file executable tersimpan pada folder `$GOPATH/bin .`

5.5. Command `go get`

Command ini berbeda dengan command-command yang sudah dibahas di atas. `go get` digunakan untuk men-download package. Sebagai contoh saya ingin men-download package **Mgo**.

```
$ go get gopkg.in/mgo.v2
$ ls $GOPATH/src/gopkg.in/mgo.v2
```

5. Command

```
[novalagung:src $ go get gopkg.in/mgo.v2
[novalagung:src $ ls $GOPATH/src/gopkg.in/mgo.v2
 LICENSE           export_test.go      session.go
 Makefile          gridfs.go        session_test.go
 README.md         gridfs_test.go   socket.go
 auth.go           internal.go     stats.go
 auth_test.go     log.go          suite_test.go
 bson              queue.go       syscall_test.go
 bulk.go           queue_test.go  syscall_windows_test.go
 bulk_test.go     raceoff.go    testdb
 cluster.go       raceon.go     testserver
 cluster_test.go  saslimpl.go   txn
 dbtest            saslstub.go
 doc.go           server.go
```

[gopkg.in/mgo.v2](#) adalah URL package mgo. Package yang sudah ter-download tersimpan dalam `$GOPATH/src` , dengan struktur folder sesuai dengan URL package-nya. Sebagai contoh, package MGO di atas tersimpan di `$GOPATH/src/gopkg.in/mgo.v2` .

6. Program Pertama: Hello World

Semua persiapan sudah selesai, saatnya mulai masuk pada sesi pembuatan program. Program pertama yang akan kita buat adalah aplikasi kecil untuk memunculkan tulisan **Hello World**.

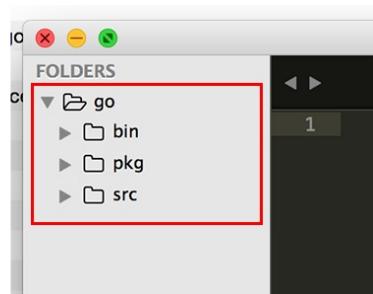
Di bab ini akan dijelaskan secara bertahap dari awal. Mulai pembuatan project, pembuatan file program, sesi penulisan kode (coding), hingga eksekusi aplikasi.

6.1. Load GOPATH Ke Sublime Text

Hal pertama yang perlu dilakukan, adalah me-load atau memunculkan folder `GOPATH` di editor Sublime. Dengan begitu proyek-proyek Golang akan lebih mudah di-maintain.

Caranya:

1. Buka Sublime.
2. Buka explorer/finder, lalu cari ke folder yang merupakan `GOPATH`.
3. Klik-drag folder tersebut (kebetulan lokasi folder `GOPATH` saya bernama `go`), tarik ke Sublime.
4. Seluruh subfolder `GOPATH` akan terbuka di Sublime.



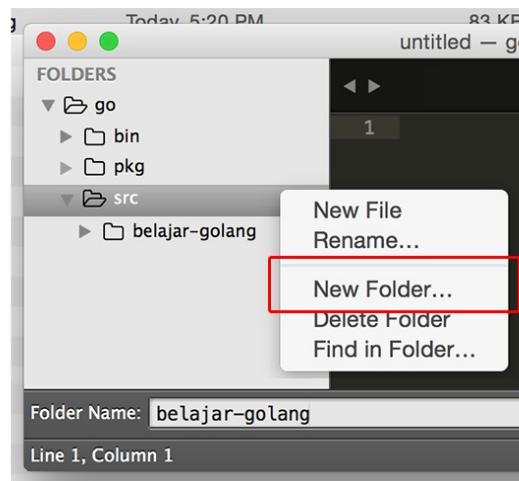
Nama variabel di sistem operasi non-Windows diawali dengan tanda dollar `$`, sebagai contoh `$GOPATH`. Sedangkan di Windows, nama variabel diapit karakter persen `%`, contohnya seperti `%GOPATH%`.

6.2. Menyiapkan Folder Project

Selanjutnya, buat project folder baru dalam `$GOPATH/src`, dengan nama folder bebas (boleh menggunakan nama `belajar-golang` atau lainnya). Agar lebih praktis, buat folder tersebut lewat Sublime. Berikut adalah caranya.

1. Klik kanan di folder `src`.

2. Klik **New Folder**, di bagian bawah akan muncul inputan kecil **Folder Name**.
3. Ketikkan nama folder, **belajar-golang**, lalu enter.



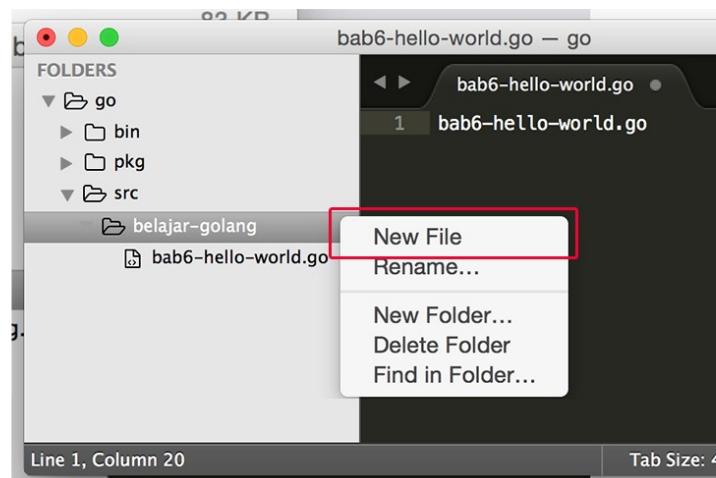
6.3. Menyiapkan File Program

File program disini maksudnya adalah file yang berisikan kode program Golang, yang berekstensi `.go`.

Di dalam project yang telah dibuat (`$GOPATH/src/belajar-golang/`), siapkan sebuah file dengan nama bebas, yang jelas harus ber-ekstensi `.go` (Pada contoh ini saya menggunakan nama file `bab6-hello-world.go`).

Pembuatan file program juga akan dilakukan lewat Sublime. Caranya silakan ikut petunjuk berikut.

1. Klik kanan di folder `belajar-golang`.
2. Klik **New File**, maka akan muncul tab baru di bagian kanan.
3. Ketikkan di konten: `bab6-hello-world.go`.
4. Lalu tekan **ctrl+s** (**cmd+s** untuk M*c OSX), kemudian enter.



6.4. Program Pertama: Hello Word

Setelah project folder dan file program sudah siap, saatnya untuk **coding**.

Dibawah ini merupakan contoh kode program sederhana untuk memunculkan text atau tulisan "**hello world**" ke layar output (command line).

Silakan salin kode berikut ke file program yang telah dibuat. Sebisa mungkin jangan copy paste. Biasakan untuk menulis dari awal, agar cepat terbiasa dan familiar dengan Golang.

```
package main

import "fmt"

func main() {
    fmt.Println("hello world")
}
```

Setelah kode disalin, buka terminal (atau CMD bagi pengguna Windows), lalu masuk ke direktori proyek menggunakan perintah `cd .`

- Windows

```
$ cd %GOPATH%\src\belajar-golang
```

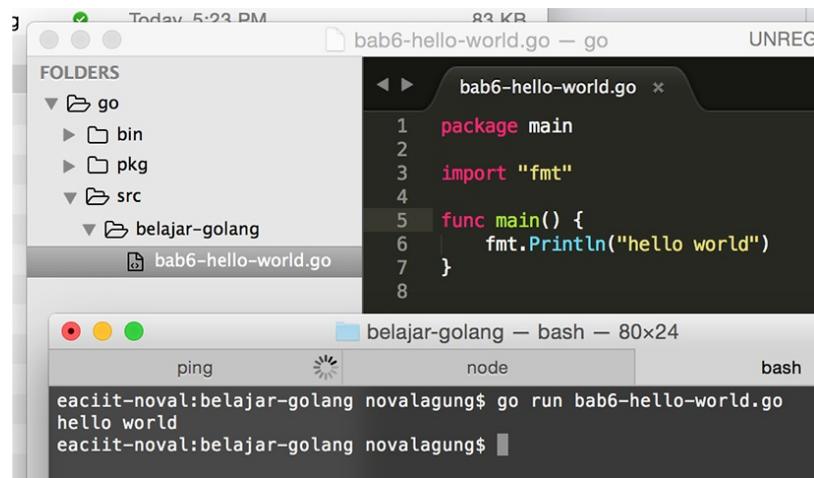
- Non-Windows

```
$ cd $GOPATH/src/belajar-golang
```

Jalankan program dengan perintah `go run .`

```
$ go run bab6-hello-world.go
```

Hasilnya, muncul tulisan **hello world** di layar console.



```
bab6-hello-world.go — go
FOLDERS
  ▶ go
    ▶ bin
    ▶ pkg
    ▶ src
      ▶ belajar-golang
        bab6-hello-world.go

bab6-hello-world.go
1 package main
2
3 import "fmt"
4
5 func main() {
6   fmt.Println("hello world")
7 }

belajar-golang — bash — 80x24
ping node bash
eaciit-noval:belajar-golang novalagung$ go run bab6-hello-world.go
hello world
eaciit-noval:belajar-golang novalagung$
```

Selamat! Anda telah berhasil membuat program menggunakan Golang!

Meski kode program di atas sangat sederhana, mungkin akan muncul beberapa pertanyaan di benak. Di bawah ini merupakan detail penjelasan kode di atas.

6.5. Penggunaan Keyword `package`

Setiap file program harus memiliki package. Setiap project harus ada satu file dengan package bernama `main`. File yang ber-package `main`, akan di eksekusi pertama kali ketika program di jalankan.

Cara menentikan package dengan menggunakan keyword `package`, berikut adalah contoh penulisannya.

```
package <nama-package>
package main
```

6.6. Penggunaan Keyword `import`

Keyword `import` digunakan untuk meng-include atau memasukan package lain kedalam file program, agar isi package yang di-include bisa dimanfaatkan.

Package `fmt` merupakan salah satu package yang disediakan oleh Golang, berisikan banyak fungsi untuk keperluan I/O yang berhubungan dengan text.

Skema penulisan keyword `import` bisa dilihat pada contoh berikut.

```
import "<nama-package>"  
import "fmt"
```

6.7. Penggunaan Fungsi `main()`

Dalam sebuah proyek harus ada file program yang berisikan sebuah fungsi bernama `main()`. Fungsi tersebut harus berada dalam package yang juga bernama `main`. Fungsi `main()` adalah yang dipanggil pertama kali pada saat eksekusi program. Contoh penulisan fungsi `main` :

```
func main() {  
}
```

6.8. Penggunaan Fungsi `fmt.Println()`

Fungsi `fmt.Println()` digunakan untuk memunculkan text ke layar (pada konteks ini, terminal atau CMD). Di program pertama yang telah kita buat, fungsi ini memunculkan tulisan **Hello World**.

Skema penulisan keyword `fmt.Println()` bisa dilihat pada contoh berikut.

```
fmt.Println("<isi-pesan>")  
fmt.Println("hello world")
```

Fungsi `fmt.Println()` berada dalam package `fmt`, maka untuk menggunakannya perlu package tersebut untuk di-import terlebih dahulu.

Fungsi `fmt.Println()` dapat menampung parameter yang tidak terbatas jumlahnya. Semua data parameter akan dimunculkan dengan pemisah tanda spasi.

```
fmt.Println("hello", "world!", "how", "are", "you")
```

Outputnya: **hello world! how are you.**

7. Komentar

Komentar biasa dimanfaatkan untuk menyisipkan catatan pada kode program, menulis penjelasan atau deskripsi mengenai suatu blok kode, atau bisa juga digunakan untuk meremark kode (men-non-aktifkan kode yg tidak digunakan). Komentar akan diabaikan ketika kompilasi maupun eksekusi program.

Ada 2 jenis komentar di Golang, inline & multiline. Di bab akan dijelaskan tentang penerapan dan perbedaan kedua jenis komentar tersebut.

7.1. Komentar Inline

Penulisan komentar jenis ini di awali dengan tanda **double slash** (//) lalu diikuti pesan komentarnya. Komentar inline hanya berlaku untuk satu baris pesan saja. Jika pesan komentar lebih dari satu baris, maka tanda // harus ditulis lagi di baris selanjutnya.

```
package main

import "fmt"

func main() {
    // komentar kode
    // menampilkan pesan hello world
    fmt.Println("hello world")

    // fmt.Println("baris ini tidak akan di eksekusi")
}
```

Mari kita praktikan kode di atas. Siapkan file program baru dalam project folder `belajar-golang` dengan nama bebas. Isi dengan kode di atas, lalu jalankan.

```
[novalagung:belajar-golang $ go run bab7.go
hello world
novalagung:belajar-golang $ ]
```

Hasilnya hanya tulisan **hello world** saja yang muncul di layar, karena semua yang di awali tanda double slash // diabaikan oleh compiler.

7.2. Komentar Multiline

7. Komentar

Komentar yang cukup panjang akan lebih rapi jika ditulis menggunakan teknik komentar multiline. Ciri dari komentar jenis ini adalah penulisannya diawali dengan tanda `/*` dan diakhiri `*/`.

```
/*
    komentar kode
    menampilkan pesan hello world
*/
fmt.Println("hello world")

// fmt.Println("baris ini tidak akan di eksekusi")
```

Sifat komentar ini sama seperti komentar inline, yaitu sama-sama diabaikan oleh compiler.

8. Variabel

Golang mengadopsi dua jenis penulisan variabel, yang dituliskan tipe data-nya dan yang tidak. Kedua cara tersebut valid dan tujuannya sama, pembedanya hanya cara penulisannya saja.

Pada bab ini akan dikupas tuntas tentang macam-macam cara deklarasi variabel.

8.1. Deklarasi Variabel Dengan Tipe Data

Golang memiliki aturan cukup ketat dalam hal penulisan variabel. Ketika deklarasi, tipe data yg digunakan harus dituliskan juga. Istilah lain dari konsep ini adalah **manifest typing**.

Berikut adalah contoh cara pembuatan variabel yang tipe datanya harus ditulis.

```
package main

import "fmt"

func main() {
    var firstName string = "john"

    var lastName string
    lastName = "wick"

    fmt.Printf("halo %s %s!\n", firstName, lastName)
}
```

Keyword `var` di atas digunakan untuk deklarasi variabel, contohnya bisa dilihat pada `firstName` dan `lastName`.

Nilai variabel `firstName` diisi langsung ketika deklarasi, berbeda dibanding `lastName` yang nilainya diisi setelah baris kode deklarasi, hal seperti ini diperbolehkan di Golang.

```
[novalagung:belajar-golang $ go run bab8.go
halo john wick!
novalagung:belajar-golang $ ]
```

8.2. Deklarasi Variabel Menggunakan Keyword `var`

Pada kode di atas bisa dilihat bagaimana sebuah variabel dideklarasikan dan di-set nilainya. Keyword `var` digunakan untuk membuat variabel baru.

Skema penggunaan keyword `var`:

```
var <nama-variabel> <tipe-data>
var <nama-variabel> <tipe-data> = <nilai>
```

Contoh:

```
var lastName string
var firstName string = "john"
```

Nilai variabel bisa di-isi langsung pada saat deklarasi variabel.

8.3. Penggunaan Fungsi `fmt.Printf()`

Fungsi ini digunakan untuk menampilkan output dalam bentuk tertentu. Kegunaannya sama seperti fungsi `fmt.Println()`, hanya saja struktur outputnya didefinisikan di awal.

Perhatikan bagian `"halo %s %s!\n"`, karakter `%s` disitu akan diganti dengan data `string` yang berada di parameter ke-2, ke-3, dan seterusnya.

Ketiga baris kode di bawah ini menghasilkan output yang sama, meskipun cara penulisannya berbeda.

```
fmt.Printf("halo john wick!\n")
fmt.Printf("halo %s %s!\n", firstName, lastName)
fmt.Println("halo", firstName, lastName + "!")
```

Tanda plus (`+`) jika ditempatkan di antara string, fungsinya adalah untuk penggabungan string (istilah lainnya: concatenation).

Fungsi `fmt.Printf()` tidak menghasilkan baris baru di akhir text, oleh karena itu digunakanlah literal `\n` untuk memunculkan baris baru di akhir. Hal ini sangat berbeda jika dibandingkan dengan fungsi `fmt.Println()` yang secara otomatis menghasilkan new line (baris baru) di akhir.

8.4. Deklarasi Variabel Tanpa Tipe Data

Selain **manifest typing**, Golang juga mengadopsi metode **type inference**, yaitu metode deklarasi variabel yang tipe data-nya ditentukan oleh tipe data nilainya, cara kontradiktif jika dibandingkan dengan cara pertama. Dengan metode jenis ini, keyword `var` dan tipe data tidak perlu dituliskan.

```
var firstName string = "john"
lastName := "wick"

fmt.Printf("halo %s %s!\n", firstName, lastName)
```

Variabel `lastName` dideklarasikan dengan menggunakan metode type inference. Penandanya tipe data tidak dituliskan pada saat deklarasi. Pada penggunaan metode ini, operand `=` harus diganti dengan `:=` dan keyword `var` dihilangkan.

Tipe data `lastName` secara otomatis akan ditentukan menyesuaikan value atau nilai-nya. Jika nilainya adalah berupa `string` maka tipe data variabel adalah `string`. Pada contoh di atas, nilainya adalah string `"wick"`.

Diperbolehkan untuk tetap menggunakan keyword `var` pada saat deklarasi meskipun tanpa menuliskan tipe data, dengan ketentuan tidak menggunakan tanda `:=`, melainkan tetap menggunakan `=`.

```
// menggunakan var, tanpa tipe data, menggunakan perantara "="
var firstName = "john"

// tanpa var, tanpa tipe data, menggunakan perantara ":="
lastName := "wick"
```

Kedua deklarasi di atas maksudnya sama. Silakan pilih yang nyaman di hati.

Tanda `:=` hanya digunakan sekali di awal pada saat deklarasi. Untuk assignment nilai selanjutnya harus menggunakan tanda `=`, contoh:

```
lastName := "wick"
lastName = "ethan"
lastName = "bourne"
```

Perlu diketahui, deklarasi menggunakan `:=` hanya bisa dilakukan di dalam fungsi, tidak bisa digunakan di luar fungsi.

8.5. Deklarasi Multi Variabel

Golang mendukung metode deklarasi banyak variabel secara bersamaan, caranya dengan menuliskan variabel-variabelnya dengan pembatas tanda koma (,). Untuk pengisian nilainya-pun diperbolehkan secara bersamaan.

```
var first, second, third string  
first, second, third = "satu", "dua", "tiga"
```

Pengisian nilai juga bisa dilakukan bersamaan pada saat deklarasi. Caranya dengan menuliskan nilai masing-masing variabel berurutan sesuai variabelnya dengan pembatas koma (,).

```
var fourth, fifth, sixth string = "empat", "lima", "enam"
```

Kalau ingin lebih ringkas:

```
seventh, eight, ninth := "tujuh", "delapan", "sembilan"
```

Dengan menggunakan teknik type inference, deklarasi multi variabel bisa dilakukan untuk variabel-variabel yang tipe data satu sama lainnya berbeda.

```
one, isFriday, twoPointTwo, say := 1, true, 2.2, "hello"
```

Istimewa bukan? Istimewa sekali.

8.6. Variabel Underscore

Golang memiliki aturan unik yang jarang dimiliki bahasa lain, yaitu tidak boleh ada satupun variabel yang menganggur. Artinya, semua variabel yang dideklarasikan harus digunakan. Jika ada variabel yang tidak digunakan tapi dideklarasikan, error akan muncul dan program tidak bisa di-run ataupun di-compile.

```
[novalagung:belajar-golang $ go run bab8.go ]  
# command-line-arguments  
.bab8.go:6: name declared and not used  
novalagung:belajar-golang $
```

Underscore (_) adalah predefined variabel yang bisa dimanfaatkan untuk menampung nilai yang tidak dipakai. Bisa dibilang variabel ini merupakan keranjang sampah.

```
_ = "belajar Golang"  
_ = "Golang itu mudah"  
name, _ := "john", "wick"
```

Pada contoh di atas, variabel `name` akan berisikan text `john`, sedang nilai `wick` ditampung oleh variabel underscore, menandakan bahwa nilai tersebut tidak akan digunakan.

Variabel underscore adalah predefined, jadi tidak perlu menggunakan `:=` untuk pengisian nilai, cukup dengan `=` saja. Namun khusus untuk pengisian nilai multi variabel yang dilakukan dengan metode type inference, boleh didalamnya terdapat variabel underscore.

Biasanya variabel underscore sering dimanfaatkan untuk menampung nilai balik fungsi yang tidak digunakan.

Perlu diketahui, bahwa isi variabel underscore tidak dapat ditampilkan. Data yang sudah masuk variabel tersebut akan hilang. Ibarat blackhole, sekali masuk, tidak akan bisa keluar 😞

8.7. Deklarasi Variabel Menggunakan Keyword new

Keyword `new` digunakan untuk mencetak data **pointer** dengan tipe data tertentu. Nilai data default-nya akan menyesuaikan tipe datanya.

```
name := new(string)

fmt.Println(name)    // 0x20818a220
fmt.Println(*name)  // ""
```

Variabel `name` menampung data bertipe **pointer string**. Jika ditampilkan yang muncul bukanlah nilainya melainkan alamat memori nilai tersebut (dalam bentuk notasi heksadesimal). Untuk menampilkan nilai aslinya, variabel tersebut perlu di-**dereference** terlebih dahulu, menggunakan tanda asterisk (`*`).

Mungkin untuk sekarang banyak yang akan bingung tentang apa itu pointer, namun tak apa, karena nantinya di bab 22 akan dikupas habis topik pointer dan dereference.

8.8. Deklarasi Variabel Menggunakan Keyword make

Keyword ini hanya bisa digunakan untuk pembuatan beberapa jenis variabel saja, yaitu:

- channel
- slice
- map

Dan lagi, mungkin banyak yang akan bingung. Ketika sudah masuk ke pembahasan masing-masing poin tersebut, akan terlihat apa kegunaan dari keyword `make` ini.

9. Tipe Data

Golang mengenal beberapa jenis tipe data, diantaranya adalah tipe data numerik (desimal & non-desimal), string, dan boolean.

Di bab-bab sebelumnya secara tak sadar kita sudah mengaplikasikan beberapa tipe data, seperti `string` dan tipe numerik `int`.

Bab ini menjelaskan beberapa macam tipe data standar yang disediakan oleh Golang, beserta cara penggunaannya.

9.1. Tipe Data Numerik Non-Desimal

Tipe data numerik non-desimal atau **non floating point** di Golang ada beberapa jenis. Secara umum ada 2 tipe data kategori ini yang perlu diketahui.

- `uint`, tipe data untuk bilangan cacah (bilangan positif).
- `int`, tipe data untuk bilangan bulat (bilangan negatif dan positif).

Kedua tipe data di atas kemudian dibagi lagi menjadi beberapa jenis, dengan pembagian berdasarkan lebar cakupan nilainya, detailnya bisa dilihat di tabel berikut.

Tipe data	Cakupan bilangan
<code>uint8</code>	$0 \leftrightarrow 255$
<code>uint16</code>	$0 \leftrightarrow 65535$
<code>uint32</code>	$0 \leftrightarrow 4294967295$
<code>uint64</code>	$0 \leftrightarrow 18446744073709551615$
<code>uint</code>	sama dengan <code>uint32</code> atau <code>uint64</code> (tergantung nilai)
<code>byte</code>	sama dengan <code>uint8</code>
<code>int8</code>	$-128 \leftrightarrow 127$
<code>int16</code>	$-32768 \leftrightarrow 32767$
<code>int32</code>	$-2147483648 \leftrightarrow 2147483647$
<code>int64</code>	$-9223372036854775808 \leftrightarrow 9223372036854775807$
<code>int</code>	sama dengan <code>int32</code> atau <code>int64</code> (tergantung nilai)
<code>rune</code>	sama dengan <code>int32</code>

Dianjurkan untuk tidak sembarangan dalam menentukan tipe data variabel, sebisa mungkin tipe yang dipilih harus disesuaikan dengan nilainya, karena efeknya adalah ke alokasi memori variabel. Pemilihan tipe data yang tepat akan membuat pemakaian memori lebih optimal, tidak berlebihan.

```
var positiveNumber uint8 = 89
var negativeNumber = -1243423644

fmt.Printf("bilangan positif: %d\n", positiveNumber)
fmt.Printf("bilangan negatif: %d\n", negativeNumber)
```

Variabel `positiveNumber` bertipe `uint8` dengan nilai awal `89`. Sedangkan variabel `negativeNumber` dideklarasikan dengan nilai awal `-1243423644`. Compiler secara cerdas akan menentukan tipe data variabel tersebut sebagai `int32` (karena angka tersebut masuk ke cakupan tipe data `int32`).

Template `%d` pada `fmt.Printf()` digunakan untuk memformat data numerik non-desimal.

9.2. Tipe Data Numerik Desimal

Tipe data numerik desimal yang perlu diketahui ada 2, `float32` dan `float64`. Perbedaan kedua tipe data tersebut berada di lebar cakupan nilai desimal yang bisa ditampung. Untuk lebih jelasnya bisa merujuk ke spesifikasi [IEEE-754 32-bit floating-point numbers](#).

```
var decimalNumber = 2.62

fmt.Printf("bilangan desimal: %f\n", decimalNumber)
fmt.Printf("bilangan desimal: %.3f\n", decimalNumber)
```

Pada kode di atas, variabel `decimalNumber` akan memiliki tipe data `float32`, karena nilainya berada di cakupan tipe data tersebut.

```
[novalagung:belajar-golang $ go run bab9.go
 bilangan desimal: 2.620000
 bilangan desimal: 2.620
 novalagung:belajar-golang $ ]
```

Template `%f` digunakan untuk memformat data numerik desimal menjadi string. Digit desimal yang akan dihasilkan adalah **6 digit**. Pada contoh di atas, hasil format variabel `decimalNumber` adalah `2.620000`. Jumlah digit yang muncul bisa dikontrol menggunakan `%.nf`, tinggal ganti `n` dengan angka yang diinginkan. Contoh: `%.3f` maka akan menghasilkan 3 digit desimal, `%.10f` maka akan menghasilkan 10 digit desimal.

9.3. Tipe Data `bool` (Boolean)

Tipe data `bool` berisikan hanya 2 variansi nilai, `true` dan `false`. Tipe data ini biasa dimanfaatkan dalam seleksi kondisi dan perulangan (yang nantinya akan kita bahas pada bab 12 dan bab 13).

```
var exist bool = true
fmt.Printf("exist? %t \n", exist)
```

Gunakan `%t` untuk memformat data `bool` menggunakan fungsi `fmt.Printf()`.

9.4. Tipe Data `string`

Ciri khas dari tipe data string adalah nilainya di apit oleh tanda *quote* atau petik dua (").

Contoh penerapannya:

```
var message string = "Halo"
fmt.Printf("message: %s \n", message)
```

Selain menggunakan tanda quote, deklarasi string juga bisa dengan tanda grave accent/backticks (`), tanda ini terletak di sebelah kiri tombol 1. Keistimewaan string yang dideklarasikan menggunakan backtics adalah membuat semua karakter didalamnya **tidak di escape**, termasuk `\n`, tanda petik dua dan tanda petik satu, baris baru, dan lainnya. Semua akan terdeteksi sebagai string.

```
var message = `Nama saya "John Wick".
Salam kenal.
Mari belajar "Golang".`

fmt.Println(message)
```

Ketika dijalankan, output akan muncul sama persis sesuai nilai variabel `message` di atas. Tanda petik dua akan muncul, baris baru juga muncul, sama persis.

```
[novalagung:belajar-golang $ go run bab9.go
Nama saya "John Wick".
Salam kenal.
Mari belajar "Golang".
novalagung:belajar-golang $ ]
```

9.5. Nilai `nil` Dan Nilai Default Tipe Data

`nil` bukan merupakan tipe data, melainkan sebuah nilai. Variabel yang isi nilainya `nil` berarti memiliki nilai kosong.

Semua tipe data yang sudah dibahas di atas memiliki nilai default. Artinya meskipun variabel dideklarasikan dengan tanpa nilai awal, akan ada nilai default-nya.

- Nilai default `string` adalah `""` (string kosong).
- Nilai default `bool` adalah `false`.
- Nilai default tipe numerik non-desimal adalah `0`.
- Nilai default tipe numerik desimal adalah `0.0`.

`nil` adalah nilai kosong, benar-benar kosong. `nil` tidak bisa digunakan pada tipe data yang sudah dibahas di atas, karena kesemuanya akan 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{}`

Nantinya kita akan sering bertemu dengan `nil` setelah masuk pada pembahasan bab-bab tersebut.

10. Konstanta

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

10.1. 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.Println("halo ", firstName, "!\\n")
```

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

```
const lastName = "wick"
fmt.Println("nice to meet you ", lastName, "!\\n")
```

10.2. Penggunaan Fungsi `fmt.Println()`

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

Perbedaan lainnya adalah, nilai pada parameter-parameter yang dimasukkan ke fungsi tersebut digabungkan tanpa pemisah. Tidak seperti pada fungsi `fmt.Println()` yang nilai paremeternya digabung menggunakan penghubung spasi.

```
fmt.Println("john wick")
fmt.Println("john", "wick")

fmt.Print("john wick\\n")
fmt.Print("john ", "wick\\n")
fmt.Print("john", " ", "wick\\n")
```

Kode di atas menunjukkan perbedaan antara `fmt.Println()` dan `fmt.Print()`. Output yang dihasilkan oleh 5 statement di atas adalah sama, meski cara yang digunakan berbeda.

Bila menggunakan `fmt.Println()` tidak perlu menambahkan spasi di tiap kata, karena fungsi tersebut akan secara otomatis menambahkannya di sela-sela nilai. Berbeda dengan `fmt.Print()`, perlu ditambahkan spasi, karena fungsi ini tidak menambahkan spasi di sela-sela nilai parameter yang digabungkan.

11. Operator

Bab ini membahas mengenai macam operator yang bisa digunakan di Golang. Secara umum operator dibagi menjadi 3 kategori: operator aritmatika, perbandingan, dan logika.

11.1. Operator Aritmatika

Operator aritmatika adalah operator yang digunakan untuk operasi yang sifatnya perhitungan. Golang mendukung beberapa operator aritmatika standar, list-nya bisa dilihat di tabel berikut.

Tanda	Penjelasan
+	penjumlahan
-	pengurangan
*	perkalian
/	pembagian
%	modulus / sisa hasil pembagian

Contoh penggunaan:

```
var value = (((2 + 6) % 3) * 4 - 2) / 3
```

11.2. Operator Perbandingan

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

Tabel di bawah ini berisikan operator perbandingan yang bisa digunakan di Golang.

Tanda	Penjelasan
<code>==</code>	apakah nilai kiri sama dengan nilai kanan
<code>!=</code>	apakah nilai kiri tidak sama dengan nilai kanan
<code><</code>	apakah nilai kiri lebih kecil daripada nilai kanan
<code><=</code>	apakah nilai kiri lebih kecil atau sama dengan nilai kanan
<code>></code>	apakah nilai kiri lebih besar dari nilai kanan
<code>>=</code>	apakah nilai kiri lebih besar atau sama dengan nilai kanan

Contoh penggunaan:

```
var value = (((2 + 6) % 3) * 4 - 2) / 3
var isEqual = (value == 2)

fmt.Printf("nilai %d (%t)\n", value, isEqual)
```

Pada kode di atas, terdapat statement operasi aritmatika yang hasilnya ditampung oleh variabel `value`. Selanjutnya, variabel tersebut tersebut dibandingkan dengan angka **2** untuk dicek apakah nilainya sama. Jika iya, maka hasilnya adalah `true`, jika tidak maka `false`. Nilai hasil operasi perbandingan tersebut kemudian disimpan dalam variabel `isEqual`.

```
[novalagung:belajar-golang $ go run bab11.go
nilai 2 (true)
novalagung:belajar-golang $ ]
```

Untuk memunculkan nilai `bool` menggunakan `fmt.Printf()`, bisa gunakan layout format `%t`.

11.3. Operator Logika

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

Beberapa operator logika standar yang bisa digunakan:

Tanda	Penjelasan
<code>&&</code>	kiri dan kanan
<code> </code>	kiri atau kanan
<code>!</code>	negasi / nilai kebalikan

Contoh penggunaan:

```
var left = false
var right = true

var leftAndRight = left && right
fmt.Printf("left && right \t(%t) \n", leftAndRight)

var leftOrRight = left || right
fmt.Printf("left || right \t(%t) \n", leftOrRight)

var leftReverse = !left
fmt.Printf("!left \t\t(%t) \n", leftReverse)
```

Hasil dari operator logika sama dengan hasil dari operator perbandingan, yaitu berupa nilai boolean.

```
[novalagung:belajar-golang $ go run bab11.go
left && right      (false)
left || right       (true)
!left              (true)
novalagung:belajar-golang $ ]
```

Berikut penjelasan statemen operator logika pada kode di atas.

- `leftAndRight` bernilai `false`, karena hasil dari `false dan true` adalah `false`.
- `leftOrRight` bernilai `true`, karena hasil dari `false atau true` adalah `true`.
- `leftReverse` bernilai `true`, karena **negasi** (atau lawan dari) `false` adalah `true`.

Template `\t` digunakan untuk menambahkan indent tabulasi. Biasa dimanfaatkan untuk merapikan tampilan output pada console.

12. Seleksi Kondisi

Seleksi kondisi digunakan untuk mengontrol alur program. Analoginya mirip seperti fungsi rambu lalu lintas di jalan raya. Kapan kendaraan diperbolehkan melaju dan kapan harus berhenti diatur oleh rambu tersebut. Sama seperti pada seleksi kondisi, kapan sebuah blok kode akan dieksekusi juga dikontrol.

Yang dijadikan acuan oleh seleksi kondisi adalah nilai bertipe `bool`, bisa berasal dari variabel, ataupun hasil operasi perbandingan. Nilai tersebut menentukan blok kode mana yang akan dieksekusi.

Golang memiliki 2 macam keyword untuk seleksi kondisi, yaitu **`if else`** dan **`switch`**. Di bab ini kita akan mempelajarinya satu-persatu.

Golang tidak mendukung seleksi kondisi menggunakan **`ternary`**.

Statement seperti: `var data = (isExist ? "ada" : "tidak ada")` adalah invalid dan menghasilkan error.

12.1. Seleksi Kondisi Menggunakan Keyword `if` , `else if` , & `else`

Cara penerapan `if-else` di Golang sama seperti pada bahasa pemrograman lain. Yang membedakan hanya tanda kurungnya (*parentheses*), di Golang tidak perlu ditulis. Kode berikut merupakan contoh penerapan seleksi kondisi `if else`, dengan jumlah kondisi 4 buah.

```
var point = 8

if point == 10 {
    fmt.Println("lulus dengan nilai sempurna")
} else if point > 5 {
    fmt.Println("lulus")
} else if point == 4 {
    fmt.Println("hampir lulus")
} else {
    fmt.Printf("tidak lulus. nilai anda %d\n", point)
}
```

Dari keempat kondisi di atas, yang terpenuhi adalah `if point > 5`, karena nilai variabel `point` memang lebih besar dari `5`. Maka blok kode tepat dibawah kondisi tersebut akan dieksekusi (blok kode ditandai kurung kurawal buka dan tutup), hasilnya text `"lulus"` muncul sebagai output.

```
[novalagung:belajar-golang $ go run bab12.go
lulus
novalagung:belajar-golang $ ]
```

Skema if else Golang sama seperti pada pemrograman umumnya. Yaitu di awal seleksi kondisi menggunakan `if`, dan ketika kondisinya tidak terpenuhi akan menuju ke `else` (jika ada). Ketika ada banyak kondisi, gunakan `else if`.

Di bahasa pemrograman lain, ketika ada seleksi kondisi yang isi blok-nya hanya 1 baris saja, kurung kurawal boleh tidak dituliskan. Berbeda dengan aturan di Golang, kurung kurawal harus tetap dituliskan meski isinya hanya 1 blok satement.

12.2. Variabel Temporary Pada `if - else`

Variabel temporary adalah variabel yang hanya bisa digunakan pada blok seleksi kondisi dimana ia ditempatkan saja. Penggunaan variabel ini membawa beberapa manfaat, antara lain:

- Scope atau cakupan variabel jelas, hanya bisa digunakan pada blok seleksi kondisi itu saja
- Kode menjadi lebih rapi
- Ketika nilai variabel tersebut didapat dari sebuah komputasi, perhitungan tidak perlu dilakukan di dalam blok masing-masing kondisi.

```
var point = 8840.0

if percent := point / 100; percent >= 100 {
    fmt.Printf("%.1f% perfect!\n", percent, "%")
} else if percent >= 70 {
    fmt.Printf("%.1f% good\n", percent, "%")
} else {
    fmt.Printf("%.1f% not bad\n", percent, "%")
}
```

Variabel `percent` nilainya didapat dari hasil perhitungan, dan hanya bisa digunakan di deretan blok seleksi kondisi itu saja.

Deklarasi variabel temporary hanya bisa dilakukan lewat metode type inference yang menggunakan tanda `:=`. Penggunaan keyword `var` disitu tidak diperbolehkan karena akan menyebabkan error.

12.3. Seleksi Kondisi Menggunakan Keyword `switch - case`

Switch merupakan seleksi kondisi yang sifatnya fokus pada satu variabel, lalu kemudian dikenakan nilai lainnya. Contoh sederhananya seperti penentuan apakah nilai variabel `x` adalah: `1`, `2`, `3`, atau lainnya.

```
var point = 6

switch point {
case 8:
    fmt.Println("perfect")
case 7:
    fmt.Println("awesome")
default:
    fmt.Println("not bad")
}
```

Pada kode di atas, tidak ada kondisi atau `case` yang terpenuhi karena nilai variabel `point` tetap `6`. Ketika hal seperti ini terjadi, blok kondisi `default` dipanggil. Bisa dibilang bahwa `default` merupakan `else` dalam sebuah switch.

Perlu diketahui, switch pada pemrograman Golang memiliki perbedaan dibanding bahasa lain. Di Golang, ketika sebuah case terpenuhi, tidak akan dilanjutkan ke pengecekan case selanjutnya, meskipun tidak ada keyword `break` di situ. Konsep ini berkebalikan dengan switch pada umumnya, yang ketika sebuah case terpenuhi, maka akan tetap dilanjut mengecek case selanjutnya kecuali ada keyword `break`.

12.4. Pemanfaatan `case` Untuk Banyak Kondisi

Sebuah `case` dapat menampung banyak kondisi. Cara penerapannya yaitu dengan menuliskan nilai pembanding-pembanding variabel yang di-switch setelah keyword `case` dipisah tanda koma (,).

```
var point = 6

switch point {
case 8:
    fmt.Println("perfect")
case 7, 6, 5, 4:
    fmt.Println("awesome")
default:
    fmt.Println("not bad")
}
```

Kondisi `case 7, 6, 5, 4:` akan terpenuhi ketika nilai variabel `point` adalah 7 atau 6 atau 5 atau 4.

12.5. Kurung Kurawal Pada Keyword `case` & `default`

Tanda kurung kurawal (`{ }`) bisa diterapkan pada keyword `case` dan `default`. Tanda ini opsional, boleh dipakai boleh tidak. Bagus jika dipakai pada blok kondisi yang didalamnya ada banyak statement, kode akan terlihat lebih rapi dan mudah di-maintain.

Perhatikan kode berikut, bisa dilihat pada keyword `default` terdapat kurung kurawal yang mengapit 2 statement didalamnya.

```
var point = 6

switch point {
case 8:
    fmt.Println("perfect")
case 7, 6, 5, 4:
    fmt.Println("awesome")
default:
{
    fmt.Println("not bad")
    fmt.Println("you can be better!")
}
}
```

12.6. Switch Dengan Gaya `if - else`

Uniknya di Golang, switch bisa digunakan dengan gaya ala if-else. Nilai yang akan dibandingkan tidak dituliskan setelah keyword `switch`, melainkan akan ditulis langsung dalam bentuk perbandingan dalam keyword `case`.

Pada kode di bawah ini, kode program switch di atas diubah ke dalam gaya `if-else`. Variabel `point` dihilangkan dari keyword `switch`, lalu kondisi-kondisinya dituliskan di tiap `case`.

```

var point = 6

switch {
case point == 8:
    fmt.Println("perfect")
case (point < 8) && (point > 3):
    fmt.Println("awesome")
default:
{
    fmt.Println("not bad")
    fmt.Println("you need to learn more")
}
}

```

12.7. Penggunaan Keyword `fallthrough` Dalam `switch`

Seperti yang sudah dijelaskan sebelumnya, bahwa switch pada Golang memiliki perbedaan dengan bahasa lain. Ketika sebuah `case` terpenuhi, pengecekan kondisi tidak akan diteruskan ke case-case setelahnya.

Keyword `fallthrough` digunakan untuk memaksa proses pengecekan diteruskan ke `case` selanjutnya.

```

var point = 6

switch {
case point == 8:
    fmt.Println("perfect")
case (point < 8) && (point > 3):
    fmt.Println("awesome")
    fallthrough
case point < 5:
    fmt.Println("you need to learn more")
default:
{
    fmt.Println("not bad")
    fmt.Println("you need to learn more")
}
}

```

Setelah pengecekan `case (point < 8) && (point > 3)` selesai, akan dilanjut ke pengecekan `case point < 5`, karena ada `fallthrough` di situ.

```
[novalagung:belajar-golang $ go run bab12.go
awesome
you need to learn more
novalagung:belajar-golang $ ]
```

12.8. Seleksi Kondisi Bersarang

Seleksi kondisi bersarang adalah seleksi kondisi, yang berada dalam seleksi kondisi, yang mungkin juga berada dalam seleksi kondisi, dan seterusnya. Seleksi kondisi bersarang bisa dilakukan pada `if - else`, `switch`, ataupun kombinasi keduanya.

```
var point = 10

if point > 7 {
    switch point {
        case 10:
            fmt.Println("perfect!")
        default:
            fmt.Println("nice!")
    }
} else {
    if point == 5 {
        fmt.Println("not bad")
    } else if point == 3 {
        fmt.Println("keep trying")
    } else {
        fmt.Println("you can do it")
        if point == 0 {
            fmt.Println("try harder!")
        }
    }
}
```

13. Perulangan

Perulangan adalah proses mengulang-ulang eksekusi blok kode tanpa henti, selama kondisi yang dijadikan acuan terpenuhi. Biasanya disiapkan variabel untuk iterasi atau variabel penanda kapan perulangan akan diberhentikan.

Di Golang keyword perulangan hanya `for` saja, tetapi meski demikian, kemampuannya merupakan gabungan `for`, `foreach`, dan `while` ibarat bahasa pemrograman lain.

13.1. Perulangan Menggunakan Keyword `for`

Ada beberapa cara standar menggunakan `for`. Cara pertama dengan memasukkan variabel counter perulangan beserta kondisinya setelah keyword. Perhatikan dan praktikan kode berikut.

```
for i := 0; i < 5; i++ {
    fmt.Println("Angka", i)
}
```

Perulangan di atas hanya akan berjalan ketika variabel `i` bernilai dibawah `5`, dengan ketentuan setiap kali perulangan, nilai variabel `i` akan di-iterasi atau ditambahkan 1 (`i++` artinya ditambah satu, sama seperti `i = i + 1`). Karena `i` pada awalnya bernilai 0, maka perulangan akan berlangsung 5 kali, yaitu ketika `i` bernilai 0, 1, 2, 3, dan 4.

```
[novalagung:belajar-golang $ go run bab13.go
Angka 0
Angka 1
Angka 2
Angka 3
Angka 4
novalagung:belajar-golang $ ]
```

13.2. Penggunaan Keyword `for` Dengan Argumen Hanya Kondisi

Cara ke-2 adalah dengan menuliskan kondisi setelah keyword `for` (hanya kondisi). Deklarasi dan iterasi variabel counter tidak dituliskan setelah keyword, hanya kondisi perulangan saja. Konsepnya mirip seperti `while` milik bahasa pemrograman lain.

Kode berikut adalah contoh `for` dengan argumen hanya kondisi (seperti if), output yang dihasilkan sama seperti penerapan for cara pertama.

```
var i = 0

for i < 5 {
    fmt.Println("Angka", i)
    i++
}
```

13.3. Penggunaan Keyword `for` Tanpa Argumen

Cara ke-3 adalah `for` ditulis tanpa kondisi. Dengan ini akan dihasilkan perulangan tanpa henti (sama dengan `for true`). Pemberhentian perulangan dilakukan dengan menggunakan keyword `break`.

```
var i = 0

for {
    fmt.Println("Angka", i)

    i++
    if i == 5 {
        break
    }
}
```

Dalam perulangan tanpa henti di atas, variabel `i` yang nilai awalnya `0` di-inkrementasi. Ketika nilai `i` sudah mencapai `5`, keyword `break` digunakan, dan perulangan akan berhenti.

13.4. Penggunaan Keyword `for - range`

Cara ke-4 adalah perulangan dengan menggunakan kombinasi keyword `for` dan `range`. Cara ini biasa digunakan untuk me-looping data bertipe array. Detailnya akan dibahas dalam bab selanjutnya (bab 14).

13.5. Penggunaan Keyword `break` & `continue`

Keyword `break` digunakan untuk menghentikan secara paksa sebuah perulangan, sedangkan `continue` dipakai untuk memaksa maju ke perulangan berikutnya.

Berikut merupakan contoh penerapan `continue` dan `break`. Kedua keyword tersebut dimanfaatkan untuk menampilkan angka genap berurutan yang lebih besar dari 0 dan dibawah 8.

```
for i := 1; i <= 10; i++ {
    if i % 2 == 1 {
        continue
    }

    if i > 8 {
        break
    }

    fmt.Println("Angka", i)
}
```

Kode di atas akan lebih mudah dicerna jika dijelaskan secara berurutan. Berikut adalah penjelasannya.

1. Dilakukan perulangan mulai angka 1 hingga 10 dengan `i` sebagai variabel iterasi.
2. Ketika `i` adalah ganjil (dapat diketahui dari `i % 2`, jika hasilnya `1`, berarti ganjil), maka akan dipaksa lanjut ke perulangan berikutnya.
3. Ketika `i` lebih besar dari 8, maka perulangan akan berhenti.
4. Nilai `m` ditampilkan.

```
[novalagung:belajar-golang $ go run bab13.go
Angka 2
Angka 4
Angka 6
Angka 8
novalagung:belajar-golang $ ]
```

13.6. Perulangan Bersarang

Tak hanya seleksi kondisi yang bisa bersarang, perulangan juga bisa. Cara pengaplikasianya kurang lebih sama, tinggal tulis blok statement perulangan didalam perulangan.

```
for i := 0; i < 5; i++ {
    for j := i; j < 5; j++ {
        fmt.Print(j, " ")
    }

    fmt.Println()
}
```

Pada kode di atas, untuk pertama kalinya fungsi `fmt.Println()` dipanggil tanpa disisipkan parameter. Cara seperti ini bisa digunakan untuk menampilkan baris baru. Kegunaannya sama seperti output dari statement `fmt.Print("\n")`.

```
[novalagung:belajar-golang $ go run bab13.go
0 1 2 3 4
1 2 3 4
2 3 4
3 4
4
novalagung:belajar-golang $ ]
```

13.7. Pemanfaatan Label Dalam Perulangan

Di perulangan bersarang, `break` dan `continue` akan berlaku pada blok perulangan dimana ia digunakan saja. Ada cara agar kedua keyword ini bisa tertuju pada perulangan terluar atau perulangan tertentu, yaitu dengan memanfaatkan teknik pemberian **label**.

Program untuk memunculkan matriks berikut merupakan contoh penerapan label perulangan.

```
outerLoop:
for i := 0; i < 5; i++ {
    for j := 0; j < 5; j++ {
        if i == 3 {
            break outerLoop
        }
        fmt.Println("matriks [", i, "][", j, "]", "\n")
    }
}
```

Tepat sebelum keyword `for` terluar, terdapat baris kode `outerLoop:`. Maksud dari kode tersebut adalah disiapkan sebuah label bernama `outerLoop` untuk `for` dibawahnya. Nama label bisa diganti dengan nama lain (dan harus diakhiri dengan tanda titik dua atau *colon* (`:`)).

Pada `for` bagian dalam, terdapat seleksi kondisi untuk pengecekan nilai `i`. Ketika nilai tersebut sama dengan `3`, maka `break` dipanggil dengan target adalah perulangan yang dilabeli `outerLoop`, perulangan tersebut akan dihentikan.

13. Perulangan

```
[novalagung:belajar-golang $ go run bab13.go ]  
matriks [0][0]  
matriks [0][1]  
matriks [0][2]  
matriks [0][3]  
matriks [0][4]  
matriks [1][0]  
matriks [1][1]  
matriks [1][2]  
matriks [1][3]  
matriks [1][4]  
matriks [2][0]  
matriks [2][1]  
matriks [2][2]  
matriks [2][3]  
matriks [2][4]  
novalagung:belajar-golang $ █
```

14. Array

Array adalah kumpulan data bertipe sama, yang disimpan dalam sebuah variabel. Array memiliki kapasitas yang nilainya ditentukan pada saat pembuatan, menjadikan elemen/data yang disimpan di array tersebut jumlahnya tidak boleh melebihi yang sudah dialokasikan. Default nilai tiap elemen array pada awalnya tergantung dari tipe datanya. Jika `int` maka default nya `0`, jika `bool` maka default-nya `false`, dan tipe data lain. Setiap elemen array memiliki indeks berupa angka yang merepresentasikan posisi urutan elemen tersebut. Indeks array dimulai dari 0.

Contoh penerapan array:

```
var names [4]string
names[0] = "trafalgar"
names[1] = "d"
names[2] = "water"
names[3] = "law"

fmt.Println(names[0], names[1], names[2], names[3])
```

Variabel `names` dideklarasikan sebagai `array string` dengan alokasi elemen `4` slot. Cara mengisi slot elemen array bisa dilihat di kode di atas, yaitu dengan langsung mengakses elemen menggunakan indeks, lalu mengisinya.

```
[novalagung:belajar-golang $ go run bab14.go
trafalgar d water law
novalagung:belajar-golang $ ]
```

14.1. Pengisian Elemen Array yang Melebihi Alokasi Awal

Pengisian elemen array pada indeks yang tidak sesuai dengan alokasi menghasilkan error. Contoh sederhana, jika array memiliki 4 slot, maka pengisian nilai slot 5 seterusnya adalah tidak valid.

```
var names [4]string
names[0] = "trafalgar"
names[1] = "d"
names[2] = "water"
names[3] = "law"
names[4] = "ez" // baris kode ini menghasilkan error
```

Solusi dari masalah di atas adalah dengan menggunakan keyword `append`, yang di bab selanjutnya akan kita bahas.

14.2. Inisialisasi Nilai Awal Array

Pengisian elemen array bisa dilakukan pada saat deklarasi variabel. Caranya dengan menuliskan data elemen dalam kurung kurawal setelah tipe data, dengan pembatas antar elemen adalah tanda koma (,).

```
var fruits = [4]string{"apple", "grape", "banana", "melon"}  
  
fmt.Println("Jumlah element \t\t", len(fruits))  
fmt.Println("Isi semua element \t", fruits)
```

Penggunaan fungsi `fmt.Println()` pada data array tanpa mengakses indeks tertentu, akan menghasilkan output dalam bentuk string dari semua array yang ada. Teknik ini biasa digunakan untuk **debugging** data array.

```
[novalagung:belajar-golang $ go run bab14.go  
Jumlah element      4  
Isi semua element    [apple grape banana melon]  
novalagung:belajar-golang $ ]
```

Fungsi `len()` dipakai untuk menghitung jumlah elemen sebuah array.

14.3. Inisialisasi Nilai Array Dengan Gaya Vertikal

Elemen array bisa dituliskan dalam bentuk horizontal (seperti yang sudah dicontohkan di atas) ataupun dalam bentuk vertikal.

```
var fruits [4]string  
  
// cara horizontal  
fruits = [4]string{"apple", "grape", "banana", "melon"}  
  
// cara vertikal  
fruits = [4]string{  
    "apple",  
    "grape",  
    "banana",  
    "melon",  
}
```

Khusus untuk deklarasi array dengan cara vertikal, tanda koma wajib dituliskan setelah elemen, termasuk elemen terakhir. Jika tidak, maka akan muncul error.

14.4. Inisialisasi Nilai Awal Array Tanpa Jumlah Elemen

Deklarasi array yang nilainya diset di awal, boleh tidak dituliskan jumlah lebar array-nya, cukup ganti dengan tanda 3 titik (...). Jumlah elemen akan dikalkulasi secara otomatis menyesuaikan data elemen yang diisi.

```
var numbers = [...]int{2, 3, 2, 4, 3}

fmt.Println("data array \t:", numbers)
fmt.Println("jumlah elemen \t:", len(numbers))
```

Variabel `numbers` akan secara otomatis memiliki jumlah elemen 5, karena pada saat deklarasi disiapkan 5 buah elemen.

```
[novalagung:belajar-golang $ go run bab14.go
data array      : [2 3 2 4 3]
jumlah elemen   : 5
novalagung:belajar-golang $ ]
```

14.5. Array Multidimensi

Array multidimensi adalah array yang tiap elemennya juga berupa array (dan bisa seterusnya, tergantung kedalaman dimensinya).

Cara deklarasi array multidimensi secara umum sama dengan cara deklarasi array biasa, dengan cara menuliskan data array dimensi selanjutnya sebagai elemen array dimensi sebelumnya.

Khusus untuk array yang merupakan sub dimensi atau elemen, boleh tidak dituliskan jumlah datanya. Contohnya bisa dilihat pada deklarasi variabel `numbers2` di kode berikut.

```
var numbers1 = [2][3]int{{3]int{3, 2, 3}, [3]int{3, 4, 5}}
var numbers2 = [2][3]int{{3, 2, 3}, {3, 4, 5}]

fmt.Println("numbers1", numbers1)
fmt.Println("numbers2", numbers2)
```

Kedua array di atas memiliki elemen yang sama.

```
[novalagung:belajar-golang $ go run bab14.go
numbers1 [[3 2 3] [3 4 5]]
numbers2 [[3 2 3] [3 4 5]]
novalagung:belajar-golang $ ]
```

14.6. Perulangan Elemen Array Menggunakan Keyword `for`

Keyword `for` dan array memiliki hubungan yang sangat erat. Dengan memanfaatkan perulangan menggunakan keyword ini, elemen-elemen dalam array bisa didapat.

Ada beberapa cara yang bisa digunakan untuk me-looping data array, yg pertama adalah dengan memanfaatkan variabel iterasi perulangan untuk mengakses elemen berdasarkan indeks-nya. Contoh:

```
var fruits = [4]string{"apple", "grape", "banana", "melon"}

for i := 0; i < len(fruits); i++ {
    fmt.Printf("elemen %d : %s\n", i, fruits[i])
}
```

Perulangan di atas dijalankan sebanyak jumlah elemen array `fruits` (bisa diketahui dari kondisi `i < len(fruits)`). Di tiap perulangan, elemen array diakses lewat variabel iterasi `i`.

```
[novalagung:belajar-golang $ go run bab14.go
elemen 0 : apple
elemen 1 : grape
elemen 2 : banana
elemen 3 : melon
novalagung:belajar-golang $ ]
```

14.7. Perulangan Elemen Array Menggunakan Keyword `for - range`

Ada cara yang lebih sederhana me-looping data array, dengan menggunakan keyword `for - range`. Contoh pengaplikasiannya bisa dilihat di kode berikut.

```
var fruits = [4]string{"apple", "grape", "banana", "melon"}

for i, fruit := range fruits {
    fmt.Printf("elemen %d : %s\n", i, fruit)
}
```

Array `fruits` diambil elemen-nya secara berurutan. Nilai tiap elemen ditampung variabel oleh `fruit` (tanpa huruf s), sedangkan indeks nya ditampung variabel `i`.

Output program di atas, sama dengan output program sebelumnya, hanya cara yang digunakan berbeda.

14.8. Penggunaan Variabel Underscore `_` Dalam `for - range`

Kadang kala ketika *looping* menggunakan `for - range`, ada kemungkinan dimana data yang dibutuhkan adalah elemen-nya saja, indeks-nya tidak. Sedangkan kode di atas, `range` mengembalikan 2 data, yaitu indeks dan elemen.

Seperti yang sudah diketahui, bahwa di Golang tidak memperbolehkan adanya variabel yang menaggur atau tidak dipakai. Jika dipaksakan, error akan muncul, contohnya seperti kode berikut.

```
var fruits = [4]string{"apple", "grape", "banana", "melon"}  
  
for i, fruit := range fruits {  
    fmt.Printf("nama buah : %s\n", fruit)  
}
```

Hasil dari kode program di atas:

```
[novalagung:belajar-golang $ go run bab14.go  
# command-line-arguments  
.bab14.go:8: i declared and not used  
novalagung:belajar-golang $ ]
```

Disinilah salah satu kegunaan variabel pengangguran, atau underscore (`_`). Tampung saja nilai yang tidak ingin digunakan ke underscore.

```
var fruits = [4]string{"apple", "grape", "banana", "melon"}  
  
for _, fruit := range fruits {  
    fmt.Printf("nama buah : %s\n", fruit)  
}
```

Pada kode di atas, yang sebelumnya adalah variabel `i` diganti dengan `_`, karena kebetulan variabel `i` tidak digunakan.

```
[novalagung:belajar-golang $ go run bab14.go
  nama buah : apple
  nama buah : grape
  nama buah : banana
  nama buah : melon
novalagung:belajar-golang $ ]
```

Jika yang dibutuhkan hanya indeks elemen-nya saja, bisa gunakan 1 buah variabel setelah keyword `for`.

```
for i, _ := range fruits { }
// atau
for i := range fruits { }
```

14.9. Alokasi Elemen Array Menggunakan Keyword `make`

Deklarasi sekaligus alokasi data array juga bisa dilakukan lewat keyword `make`.

```
var fruits = make([]string, 2)
fruits[0] = "apple"
fruits[1] = "manggo"

fmt.Println(fruits) // [apple manggo]
```

Parameter pertama keyword `make` diisi dengan tipe data elemen array yang diinginkan, parameter kedua adalah jumlah elemennya. Pada kode di atas, variabel `fruits` tercetak sebagai array string dengan alokasi 2 slot.

15. Slice

Slice adalah *reference* elemen array. Slice bisa dibuat, atau bisa juga dihasilkan dari manipulasi sebuah array ataupun slice lainnya. Karena merupakan data *reference*, menjadikan perubahan data di tiap elemen slice akan berdampak pada slice lain yang memiliki alamat memori yang sama.

15.1. Inisialisasi Slice

Cara pembuatan slice mirip seperti pembuatan array, bedanya tidak perlu mendefinisikan jumlah elemen ketika awal deklarasi. Pengaksesan nilai elemen-nya juga sama. Kode berikut adalah contoh pembuatan slice.

```
var fruits = []string{"apple", "grape", "banana", "melon"}  
fmt.Println(fruits[0]) // "apple"
```

Salah satu perbedaan slice dan array bisa diketahui pada saat deklarasi variabel-nya, jika jumlah elemen tidak dituliskan, maka variabel tersebut adalah slice.

```
var fruitsA = []string{"apple", "grape"}      // slice  
var fruitsB = [2]string{"banana", "melon"}     // array  
var fruitsC = [...]string{"papaya", "grape"}   // array
```

15.2. Hubungan Slice Dengan Array & Operasi Slice

Kalau perbedannya hanya di penentuan alokasi pada saat inisialisasi, kenapa tidak menggunakan satu istilah saja? atau adakah perbedaan lainnya?

Sebenarnya slice dan array tidak bisa dibedakan karena merupakan sebuah kesatuan. Array adalah kumpulan nilai atau elemen, sedang slice adalah referensi tiap elemen tersebut.

Slice bisa dibentuk dari array yang sudah didefinisikan, caranya dengan memanfaatkan teknik **2 index** untuk mengambil elemen-nya. Contoh bisa dilihat pada kode berikut.

```
var fruits = []string{"apple", "grape", "banana", "melon"}  
var newFruits = fruits[0:2]  
  
fmt.Println(newFruits) // ["apple", "grape"]
```

Kode `fruits[0:2]` maksudnya adalah pengaksesan elemen dalam slice `fruits` yang **dimulai dari indeks ke-0, hingga elemen sebelum indeks ke-2**. Elemen yang memenuhi kriteria tersebut akan didapat, untuk kemudian disimpan pada variabel lain sebagai slice baru. Pada contoh di atas, `newFruits` adalah slice baru yang tercetak dari slice `fruits`, dengan isi 2 elemen, yaitu `"apple"` dan `"grape"`.

```
[novalagung:belajar-golang $ go run bab15.go  
[apple grape]  
novalagung:belajar-golang $ ]
```

Ketika mengakses elemen array menggunakan satu buah indeks (seperti `data[2]`), nilai yang didapat merupakan hasil **copy** dari referensi aslinya. Berbeda dengan pengaksesan elemen menggunakan 2 indeks (seperti `data[0:2]`), nilai yang didapat adalah **reference** elemen atau slice.

Tidak apa jikalau pembaca masih bingung, di bawah akan dijelaskan lebih mendetail lagi tentang slice dan *reference*

Tabel berikut adalah list operasi operasi menggunakan teknik 2 indeks yang bisa dilakukan.

```
var fruits = []string{"apple", "grape", "banana", "melon"}
```

Kode	Output	Penjelasan
<code>fruits[0:2]</code>	<code>[apple, grape]</code>	semua elemen mulai indeks ke-0, hingga sebelum indeks ke-2
<code>fruits[0:4]</code>	<code>[apple, grape, banana, melon]</code>	semua elemen mulai indeks ke-0, hingga sebelum indeks ke-4
<code>fruits[0:0]</code>	<code>[]</code>	menghasilkan slice kosong, karena tidak ada elemen sebelum indeks ke-0
<code>fruits[4:4]</code>	<code>[]</code>	menghasilkan slice kosong, karena tidak ada elemen yang dimulai dari indeks ke-4
<code>fruits[4:0]</code>	<code>[]</code>	error, pada penulisan <code>fruits[a,b]</code> nilai <code>a</code> harus lebih besar atau sama dengan <code>b</code>
<code>fruits[:]</code>	<code>[apple, grape, banana, melon]</code>	semua elemen
<code>fruits[2:]</code>	<code>[banana, melon]</code>	semua elemen mulai indeks ke-2
<code>fruits[:2]</code>	<code>[apple, grape]</code>	semua elemen hingga sebelum indeks ke-2

15.3. Slice Merupakan Tipe Data Reference

Slice merupakan tipe data *reference* atau referensi. Artinya jika ada slice baru yang terbentuk dari slice lama, maka data elemen slice yang baru akan memiliki alamat memori yang sama dengan elemen slice lama. Setiap perubahan yang terjadi di elemen slice baru, akan berdampak juga pada elemen slice lama yang memiliki referensi yang sama.

Program berikut merupakan pembuktian tentang teori yang baru kita bahas. Kita akan mencoba mengubah data elemen slice baru, yang terbentuk dari slice lama.

```

var fruits = []string{"apple", "grape", "banana", "melon"}

var aFruits = fruits[0:3]
var bFruits = fruits[1:4]

var aaFruits = aFruits[1:2]
var baFruits = bFruits[0:1]

fmt.Println(fruits)    // [apple grape banana melon]
fmt.Println(aFruits)   // [apple grape banana]
fmt.Println(bFruits)   // [grape banana melon]
fmt.Println(aaFruits) // [grape]
fmt.Println(baFruits) // [grape]

// Buah "grape" diubah menjadi "pinnapple"
baFruits[0] = "pinnapple"

fmt.Println(fruits)    // [apple pinnapple banana melon]
fmt.Println(aFruits)   // [apple pinnapple banana]
fmt.Println(bFruits)   // [pinnapple banana melon]
fmt.Println(aaFruits) // [pinnapple]
fmt.Println(baFruits) // [pinnapple]

```

Sekilas bisa kita lihat bahwa setelah slice yang isi datanya adalah `grape` di-ubah menjadi `pinnapple`, semua slice pada 4 variabel lainnya juga ikut berubah.

Variabel `aFruits`, `bFruits` merupakan slice baru yang terbentuk dari variabel `fruits`. Dengan menggunakan dua slice baru tersebut, diciptakan lagi slice lainnya, yaitu `aaFruits`, dan `baFruits`. Kelima slice tersebut ditampilkan nilainya.

Selanjutnya, nilai dari `baFruits[0]` diubah, dan 5 slice tadi ditampilkan lagi. Hasilnya akan ada banyak slice yang elemennya ikut berubah. Yaitu elemen-elemen yang referensi-nya sama dengan referensi elemen `baFruits[0]`.

```

[novalagung:belajar-golang $ go run bab15.go
fruits          [apple grape banana melon]
aFruits         [apple grape banana]
bFruits         [grape banana melon]
aaFruits        [grape]
baFruits        [grape]

Buah "grape" diubah menjadi "pinnapple"

fruits          [apple pinnapple banana melon]
aFruits         [apple pinnapple banana]
bFruits         [pinnapple banana melon]
aaFruits        [pinnapple]
baFruits        [pinnapple]
novalagung:belajar-golang $ ]

```

Bisa dilihat pada output di atas, elemen yang sebelumnya bernilai `"grape"` pada variabel `fruits`, `aFruits`, `bFruits`, `aaFruits`, dan `baFruits`; kesemuanya berubah menjadi `"pinnapple"`, karena memiliki referensi yang sama.

Pembahasan mengenai dasar slice sepertinya sudah cukup, selanjutnya kita akan membahas tentang beberapa *built in function* bawaan Golang, yang bisa dimanfaatkan untuk keperluan operasi slice.

15.4. Fungsi `len()`

Fungsi `len()` digunakan untuk menghitung jumlah elemen slice yang ada. Sebagai contoh jika sebuah variabel adalah slice dengan data 4 buah, maka fungsi ini pada variabel tersebut akan mengembalikan angka **4**.

```
var fruits = []string{"apple", "grape", "banana", "melon"}
fmt.Println(len(fruits)) // 4
```

15.5. Fungsi `cap()`

Fungsi `cap()` digunakan untuk menghitung lebar atau kapasitas maksimum slice. Nilai kembalian fungsi ini untuk slice yang baru dibuat pasti sama dengan `len`, tapi bisa berubah seiring operasi slice yang dilakukan. Agar lebih jelas, silakan disimak kode berikut.

```
var fruits = []string{"apple", "grape", "banana", "melon"}
fmt.Println(len(fruits)) // len: 4
fmt.Println(cap(fruits)) // cap: 4

var aFruits = fruits[0:3]
fmt.Println(len(aFruits)) // len: 3
fmt.Println(cap(aFruits)) // cap: 4

var bFruits = fruits[1:4]
fmt.Println(len(bFruits)) // len: 3
fmt.Println(cap(bFruits)) // cap: 3
```

Variabel `fruits` disiapkan di awal dengan jumlah elemen 4, fungsi `len(fruits)` dan `cap(fruits)` pasti hasilnya 4.

Variabel `aFruits` dan `bFruits` merupakan slice baru berisikan 3 buah elemen milik slice `fruits`. Variabel `aFruits` mengambil elemen index 0, 1, 2; sedangkan `bFruits` 1, 2, 3.

Fungsi `len()` menghasilkan angka 3, karena jumlah elemen kedua slice ini adalah 3. Tetapi `cap(aFruits)` menghasilkan angka yang berbeda, yaitu 4 untuk `aFruits` dan 3 untuk `bFruits`. Kenapa? jawabannya bisa dilihat pada tabel berikut.

Kode	Output	len()	cap()
fruits[0:4]	[buah buah buah buah]	4	4
aFruits[0:3]	[buah buah buah ----]	3	4
bFruits[1:3]	---- [buah buah buah]	3	3

Kita analogikan slicing 2 index menggunakan **x** dan **y**.

```
fruits[x:y]
```

Slicing yang dimulai dari indeks **0** hingga **y** akan mengembalikan elemen-elemen mulai indeks **0** hingga sebelum indeks **y**, dengan lebar kapasitas adalah sama dengan slice aslinya.

Sedangkan slicing yang dimulai dari indeks **x**, yang dimana nilai **x** adalah lebih dari **0**, membuat elemen ke-**x** slice yang diambil menjadi elemen ke-0 slice baru. Hal inilah yang membuat kapasitas slice berubah.

15.6. Fungsi append()

Fungsi `append()` digunakan untuk menambahkan elemen pada slice. Elemen baru tersebut diposisikan setelah indeks paling akhir. Nilai balik fungsi ini adalah slice yang sudah ditambahkan nilai barunya. Contoh penggunaannya bisa dilihat di kode berikut.

```
var fruits = []string{"apple", "grape", "banana"}
var cFruits = append(fruits, "papaya")

fmt.Println(fruits) // ["apple", "grape", "banana"]
fmt.Println(cFruits) // ["apple", "grape", "banana", "papaya"]
```

Ada 3 hal yang perlu diketahui dalam penggunaan fungsi ini.

- Ketika jumlah elemen dan lebar kapasitas adalah sama (`len(fruits) == cap(fruits)`), maka elemen baru hasil `append()` merupakan referensi baru.
- Ketika jumlah elemen lebih kecil dibanding kapasitas (`len(fruits) < cap(fruits)`), elemen baru tersebut ditempatkan kedalam cakupan kapasitas, menjadikan semua elemen slice lain yang referensi-nya sama akan berubah nilainya.

Agar lebih jelas silakan perhatikan contoh berikut.

```

var fruits = []string{"apple", "grape", "banana"}
var bFruits = fruits[0:2]

fmt.Println(cap(bFruits)) // 3
fmt.Println(len(bFruits)) // 2

fmt.Println(fruits) // ["apple", "grape", "banana"]
fmt.Println(bFruits) // ["apple", "grape"]

var cFruits = append(bFruits, "papaya")

fmt.Println(fruits) // ["apple", "grape", "papaya"]
fmt.Println(bFruits) // ["apple", "grape"]
fmt.Println(cFruits) // ["apple", "grape", "papaya"]

```

Pada contoh di atas bisa dilihat, elemen indeks ke-2 slice `fruits` nilainya berubah setelah ada penggunaan keyword `append()` pada `bFruits`. Slice `bFruits` kapasitasnya adalah **3** sedang jumlah datanya hanya **2**. Karena `len(bFruits) < cap(bFruits)`, maka elemen baru yang dihasilkan, terdeteksi sebagai perubahan nilai pada referensi yang lama (referensi elemen indeks ke-2 slice `fruits`), membuat elemen yang referensinya sama, nilainya berubah.

15.7. Fungsi `copy()`

Fungsi `copy()` digunakan untuk men-copy elemen slice pada parameter ke-2, untuk digabungkan dengan slice pada parameter ke-1. Fungsi ini mengembalikan jumlah elemen yang berhasil di-copy (yang nilai tersebut merupakan nilai terkecil antara `len(sliceTarget)` dan `len(sliceTujuan)`). Berikut merupakan contoh penerapannya.

```

var fruits = []string{"apple"}
var aFruits = []string{"watermelon", "pinnacle"}

var copiedElemen = copy(fruits, aFruits)

fmt.Println(fruits)      // ["apple", "watermelon", "pinnacle"]
fmt.Println(aFruits)     // ["watermelon", "pinnacle"]
fmt.Println(copiedElemen) // 1

```

15.8. Pengaksesan Elemen Slice Dengan 3 Indeks

3 index adalah teknik slicing elemen yang sekaligus menentukan kapasitasnya. Cara menggunakaninya yaitu dengan menyisipkan angka kapasitas di belakang, seperti `fruits[0:1:1]`. Angka kapasitas yang diisikan tidak boleh melebihi kapasitas slice yang akan di slicing.

Berikut merupakan contoh penerapannya.

```
var fruits = []string{"apple", "grape", "banana"}  
var aFruits = fruits[0:2]  
var bFruits = fruits[0:2:2]  
  
fmt.Println(fruits)      // ["apple", "grape", "banana"]  
fmt.Println(len(fruits)) // len: 3  
fmt.Println(cap(fruits)) // cap: 3  
  
fmt.Println(aFruits)      // ["apple", "grape"]  
fmt.Println(len(aFruits)) // len: 2  
fmt.Println(cap(aFruits)) // cap: 3  
  
fmt.Println(bFruits)      // ["apple", "grape"]  
fmt.Println(len(bFruits)) // len: 2  
fmt.Println(cap(bFruits)) // cap: 2
```

16. Map

Map adalah tipe data asosiatif yang ada di Golang, berbentuk *key-value*. Untuk setiap data (atau value) yang disimpan, disiapkan juga key-nya. Key harus unik, karena digunakan sebagai penanda (atau identifier) untuk pengaksesan value yang bersangkutan.

Kalau dilihat, `map` mirip seperti slice, hanya saja indeks yang digunakan untuk pengaksesan bisa ditentukan sendiri tipe-nya (indeks tersebut adalah key).

16.1. Penggunaan Map

Cara menggunakan map cukup dengan menuliskan keyword `map` diikuti tipe data key dan value-nya. Agar lebih mudah dipahami, silakan perhatikan contoh di bawah ini.

```
var chicken map[string]int
chicken = map[string]int{}

chicken["januari"] = 50
chicken["februari"] = 40

fmt.Println("januari", chicken["januari"]) // januari 50
fmt.Println("mei",     chicken["mei"])      // mei 0
```

Variabel `chicken` dideklarasikan sebagai map, dengan tipe data key adalah `string` dan value-nya `int`. Dari kode tersebut bisa dilihat bagaimana cara penggunaan keyword `map`.

Kode `map[string]int` maknanya adalah, tipe data `map` dengan key bertipe `string` dan value bertipe `int`.

Default nilai variabel `map` adalah `nil`. Oleh karena itu perlu dilakukan inisialisasi nilai default di awal, caranya cukup dengan tambahkan kurung kurawal pada akhir tipe, contoh seperti pada kode di atas: `map[string]int{}`.

Cara menge-set nilai pada sebuah map adalah dengan menuliskan variabel-nya, kemudian disisipkan `key` pada kurung siku variabel (mirip seperti cara pengaksesan elemen slice), lalu isi nilainya. Contohnya seperti `chicken["februari"] = 40`. Sedangkan cara pengambilan value adalah cukup dengan menyisipkan `key` pada kurung siku variabel.

Pengisian data pada map bersifat **overwrite**, ketika variabel sudah memiliki item dengan key yang sama, maka value lama akan ditimpas dengan value baru.

```
[novalagung:belajar-golang $ go run bab16.go
januari 50
mei 0
novalagung:belajar-golang $ ]
```

Pada pengaksesan item menggunakan key yang belum tersimpan di map, akan dikembalikan nilai default tipe data value-nya. Contohnya seperti pada kode di atas, `chicken["mei"]` menghasilkan nilai 0 (nilai default tipe `int`), karena belum ada item yang tersimpan menggunakan key `"mei"`.

16.2. Inisialisasi Nilai Map

Nilai variabel bertipe map bisa didefinisikan di awal, caranya dengan menambahkan kurung kurawal setelah tipe data, lalu menuliskan key dan value didalamnya. Cara ini sekilas mirip dengan definisi nilai array/slice namun dalam bentuk key-value.

```
// cara vertikal
var chicken1 = map[string]int{"januari": 50, "februari": 40}

// cara horizontal
var chicken2 = map[string]int{
    "januari": 50,
    "februari": 40,
}
```

Key dan value dituliskan dengan pembatas tanda titik dua (`:`). Sedangkan tiap itemnya dituliskan dengan pembatas tanda koma (`,`). Khusus deklarasi dengan gaya vertikal, tanda koma perlu dituliskan setelah item terakhir.

Variabel `map` bisa di-inisialisasi dengan tanpa nilai awal, caranya menggunakan tanda kurung kurawal, contoh: `map[string]int{}``. Atau bisa juga dengan menggunakan keyword `make` dan `new`. Contohnya bisa dilihat pada kode berikut. Ketiga cara di bawah ini intinya adalah sama.

```
var chicken3 = map[string]int{}
var chicken4 = make(map[string]int)
var chicken5 = *new(map[string]int)
```

Khusus inisialisasi data menggunakan keyword `new`, yang dihasilkan adalah data pointer. Untuk mengambil nilai aslinya bisa dengan menggunakan tanda asterisk (`*`). Topik pointer akan dibahas lebih detail ketika sudah masuk bab 22.

16.3. Iterasi Item Map Menggunakan `for - range`

Item variabel `map` bisa di iterasi menggunakan `for - range`. Cara penerapannya masih sama seperti pada slice, pembedanya data yang dikembalikan di tiap perulangan adalah key dan value, bukan indeks dan elemen. Contohnya bisa dilihat pada kode berikut.

```
var chicken = map[string]int{
    "januari": 50,
    "februari": 40,
    "maret": 34,
    "april": 67,
}

for key, val := range chicken {
    fmt.Println(key, "\t:", val)
}
```

```
[novalagung:belajar-golang $ go run bab16.go
januari      : 50
februari     : 40
maret        : 34
april        : 67
novalagung:belajar-golang $ ]
```

16.4. Menghapus Item Map

Fungsi `delete()` digunakan untuk menghapus item dengan key tertentu pada variabel map. Cara penggunaannya, dengan memasukan objek map dan key item yang ingin dihapus sebagai parameter.

```
var chicken = map[string]int{"januari": 50, "februari": 40}

fmt.Println(len(chicken)) // 2
fmt.Println(chicken)

delete(chicken, "januari")

fmt.Println(len(chicken)) // 1
fmt.Println(chicken)
```

Item yang memiliki key `"januari"` dalam variabel `chicken` akan dihapus.

```
[novalagung:belajar-golang $ go run bab16.go
4 items : map[april:67 januari:50 februari:40 maret:34]
3 items : map[februari:40 maret:34 april:67]
novalagung:belajar-golang $ ]
```

Fungsi `len()` jika digunakan pada map akan mengembalikan jumlah item.

16.5. Deteksi Keberadaan Item Dengan Key Tertentu

Ada cara untuk mengetahui apakah dalam sebuah variabel map terdapat item dengan key tertentu atau tidak, yaitu dengan memanfaatkan 2 variabel sebagai penampung nilai kembalian pengaksesan item. Variabel ke-2 akan berisikan nilai `bool` yang menunjukkan ada atau tidaknya item yang dicari.

```
var chicken = map[string]int{"januari": 50, "februari": 40}
var value, isExist = chicken["mei"]

if isExist {
    fmt.Println(value)
} else {
    fmt.Println("item is not exists")
}
```

16.6. Kombinasi Slice & Map

Slice dan `map` bisa dikombinasikan, dan sering digunakan pada banyak kasus, contohnya seperti data array yang berisikan informasi siswa, dan banyak lainnya.

Cara menggunakannya cukup mudah, contohnya seperti `[]map[string]int`, artinya slice yang tipe tiap elemen-nya adalah `map[string]int`.

Agar lebih jelas, silakan praktikan contoh berikut.

```
var chickens = []map[string]string{
    map[string]string{"name": "chicken blue", "gender": "male"},
    map[string]string{"name": "chicken red", "gender": "male"},
    map[string]string{"name": "chicken yellow", "gender": "female"},
}

for _, chicken := range chickens {
    fmt.Println(chicken["gender"], chicken["name"])
}
```

Variabel `chickens` di atas berisikan informasi bertipe `map[string]string`, yang kebetulan tiap elemen memiliki 2 key yang sama.

Jika anda menggunakan versi go terbaru, cara deklarasi slice-map bisa dipersingkat, tipe tiap elemen tidak wajib untuk dituliskan.

```
var chickens = []map[string]string{
    {"name": "chicken blue", "gender": "male"},
    {"name": "chicken red", "gender": "male"},
    {"name": "chicken yellow", "gender": "female"},
}
```

Dalam `[]map[string]string`, tiap elemen bisa saja memiliki key yang berbeda-beda, sebagai contoh seperti kode berikut.

```
var data = []map[string]string{
    {"name": "chicken blue", "gender": "male", "color": "brown"},
    {"address": "mangga street", "id": "k001"},
    {"community": "chicken lovers"}
}
```

17. Fungsi

Fungi merupakan aspek penting dalam pemrograman. Definisi fungsi sendiri adalah sekumpulan blok kode yang dibungkus dengan nama tertentu. Penerapan fungsi yang tepat akan menjadikan kode lebih modular dan juga *dry* (kependekan dari *don't repeat yourself*), tak perlu menuliskan banyak kode yang kegunaannya berkali-kali, cukup sekali saja lalu panggil sesuai kebutuhan.

Di bab ini kita akan belajar tentang penggunaan fungsi di Golang.

17.1. Penerapan Fungsi

Sebenarnya tanpa sadar, kita sudah menerapkan fungsi di bab-bab sebelum ini, yaitu pada fungsi `main`. Fungsi `main` merupakan fungsi yang paling utama pada program Golang.

Cara membuat fungsi cukup mudah, yaitu dengan menuliskan keyword `func`, diikuti setelahnya nama fungsi, kurung yang berisikan parameter, dan kurung kurawal untuk membungkus blok kode.

Parameter sendiri adalah variabel yang disisipkan pada saat pemanggilan fungsi.

Silakan lihat dan praktikan kode tentang implementasi fungsi berikut.

```
package main

import "fmt"
import "strings"

func main() {
    var names = []string{"John", "Wick"}
    printMessage("halo", names)
}

func printMessage(message string, arr []string) {
    var nameString = strings.Join(arr, " ")
    fmt.Println(message, nameString)
}
```

Pada kode di atas, sebuah fungsi baru dibuat dengan nama `printMessage` memiliki 2 buah parameter yaitu string `message` dan slice string `arr`.

Fungsi tersebut dipanggil dalam `main`, dengan disisipkan 2 buah data sebagai parameter, data pertama adalah string `"hallo"` yang ditampung parameter `message`, dan parameter ke 2 adalah slice string `names` yang nilainya ditampung oleh parameter `arr`.

Di dalam `printMessage`, nilai `arr` yang merupakan slice string digabungkan menjadi sebuah string dengan pembatas adalah karakter **spasi**. Penggabungan slice dapat dilakukan dengan memanfaatkan fungsi `strings.Join()` (berada di dalam package `strings`).

```
[novalagung:belajar-golang $ go run bab17.go
halo John Wick
novalagung:belajar-golang $ ]
```

17.2. Fungsi Dengan Return Value / Nilai Balik

Sebuah fungsi bisa didesain tidak mengembalikan nilai balik (*void*), atau bisa mengembalikan suatu nilai. 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 = randomInRange(2, 10)
    fmt.Println("random number:", randomValue)
    randomValue = randomInRange(2, 10)
    fmt.Println("random number:", randomValue)
    randomValue = randomInRange(2, 10)
    fmt.Println("random number:", randomValue)
}

func randomInRange(min, max int) int {
    var value = rand.Int() % (max - min + 1) + min
    return value
}
```

Fungsi `randomInRange` bertugas untuk *generate* angka acak sesuai dengan range yang ditentukan, yang kemudian angka tersebut dijadikan nilai kembalian fungsi.

```
[novalagung:belajar-golang $ go run bab17.go
random number: 9
random number: 6
random number: 2
novalagung:belajar-golang $ ]
```

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 `randomInRange`.

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

Sedangkan cara untuk mengembalikan nilai itu sendiri 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.

Dari kode di atas mungkin ada beberapa hal yang belum pernah kita lakukan pada bab-bab sebelumnya, kita akan bahas satu-persatu.

17.3. Penggunaan Fungsi `rand.Seed()`

Fungsi ini diperlukan untuk memastikan bahwa angka random yang akan di-generate benar-benar acak. Kita bisa gunakan angka apa saja sebagai nilai parameter fungsi ini (umumnya diisi `time.Now().Unix()`).

```
rand.Seed(time.Now().Unix())
```

Fungsi `rand.Seed()` berada dalam package `math/rand`, yang harus di-import terlebih dahulu sebelum bisa dimanfaatkan.

Package `time` juga perlu di-import karena kita menggunakan fungsi `(time.Now().Unix())` disitu.

17.4. Import Banyak Package

Penulisan keyword `import` untuk banyak package bisa dilakukan dengan dua cara, dengan menuliskannya di tiap package, atau cukup sekali saja, bebas.

```
import "fmt"
import "math/rand"
import "time"

// atau

import (
    "fmt"
    "math/rand"
    "time"
)
```

17.5. Deklarasi Parameter Bertipe Data Sama

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

17.6. Penggunaan Keyword `return` Untuk Menghentikan Proses Dalam Fungsi

Selain sebagai penanda nilai balik, keyword `return` juga bisa dimanfaatkan untuk menghentikan proses dalam blok fungsi dimana ia dipakai. Contohnya bisa dilihat pada kode berikut.

```
package main

import "fmt"

func main() {
    divideNumber(10, 2)
    divideNumber(4, 0)
    divideNumber(8, -4)
}

func divideNumber(m, n int) {
    if n == 0 {
        fmt.Printf("invalid divider. %d cannot divided by %d\n", m, n)
        return
    }

    var res = m / n
    fmt.Printf("%d / %d = %d\n", m, n, res)
}
```

Fungsi `divideNumber` didesain tidak memiliki nilai balik. Fungsi ini dibuat untuk membungkus proses pembagian 2 bilangan, lalu menampilkan hasilnya.

Dalamnya terdapat proses validasi nilai variabel pembagi, jika nilainya adalah 0, maka akan ditampilkan pesan bahwa pembagian tidak bisa dilakukan, lalu proses dihentikan pada saat itu juga (dengan memanfaatkan keyword `return`). Jika nilai pembagi valid, maka proses pembagian diteruskan.

```
[novalagung:belajar-golang $ go run bab17.go
10 / 2 = 5
invalid divider. 4 cannot divided by 0
8 / -4 = -2
novalagung:belajar-golang $ ]
```

18. Fungsi Multiple Return

Umumnya fungsi hanya memiliki satu buah nilai balik saja. Jika ada kebutuhan dimana data yang dikembalikan harus banyak, biasanya digunakanlah tipe seperti `map`, `slice`, atau `struct` sebagai nilai balik.

Golang menyediakan kapabilitas bagi programmer untuk membuat fungsi memiliki banyak nilai balik. Di bab ini akan dibahas bagaimana penerapannya.

18.1 Penerapan Fungsi Multiple Return

Cara membuat fungsi yang memiliki banyak nilai balik tidaklah sulit. Tinggal tulis saja pada saat deklarasi fungsi semua tipe data nilai yang dikembalikan, dan pada keyword `return` tulis semua data yang ingin dikembalikan. Contoh bisa dilihat pada berikut.

```
package main

import "fmt"
import "math"

func calculate(d float64) (float64, float64) {
    // hitung luas
    var area = math.Pi * math.Pow(d / 2, 2)
    // hitung keliling
    var circumference = math.Pi * d

    // kembalikan 2 nilai
    return area, circumference
}
```

Fungsi `calculate()` di atas menerima satu buah parameter (`diameter`) yang digunakan dalam proses perhitungan. Di dalam fungsi tersebut ada 2 hal yang dihitung, yaitu nilai **keliling** dan **lingkaran**. Kedua nilai tersebut kemudian dijadikan sebagai return value fungsi.

Cara pendefinisian banyak nilai balik bisa dilihat pada kode di atas, langsung tulis tipe data semua nilai balik dipisah tanda koma, lalu ditambahkan kurung diantaranya.

```
func calculate(d float64) (float64, float64)
```

Tak lupa di bagian penulisan keyword `return` harus dituliskan juga semua data yang dijadikan nilai balik (dengan pemisah tanda koma).

```
return area, circumference
```

Implementasi dari fungsi `calculate()` di atas, bisa dilihat pada kode berikut.

```
func main() {
    var diameter float64 = 15
    var area, circumference = calculate(diameter)

    fmt.Printf("luas lingkaran\t\t: %.2f \n", area)
    fmt.Printf("keliling lingkaran\t: %.2f \n", circumference)
}
```

Output program:

```
[novalagung:belajar-golang $ go run bab18.go
luas lingkaran      : 176.71
keliling lingkaran : 47.12
novalagung:belajar-golang $ ]
```

Karena fungsi tersebut memiliki banyak nilai balik, maka pada pemanggilannya harus disiapkan juga banyak variabel untuk menampung nilai kembalian yang ada (sesuai jumlah nilai balik fungsi).

```
var area, circumference = calculate(diameter)
```

18.2 Fungsi Dengan Predefined Return Value

Keunikan lainnya yang jarang ditemui di bahasa lain adalah, di Golang variabel yang digunakan sebagai nilai balik bisa didefinisikan di-awal.

```
func calculate(d float64) (area float64, circumference float64) {
    area = math.Pi * math.Pow(d / 2, 2)
    circumference = math.Pi * d

    return
}
```

Fungsi `calculate` kita modif menjadi lebih sederhana. Bisa dilihat di kode di atas, ada cukup banyak perbedaan dibanding fungsi `calculate` sebelumnya. Perhatikan kode berikut.

```
func calculate(d float64) (area float64, circumference float64) {
```

Fungsi dideklarasikan memiliki 2 buah tipe data, dan variabel yang nantinya dijadikan nilai balik juga dideklarasikan. Variabel `area` yang bertipe `float64`, dan `circumference` bertipe `float64`.

Karena variabel nilai balik sudah ditentukan di awal, untuk mengembalikan nilai cukup dengan memanggil `return` tanpa perlu diikuti variabel apapun. Nilai terakhir `area` dan `circumference` sebelum pemanggilan keyword `return` adalah hasil dari fungsi di atas.

Ada beberapa hal baru dari kode di atas yang perlu dibahas, seperti `math.Pow()` dan `math.Pi`. Berikut adalah penjelasannya.

18.3. Penggunaan Fungsi `math.Pow()`

Fungsi `math.Pow()` digunakan untuk memangkat nilai. `math.Pow(2, 3)` berarti 2 pangkat 3, hasilnya 8. Fungsi ini berada dalam package `math`.

18.4. Penggunaan Konstanta `math.Pi`

`math.Pi` adalah konstanta bawaan package `math` yang merepresentasikan **Pi** atau **22/7**.

19. 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 dari parameter-parameter yang disisipkan bertipe data sama, dan ditampung oleh sebuah variabel saja. Cara pengaksesan tiap datanya juga sama, dengan menggunakan index.

Di bab ini kita akan belajar mengenai cara penerapan fungsi variadic.

19.1. Penerapan Fungsi Variadic

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

Berikut merupakan contoh peniterepannya.

```
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
}
```

Output program:

```
[novalagung:belajar-golang $ go run bab19.go
Rata-rata : 3.70
novalagung:belajar-golang $ ]
```

Bisa dilihat pada fungsi `calculate()`, parameter `numbers` dideklarasikan dengan disisipkan tanda 3 titik (`...`), menandakan bahwa `numbers` adalah sebuah parameter variadic dengan tipe data `int`.

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

Pemanggilan fungsi dilakukan seperti biasa, hanya saja jumlah parameter yang disisipkan bisa banyak.

```
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 {
```

Berikut merupakan penjelasan tambahan dari kode yang telah kita tulis.

19.2. Penggunaan 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 kembalian `fmt.Sprintf()` ditampung oleh variabel `msg`.

Selain `fmt.Sprintf()`, ada juga `fmt.Sprint()` dan `fmt.Sprintln()`.

19.3. Penggunaan 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. Hampir semua jenis tipe data dasar yang telah dipelajari di bab 9 bisa digunakan untuk casting. Dan cara penerealannya juga sama, cukup panggil sebagai fungsi, lalu masukan data yang ingin dikonversi sebagai parameter.

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) di Golang hanya bisa dilakukan jika tipe datanya sejenis. Maka dari itulah perlu adanya casting ke tipe `float64` pada tiap operand.

19.4. Pengisian Parameter Fungsi Variadic Menggunakan Data Slice

Slice bisa digunakan sebagai parameter variadic. Caranya dengan menambahkan tanda titik tiga kali, tepat setelah nama variabel yang dijadikan parameter. Contohnya bisa dilihat pada kode berikut.

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

19.5. Fungsi Dengan Parameter Biasa & Variadic

Parameter variadic bisa dikombinasikan dengan parameter biasa, dengan syarat parameter variadic-nya harus diposisikan di akhir. Contohnya bisa dilihat pada kode berikut.

```
import "fmt"
import "strings"

func yourHobbies(name string, hobbies ...string) {
    var hobbiesAsString = strings.Join(hobbies, ", ")

    fmt.Printf("Hello, my name is: %s\n", name)
    fmt.Printf("My hobbies are: %s\n", hobbiesAsString)
}
```

Nilai parameter pertama fungsi `yourHobbies()` akan ditampung oleh `name`, sedangkan nilai parameter kedua dan seterusnya akan ditampung oleh `hobbies` sebagai slice.

Cara pemanggilannya masih sama seperti pada fungsi biasa.

```
func main() {
    yourHobbies("wick", "sleeping", "eating")
}
```

Jika parameter kedua dan seterusnya ingin diisi dengan data dari slice, maka gunakan tanda titik tiga kali.

```
func main() {
    var hobbies = []string{"sleeping", "eating"}
    yourHobbies("wick", hobbies...)
}
```

Output program:

```
[novalagung:belajar-golang $ go run bab19.go
Hello, my name is: wick
My hobbies are: sleeping, eating
novalagung:belajar-golang $ ]
```

20. 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.

20.1. 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)
```

Output program:

```
[novalagung:belajar-golang $ go run bab20.go  
data : [2 3 4 3 4 2 3]  
min  : 2  
max  : 4  
novalagung:belajar-golang $ ]
```

Berikut adalah penjelasan tambahan mengenai kode di atas

20.2. Penggunaan Template String %v

Template `%v` digunakan untuk menampilkan segala jenis data. Bisa array, int, float, bool, dan lainnya.

```
fmt.Printf("data : %v\nmin  : %v\nmax  : %v\n", numbers, min, max)
```

Bisa dilihat pada statement di atas, data bertipe array dan numerik ditampilkan menggunakan `%v`. Template ini biasa dimanfaatkan untuk menampilkan sebuah data yang tipe nya bisa dinamis atau belum diketahui. Sangat tepat jika digunakan pada data bertipe `interface{}` yang nantinya akan di bahas pada bab 27.

20.3. 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)
}
```

Output program:

```
[novalagung:belajar-golang $ go run bab20.go
original number : [2 3 0 4 3 2 0 4 2 0 3]
filtered number : [3 4 3 4 3]
novalagung:belajar-golang $ ]
```

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 di atas IIFE menghasilkan nilai balik yang kemudian ditampung `newNumber`. Perlu diperhatikan bahwa yang ditampung adalah **nilai kembalinya** bukan body fungsi atau **closure**.

Closure bisa juga dengan gaya manifest typing, caranya dengan menuliskan skema closure-nya sebagai tipe data. Contoh:

```
var closure (func (string, int, []string) int)
closure = func (a string, b int, c []string) int {
    // ...
}
```

20.4. 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 nilainya kembalinya 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
}
```

Fungsi tanpa nama yang akan dikembalikan boleh disimpan pada variabel terlebih dahulu. Contohnya:

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

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]
}
```

Output program:

```
[novalagung:belajar-golang $ go run bab20.go
numbers : [2 3 0 4 3 2 0 4 2 0 3]
find     : 3

found   : 9
value   : [2 3 0 3 2 0 2 0 3]
novalagung:belajar-golang $ ]
```

21. Fungsi Sebagai parameter

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

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

21.1. 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 dataContains0 = filter(data, func(each string) bool {
        return strings.Contains(each, "o")
    })
    var dataLength5 = 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:", dataContains0)
    // filter ada huruf "o" : [jason]

    fmt.Println("filter jumlah huruf \"5\"\t:", dataLength5)
    // filter jumlah huruf "5" : [jason ethan]
}

```

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()`.

```
[novalagung:belajar-golang $ go run bab21.go
data asli          : [wick jason ethan]
filter ada huruf "o"   : [jason]
filter jumlah huruf "5" : [jason ethan]
novalagung:belajar-golang $ ]
```

Pada `dataContains0`, 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.

21.2. 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`.

Di bawah ini merupakan penjelasan tambahan mengenai fungsi `strings.Contains()`.

21.3. 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"`.

22. Pointer

Pointer adalah *reference* atau alamat memory. Variabel pointer berarti variabel yang berisi alamat memori suatu nilai. Sebagai contoh sebuah variabel bertipe integer memiliki nilai **4**, maka yang dimaksud pointer adalah **alamat memori dimana nilai 4 disimpan**, bukan nilai 4 nya sendiri.

Variabel-variael yang memiliki *reference* atau alamat pointer yang sama, saling berhubungan satu sama lain dan nilainya pasti sama. Ketika ada perubahan nilai, maka akan memberikan efek kepada variabel lain (yang referensi-nya sama) yaitu nilainya ikut berubah.

22.1. Penerapan Pointer

Variabel bertipe pointer ditandai dengan adanya tanda **asterisk** (`*`) tepat sebelum penulisan tipe data ketika deklarasi.

```
var number *int
var name *string
```

Nilai default variabel pointer adalah `nil` (kosong). Variabel pointer tidak bisa menampung nilai yang bukan pointer, dan sebaliknya variabel biasa tidak bisa menampung nilai pointer.

Variabel biasa sebenarnya juga bisa diambil nilai pointernya, caranya dengan menambahkan tanda **ampersand** (`&`) tepat sebelum nama variabel. Metode ini disebut dengan **referencing**.

Dan sebaliknya, nilai asli variabel pointer juga bisa diambil, dengan cara menambahkan tanda **asterisk** (`*`) tepat sebelum nama variabel. Metode ini disebut dengan **dereferencing**.

OK, langsung saja kita praktikan. Berikut adalah contoh penerapan pointer.

```
var numberA int = 4
var numberB *int = &numberA

fmt.Println("numberA (value)    :", numberA) // 4
fmt.Println("numberA (address) :", &numberA) // 0xc20800a220

fmt.Println("numberB (value)    :", *numberB) // 4
fmt.Println("numberB (address) :", numberB) // 0xc20800a220
```

Variabel `numberB` dideklarasikan bertipe pointer `int` dengan nilai awal adalah referensi variabel `numberA` (bisa dilihat pada kode `&numberA`). Dengan ini, variabel `numberA` dan `numberB` menampung data dengan referensi alamat memori yang sama.

```
[novalagung:belajar-golang $ go run bab22.go
numberA (value) : 4
numberA (address) : 0xc20800a220
numberB (value) : 4
numberB (address) : 0xc20800a220
novalagung:belajar-golang $ ]
```

Variabel pointer jika di-print akan menghasilkan string alamat memori (dalam notasi heksadesimal), contohnya seperti `numberB` yang diprint menghasilkan `0xc20800a220`.

Nilai asli sebuah variabel pointer bisa didapatkan dengan cara di-dereference terlebih dahulu (bisa dilihat pada kode `*numberB`).

22.2. Efek Perubahan Nilai Pointer

Ketika salah satu variabel pointer di ubah nilainya, sedang ada variabel lain yang memiliki referensi memori yang sama, maka nilai variabel lain tersebut juga akan berubah.

```
var numberA int = 4
var numberB *int = &numberA

fmt.Println("numberA (value) : ", numberA)
fmt.Println("numberA (address) : ", &numberA)
fmt.Println("numberB (value) : ", *numberB)
fmt.Println("numberB (address) : ", numberB)

fmt.Println("")

numberA = 5

fmt.Println("numberA (value) : ", numberA)
fmt.Println("numberA (address) : ", &numberA)
fmt.Println("numberB (value) : ", *numberB)
fmt.Println("numberB (address) : ", numberB)
```

Variabel `numberA` dan `numberB` memiliki referensi memori yang sama. Perubahan pada salah satu nilai variabel tersebut akan memberikan efek pada variabel lainnya. Pada contoh di atas, `numberA` nilainya di ubah menjadi `5`. membuat nilai asli variabel `numberB` ikut berubah menjadi `5`.

```
[novalagung:belajar-golang $ go run bab22.go
numberA (value) : 4
numberA (address) : 0xc20800a220
numberB (value) : 4
numberB (address) : 0xc20800a220

numberA (value) : 5
numberA (address) : 0xc20800a220
numberB (value) : 5
numberB (address) : 0xc20800a220
novalagung:belajar-golang $ ]
```

22.3. Parameter Pointer

Parameter bisa juga didesain sebagai pointer. Cara penerapannya kurang lebih sama, dengan cara mendeklarasikan parameter sebagai pointer.

```
package main

import "fmt"

func main() {
    var number = 4
    fmt.Println("before :", number) // 4

    change(&number, 10)
    fmt.Println("after  :", number) // 10
}

func change(original *int, value int) {
    *original = value
}
```

Fungsi `change()` memiliki 2 parameter, yaitu `original` yang tipenya adalah pointer `int`, dan `value` yang bertipe `int`. Di dalam fungsi tersebut nilai asli parameter pointer `original` diubah.

Fungsi `change()` kemudian diimplementasikan di `main`. Variabel `number` yang nilai awalnya adalah `4` diambil referensi-nya lalu digunakan sebagai parameter pada pemanggilan fungsi `change()`.

Nilai variabel `number` berubah menjadi `10` karena perubahan yang terjadi di dalam fungsi `change` adalah pada variabel pointer.

```
[novalagung:belajar-golang $ go run bab22.go
before : 4
after  : 10
novalagung:belajar-golang $ ]
```


23. Struct

Struct adalah kumpulan definisi variabel (atau property) dan atau fungsi (atau method), yang dibungkus dengan nama tertentu.

Property dalam struct, tipe datanya bisa bervariasi. Mirip seperti `map`, hanya saja key-nya sudah didefinisikan di awal, dan tipe data tiap itemnya bisa berbeda.

Dengan memanfaatkan struct, data akan terbungkus lebih rapi dan mudah di-maintain.

Struct merupakan cetakan, digunakan untuk mencetak variabel objek (istilah untuk variabel yang memiliki property). Variabel objek memiliki behaviour atau sifat yang sama sesuai struct pencetaknya. Konsep ini sama dengan konsep **class** pada pemrograman berbasis objek. Sebuah buah struct bisa dimanfaatkan untuk mencetak banyak objek.

Disini penulis menggunakan konsep OOP sebagai analogi, dengan tujuan untuk mempermudah dalam mencerna isi bab ini.

23.1. Deklarasi Struct

Keyword `type` digunakan untuk deklarasi struct. Di bawah ini merupakan contoh cara penggunaannya.

```
type student struct {
    name string
    grade int
}
```

Struct `student` dideklarasikan memiliki 2 property, yaitu `name` dan `grade`. Objek yang dicetak dengan struct ini nantinya akan memiliki sifat yang sama.

23.2. Penerapan Struct

Struct `student` yang sudah disiapkan di atas akan kita manfaatkan untuk mencetak variabel objek. Property variabel tersebut di-isi kemudian ditampilkan.

```
func main() {
    var s1 student
    s1.name = "john wick"
    s1.grade = 2

    fmt.Println("name : ", s1.name)
    fmt.Println("grade : ", s1.grade)
}
```

Cara membuat variabel objek sama seperti pembuatan variabel biasa. Tinggal tulis saja nama variabel diikuti nama struct, contoh: `var s1 student`.

Semua property variabel objek pada awalnya memiliki nilai default sesuai tipe datanya.

Property variabel objek bisa diakses nilainya menggunakan notasi titik, contohnya `s1.name`. Nilai property-nya juga bisa diubah, contohnya `s1.grade = 2`.

```
[novalagung:belajar-golang $ go run bab23.go
name : john wick
grade : 2
novalagung:belajar-golang $ ]
```

23.3. Inisialisasi Object Struct

Cara inisialisasi variabel objek adalah dengan menambahkan kurung kurawal setelah nama struct. Nilai masing-masing property bisa diisi pada saat inisialisasi.

Pada contoh berikut, terdapat 3 buah variabel objek yang dideklarasikan dengan cara berbeda.

```
var s1 = student{}
s1.name = "wick"
s1.grade = 2

var s2 = student{"ethan", 2}

var s3 = student{name: "jason"}

fmt.Println("student 1 :", s1.name)
fmt.Println("student 2 :", s2.name)
fmt.Println("student 3 :", s3.name)
```

Pada kode di atas, variabel `s1` menampung objek cetakan `student`. Variabel tersebut kemudian di-set nilai property-nya.

Variabel objek `s2` dideklarasikan dengan metode yang sama dengan `s1`, pembedanya di `s2` nilai propertinya di isi langsung ketika deklarasi. Nilai pertama akan menjadi nilai property pertama (yaitu `name`), dan selanjutnya berurutan.

Pada deklarasi `s3`, dilakukan juga pengisian property ketika pencetakan objek. Hanya saja, yang diisi hanya `name` saja. Cara ini cukup efektif jika digunakan untuk membuat objek baru yang nilai property-nya tidak semua harus disiapkan di awal. Keistimewaan lain menggunakan cara ini adalah penentuan nilai property bisa dilakukan dengan tidak berurutan. Contohnya:

```
var s4 = student{name: "wayne", grade: 2}
var s5 = student{grade: 2, name: "bruce"}
```

23.4. Variabel Objek Pointer

Objek hasil cetakan struct bisa diambil nilai pointer-nya, dan bisa disimpan pada variabel objek yang bertipe struct pointer. Contoh penerapannya:

```
var s1 = student{name: "wick", grade: 2}

var s2 *student = &s1
fmt.Println("student 1, name :", s1.name)
fmt.Println("student 4, name :", s2.name)

s2.name = "ethan"
fmt.Println("student 1, name :", s1.name)
fmt.Println("student 4, name :", s2.name)
```

`s2` adalah variabel pointer hasil cetakan struct `student`. `s2` menampung nilai referensi `s1`, menjadikan setiap perubahan pada property variabel tersebut, akan juga berpengaruh pada variabel objek `s1`.

Meskipun `s2` bukan variabel asli, property nya tetap bisa diakses seperti biasa. Inilah keistimewaan property dalam objek pointer, tanpa perlu di-dereferensi nilai asli property tetap bisa diakses. Pengisian nilai pada property tersebut juga bisa langsung menggunakan nilai asli, contohnya seperti `s2.name = "ethan"`.

```
[novalagung:belajar-golang $ go run bab23.go
student 1, name : wick
student 4, name : wick
student 1, name : ethan
student 4, name : ethan
novalagung:belajar-golang $ ]
```

23.5. Embedded Struct

Embedded struct adalah mekanisme untuk menyimpan objek cetakan struct kedalam properti sebuah struct lain. Agar lebih mudah dipahami, mari kita bahas kode berikut.

```
package main

import "fmt"

type person struct {
    name string
    age   int
}

type student struct {
    grade int
    person
}

func main() {
    var s1 = student{}
    s1.name = "wick"
    s1.age = 21
    s1.grade = 2

    fmt.Println("name : ", s1.name)
    fmt.Println("age : ", s1.age)
    fmt.Println("age : ", s1.person.age)
    fmt.Println("grade : ", s1.grade)
}
```

Pada kode di atas, disiapkan struct `person` dengan properti yang tersedia adalah `name` dan `age`. Disiapkan juga struct `student` dengan property `grade`. Struct `person` di-embed kedalam struct `student`. Caranya cukup mudah, yaitu dengan menuliskan nama struct yang ingin di-embed ke dalam body struct target.

Embedded struct adalah **mutable**, nilai property-nya bisa diubah.

Khusus untuk properti yang bukan properti asli (properti turunan dari struct lain), bisa diakses dengan cara mengakses struct *parent*-nya terlebih dahulu, contohnya

`s1.person.age`. Nilai yang dikembalikan memiliki referensi yang sama dengan `s1.age`.

23.6. Embedded Struct Dengan Nama Property Yang Sama

Jika salah satu nama properti sebuah struct memiliki kesamaan dengan properti milik struct lain yang di-embed, maka pengaksesan property-nya harus dilakukan secara explisit atau jelas. Contoh bisa dilihat di kode berikut.

```
package main

import "fmt"

type person struct {
    name string
    age  int
}

type student struct {
    person
    age   int
    grade int
}

func main() {
    var s1 = student{}
    s1.name = "wick"
    s1.age = 21           // age of student
    s1.person.age = 22 // age of person

    fmt.Println(s1.name)
    fmt.Println(s1.age)
    fmt.Println(s1.person.age)
}
```

Struct `person` di-embed ke dalam struct `student`, dan kedua struct tersebut kebetulan salah satu nama property-nya ada yg sama, yaitu `age`. Cara mengakses property `age` milik struct `person` lewat objek struct `student`, adalah dengan menuliskan nama struct yg di-embed kemudian nama property-nya, contohnya: `s1.person.age = 22`.

23.7. Pengisian Nilai Sub-Struct

Pengisian nilai property sub-struct bisa dilakukan dengan langsung memasukkan variabel objek yang tercetak dari struct yang sama.

```
var p1 = person{name: "wick", age: 21}
var s1 = student{person: p1, grade: 2}

fmt.Println("name : ", s1.name)
fmt.Println("age  : ", s1.age)
fmt.Println("grade : ", s1.grade)
```

Pada deklarasi `s1`, property `person` diisi variabel objek `p1`.

23.8. Anonymous Struct

Anonymous struct adalah struct yang tidak dideklarasikan di awal, melainkan ketika dibutuhkan saja, langsung pada saat penciptaan objek. Teknik ini cukup efisien untuk pembuatan variabel objek yang struct nya hanya dipakai sekali.

```
package main

import "fmt"

type person struct {
    name string
    age  int
}

func main() {
    var s1 = struct {
        person
        grade int
    }()
    s1.person = person{"wick", 21}
    s1.grade = 2

    fmt.Println("name : ", s1.person.name)
    fmt.Println("age : ", s1.person.age)
    fmt.Println("grade : ", s1.grade)
}
```

Pada kode di atas, variabel `s1` langsung diisi objek anonymous struct yang memiliki property `grade`, dan property `person` yang merupakan embedded struct.

Salah satu aturan yang perlu diingat dalam pembuatan anonymous struct adalah, deklarasi harus diikuti dengan inisialisasi. Bisa dilihat pada `s1` setelah deklarasi struktur struct, terdapat kurung kurawal untuk inisialisasi objek. Meskipun nilai tidak diisikan di awal, kurung kurawal tetap harus ditulis.

```
// anonymous struct tanpa pengisian property
var s1 = struct {
    person
    grade int
} {}

// anonymous struct dengan pengisian property
var s2 = struct {
    person
    grade int
} {
    person: person{"wick", 21},
    grade: 2,
}
```

23.9. Kombinasi Slice & Struct

Slice dan `struct` bisa dikombinasikan seperti pada slice dan `map`, caranya pun mirip, cukup tambahkan tanda `[]` sebelum tipe data pada saat deklarasi.

```
type person struct {
    name string
    age int
}

var allStudents = []person{
    {name: "Wick", age: 23},
    {name: "Ethan", age: 23},
    {name: "Bourne", age: 22},
}

for _, student := range allStudents {
    fmt.Println(student.name, "age is", student.age)
}
```

23.10. Inisialisasi Slice Anonymous Struct

Anonymous struct bisa dijadikan sebagai tipe sebuah slice. Dan nilai awalnya juga bisa diinisialisasi langsung pada saat deklarasi. Berikut adalah contohnya:

```

var allStudents = []struct {
    person
    grade int
}{

    {person: person{"wick", 21}, grade: 2},
    {person: person{"ethan", 22}, grade: 3},
    {person: person{"bond", 21}, grade: 3},
}

for _, student := range allStudents {
    fmt.Println(student)
}

```

23.11. Deklarasi Anonymous Struct Menggunakan Keyword var

Cara lain untuk deklarasi anonymous struct adalah dengan menggunakan keyword `var`.

```

var student struct {
    person
    grade int
}

student.person = person{"wick", 21}
student.grade = 2

```

Statement `type student struct` adalah contoh cara deklarasi struct. Maknanya akan berbeda ketika keyword `type` diganti `var`, seperti pada contoh di atas `var student struct`, yang artinya dicetak sebuah objek dari anonymous struct kemudian disimpan pada variabel bernama `student`.

Deklarasi anonymous struct menggunakan metode ini juga bisa dilakukan beserta inisialisasi-nya.

```

// hanya deklarasi
var student struct {
    grade int
}

// deklarasi sekaligus inisialisasi
var student = struct {
    grade int
} {
    12,
}

```

23.12. Nested struct

Nested struct adalah anonymous struct yang di-embed ke sebuah struct. Deklarasinya langsung didalam struct peng-embed. Contoh:

```
type student struct {
    person struct {
        name string
        age   int
    }
    grade   int
    hobbies []string
}
```

Teknik ini biasa digunakan ketika decoding data **json** yang strukturnya cukup kompleks dengan proses decode hanya sekali.

23.13. Deklarasi Dan Inisialisasi Struct Secara Horizontal

Deklarasi struct bisa dituliskan secara horizontal, caranya bisa dilihat pada kode berikut:

```
type person struct { name string; age int; hobbies []string }
```

Tanda semi-colon (;) digunakan sebagai pembatas deklarasi property yang dituliskan secara horizontal. Inisialisasi nilai juga bisa dituliskan dengan metode ini. Contohnya:

```
var p1 = struct { name string; age int } { age: 22, name: "wick" }
var p2 = struct { name string; age int } { "ethan", 23 }
```

Bagi pengguna editor Sublime yang terinstal plugin GoSublime didalamnya, cara ini tidak akan bisa dilakukan, karena setiap kali file di-save, kode program dirapikan. Jadi untuk mengetesnya bisa dengan menggunakan editor lain.

23.14. Tag property dalam struct

Tag merupakan informasi opsional yang bisa ditambahkan pada masing-masing property struct.

```
type person struct {
    name string `tag1`
    age  int    `tag2`
}
```

Tag biasa dimanfaatkan untuk keperluan encode/decode data json. Informasi tag juga bisa diakses lewat reflect. Nantinya akan ada pembahasan yang lebih detail mengenai pemanfaatan tag dalam struct, terutama ketika sudah masuk bab JSON.

24. Method

Method adalah fungsi yang menempel pada `struct`, sehingga hanya bisa diakses lewat variabel objek.

Keunggulan method dibanding fungsi biasa adalah memiliki akses ke property struct hingga level *private* (level akses nantinya akan dibahas lebih detail pada bab selanjutnya). Dan juga, dengan menggunakan method sebuah proses bisa di-enkapsulasi dengan baik.

24.1. Penerapan Method

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 variabel objek di sela-sela 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 bisa dilihat pada kode berikut.

```
func main() {
    var s1 = student{"john wick", 21}
    s1.sayHello()

    var name = s1.getNameAt(2)
    fmt.Println("nama panggilan :", name)
}
```

Output:

```
[novalagung:belajar-golang $ go run bab24.go
halo john wick
nama panggilan : wick
novalagung:belajar-golang $ ]
```

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 {
```

24.2. Method Pointer

Method pointer adalah method yang variabel objek pemilik method tersebut berupa pointer.

Kelebihan method jenis ini adalah, ketika kita melakukan manipulasi nilai pada property lain yang masih satu struct, nilai pada property tersebut akan di rubah pada reference nya. Lebih jelasnya perhatikan kode berikut.

```

package main

import "fmt"

type student struct {
    name string
    grade int
}

func (s student) changeName1(name string) {
    fmt.Println("---> on changeName1, name changed to", name)
    s.name = name
}

func (s *student) changeName2(name string) {
    fmt.Println("---> on changeName2, name changed to", name)
    s.name = name
}

func main() {
    var s1 = student{"john wick", 21}
    fmt.Println("s1 before", s1.name)
    // john wick

    s1.changeName1("jason bourne")
    fmt.Println("s1 after changeName1", s1.name)
    // john wick

    s1.changeName2("ethan hunt")
    fmt.Println("s1 after changeName2", s1.name)
    // ethan hunt
}

```

Output:

```

[novalagung:chapter-24 $ go run 2-method-pointer.go
s1 before john wick
---> on changeName1, name changed to jason bourne
s1 after changeName1 john wick
---> on changeName2, name changed to ethan hunt
s1 after changeName2 ethan hunt

```

Setelah eksekusi statement `s1.changeName1("jason bourne")`, nilai `s1.name` tidak berubah. Sebenarnya nilainya berubah tapi hanya dalam method `changeName1()` saja, nilai pada reference di objek-nya tidak berubah. Karena itulah ketika objek di print value dari `s1.name` tidak berubah.

Keistimewaan lain method pointer adalah, method itu sendiri bisa dipanggil dari objek pointer maupun objek biasa.

```
// pengaksesan method dari variabel objek biasa
var s1 = student{"john wick", 21}
s1.sayHello()

// pengaksesan method dari variabel objek pointer
var s2 = &student{"ethan hunt", 22}
s2.sayHello()
```

Berikut adalah penjelasan tambahan mengenai beberapa hal pada bab ini.

24.3. Penggunaan Fungsi `strings.Split()`

Di bab ini ada fungsi baru yang kita gunakan: `strings.Split()`. Fungsi ini berguna untuk memisah string menggunakan pemisah yang ditentukan sendiri. Hasilnya adalah array berisikan kumpulan substring.

```
strings.Split("ethan hunt", " ")
// ["ethan", "hunt"]
```

Pada contoh di atas, string `"ethan hunt"` dipisah menggunakan separator spasi `" "`. Maka hasilnya terbentuk array berisikan 2 data, `"ethan"` dan `"hunt"`.

24.4. Apakah `fmt.Println()` & `strings.Split()` Juga Merupakan Method ?

Setelah tahu apa itu method dan bagaimana penggunaannya, mungkin akan muncul di benak kita bahwa kode seperti `fmt.Println()`, `strings.Split()` dan lainnya-yang-berada-pada-package-lain juga merupakan method.

Tapi sayangnya **bukan**. `fmt` disitu bukanlah variabel objek, dan `Println()` bukan merupakan method-nya.

`fmt` adalah nama **package** yang di-import (bisa dilihat pada kode `import "fmt"`). Sedangkan `Println()` adalah **nama fungsi**. Untuk mengakses fungsi yang berada pada package lain, harus dituliskan nama package-nya. Hal ini berlaku juga di dalam package `main`. Jika ada fungsi dalam package `main` yang diakses dari package lain yang berbeda, maka penulisannya `main.NamaFungsi()`.

Lebih detailnya akan dibahas di bab selanjutnya.

25. Property Public Dan Private

Bab ini membahas mengenai *modifier* public dan private dalam Golang. Kapan sebuah struct, fungsi, atau method bisa diakses dari package lain dan kapan tidak.

25.1. Package Public & Private

Pengembangan aplikasi dalam *real development* pasti membutuhkan banyak sekali file program. Tidak mungkin dalam satu project semua file memiliki nama package `main`, biasanya akan dipisah sebagai package berbeda sesuai bagiannya.

Project folder selain berisikan file-file `.go` juga bisa berisikan folder. Di bahasa Golang, setiap satu folder atau subfolder adalah satu package, file-file yang ada didalamnya harus memiliki nama package yang berbeda dengan file di folder lain.

Dalam sebuah package, biasanya kita menulis sangat banyak komponen, entah itu fungsi, struct, variabel, atau lainnya. Komponen tersebut bisa leluasa digunakan dalam package yang sama. Contoh sederhananya seperti program yang telah kita praktikan di bab sebelum-sebelumnya, dalam package `main` ada banyak yang di-define: fungsi, variabel, closure, struct, dan lainnya; kesemuanya bisa langsung dimanfaatkan.

Jika dalam satu program terdapat lebih dari 1 package, atau ada package lain selain `main`, tidak bisa komponen dalam package lain tersebut diakses secara bebas dari `main`, karena tiap komponen memiliki hak akses.

Ada 2 jenis hak akses:

- Akses Public, menandakan komponen tersebut diperbolehkan untuk diakses dari package lain yang berbeda
- Akses Private, berarti komponen hanya bisa diakses dalam package yang sama, bisa dalam satu file saja atau dalam beberapa file yang masih 1 folder.

Penentuan hak akses yang tepat untuk tiap komponen sangatlah penting.

Di Golang cara menentukan level akses atau modifier sangat mudah, penandanya adalah **character case** karakter pertama nama fungsi, struct, variabel, atau lainnya. Ketika namanya diawali dengan huruf kapital menandakan bahwa modifier-nya public. Dan sebaliknya, jika diawali huruf kecil, berarti private.

25.2. Penggunaan Package, Import, Dan Modifier Public & Private

Agar lebih mudah dipahami, maka langsung saja kita praktikan.

Pertama buat folder proyek baru bernama `belajar-golang-level-akses` di dalam folder `$GOPATH/src`, didalamnya buat file baru bernama `main.go`, set nama package file tersebut sebagai **main**.

Selanjutnya buat folder baru bernama `library` di dalam folder baru yang sudah dibuat tadi. Didalamnya buat file baru `library.go`, set nama package-nya **library**.

▼	go		Today, 5:07 PM	--
►	bin		Today, 5:07 PM	--
►	pkg		Today, 5:07 PM	--
▼	src		Today, 5:09 PM	--
►	belajar-golang-level-akses		Today, 5:11 PM	--
►	library		Today, 5:11 PM	--
	library.go		Today, 5:02 PM	Zero bytes
	main.go		Today, 5:02 PM	Zero bytes

Buka file `library.go` lalu isi dengan kode berikut.

```
package library

import "fmt"

func SayHello() {
    fmt.Println("hello")
}

func introduce(name string) {
    fmt.Println("nama saya", name)
}
```

File `library.go` yang telah dibuat ditentukan nama package-nya adalah `library` (sesuai dengan nama folder), berisi dua buah fungsi, `SayHello()` dan `introduce()`.

- Fungsi `SayHello()`, level aksesnya adalah publik, ditandai dengan nama fungsi diawali huruf besar.
- Fungsi `introduce()` dengan level akses private, ditandai oleh huruf kecil di awal nama fungsi.

Selanjutnya kita lakukan tes apakah memang fungsi yang ber-modifier private dalam package `library` tidak bisa diakses dari package lain.

Buka file `main.go`, lalu tulis kode berikut.

```
package main

import "belajar-golang-level-akses/library"

func main() {
    library.SayHello()
    library.introduce("ethan")
}
```

Bisa dilihat bahwa package `library` yang telah dibuat tadi, di-import ke dalam package `main`.

Folder utama atau root folder dalam project yang sedang digarap adalah `belajar-golang-level-akses`, sehingga untuk import package lain yang merupakan subfolder, harus dituliskan lengkap path folder nya, seperti `"belajar-golang-level-akses/library"`.

Penanda root folder adalah, file di dalam folder tersebut package-nya `main`, dan file tersebut dijalankan menggunakan `go run`.

Perhatikan kode berikut.

```
library.SayHello()
library.introduce("ethan")
```

Cara pemanggilan fungsi yang berada dalam package lain adalah dengan menuliskan nama package target diikuti dengan nama fungsi menggunakan *dot notation* atau tanda titik, seperti `library.SayHello()` atau `library.introduce("ethan")`

OK, sekarang coba jalankan kode yang sudah disiapkan di atas, hasilnya error.

```
[novalagung:belajar-golang-level-akses $ go run main.go
# command-line-arguments
./main.go:7: cannot refer to unexported name library.introduce
./main.go:7: undefined: library.introduce
novalagung:belajar-golang-level-akses $ ]
```

Error di atas disebabkan karena fungsi `introduce()` yang berada dalam package `library` memiliki level akses **private**, fungsi ini tidak bisa diakses dari package lain (pada kasus ini `main`). Agar bisa diakses, solusinya bisa dengan menjadikannya public, atau diubah cara pemanggilannya. Disini kita menggunakan cara ke-2.

Tambahkan parameter `name` pada fungsi `SayHello()`, lalu panggil fungsi `introduce()` dengan menyisipkan parameter `name` dari dalam fungsi `SayHello()`.

```
func SayHello(name string) {
    fmt.Println("hello")
    introduce(name)
}
```

Di `main`, cukup panggil fungsi `SayHello()` saja, sisipkan sebuah string sebagai parameter.

```
func main() {
    library.SayHello("ethan")
}
```

Coba jalankan lagi.

```
[novalagung:belajar-golang-level-akses $ go run main.go
hello
nama saya ethan
novalagung:belajar-golang-level-akses $ ]
```

26.3. Penggunaan Public & Private Pada Struct Dan Propertinya

Level akses private dan public juga bisa diterapkan di fungsi, struct, method, maupun property variabel. Cara penggunaannya sama seperti pada pembahasan sebelumnya, yaitu dengan menentukan **character case** huruf pertama nama komponen, apakah huruf besar atau kecil.

Belajar tentang level akses di Golang akan lebih cepat jika langsung praktik. Oleh karena itu langsung saja. Hapus isi file `library.go`, buat struct baru dengan nama `student` didalamnya.

```
package library

type student struct {
    Name string
    grade int
}
```

Buat contoh sederhana penerapan struct di atas pada file `main.go`.

```

package main

import "belajar-golang-level-akses/library"
import "fmt"

func main() {
    var s1 = library.student{"ethan", 21}
    fmt.Println("name ", s1.Name)
    fmt.Println("grade", s1.grade)
}

```

Setelah itu jalankan program.

```

[novalagung:belajar-golang-level-akses $ go run main.go
# command-line-arguments
./main.go:7: cannot refer to unexported name library.student
./main.go:7: undefined: library.student
novalagung:belajar-golang-level-akses $

```

Error muncul lagi, kali ini penyebabnya adalah karena struct `student` masih di set sebagai private. Ganti menjadi public (dengan cara mengubah huruf awalnya menjadi huruf besar) lalu jalankan.

```

// pada library/library.go
type Student struct {
    Name string
    grade int
}

// pada main.go
var s1 = library.Student{"ethan", 21}
fmt.Println("name ", s1.Name)
fmt.Println("grade", s1.grade)

```

Output:

```

[novalagung:belajar-golang-level-akses $ go run main.go
# command-line-arguments
./main.go:7: implicit assignment of unexported field 'grade' in library.Student
literal
novalagung:belajar-golang-level-akses $

```

Error masih tetap muncul, tapi kali ini berbeda. Error yang baru ini disebabkan karena salah satu properti dari struct `Student` bermodifier private. Properti yg dimaksud adalah `grade`. Ubah menjadi public, lalu jalankan lagi.

```
// pada library/library.go
type Student struct {
    Name string
    Grade int
}

// pada main.go
var s1 = library.Student{"ethan", 21}
fmt.Println("name ", s1.Name)
fmt.Println("grade", s1.Grade)
```

Dari contoh program di atas, bisa disimpulkan bahwa untuk menggunakan `struct` yang berada di package lain, selain nama `struct`-nya harus bermodifier public, properti yang diakses juga harus public.

```
[novalagung:belajar-golang-level-akses $ go run main.go
name ethan
grade 21
novalagung:belajar-golang-level-akses $ ]
```

26.4. Import Dengan Prefix Tanda Titik

Seperti yang kita tahu, untuk mengakses fungsi/struct/variabel yg berada di package lain, nama package nya perlu ditulis, contohnya seperti pada penggunaan penggunaan

`library.Student` dan `fmt.Println()`.

Di Golang, komponen yang berada di package lain yang di-import bisa dijadikan se-level dengan komponen package peng-import, caranya dengan menambahkan tanda titik (`.`) setelah penulisan keyword `import`. Maksud dari se-level disini adalah, semua properti di package lain yg di-import bisa diakses tanpa perlu menuliskan nama package, seperti ketika mengakses sesuatu dari file yang sama.

```
import (
    . "belajar-golang-level-akses/library"
    "fmt"
)

func main() {
    var s1 = Student{"ethan", 21}
    fmt.Println("name ", s1.Name)
    fmt.Println("grade", s1.Grade)
}
```

Pada kode di atas package `library` di-import menggunakan tanda titik. Dengan itu, pemanggilan `struct` `Student` tidak perlu dengan menuliskan nama package nya.

26.5. Pemanfaatan Alias Ketika Import Package

Fungsi yang berada di package lain bisa diakses dengan cara menuliskan nama-package diikuti nama fungsi-nya, contohnya seperti `fmt.Println()`. Package yang sudah di-import tersebut bisa diubah namanya dengan cara menggunakan alias pada saat import.

Contohnya bisa dilihat pada kode berikut.

```
import (
    f "fmt"
)

func main() {
    f.Println("Hello World!")
}
```

Pada kode di atas, package `fmt` di tentukan aliasnya adalah `f`, untuk mengakses `Println()` cukup dengan `f.Println()`.

26.6. Mengakses Properti Dalam File Yang Package-nya Sama

Jika properti yang ingin di akses masih dalam satu package tapi berbeda file, cara mengaksesnya bisa langsung dengan memanggil namanya. Hanya saja ketika eksekusi, file-file lain yang yang nama package-nya sama juga ikut dipanggil.

Langsung saja kita praktikan, buat file baru dalam `belajar-golang-level-akses` dengan nama `partial.go`.

▼	go		Today, 5:07 PM	--
►	bin		Today, 5:07 PM	--
►	pkg		Today, 5:07 PM	--
▼	src		Today, 5:09 PM	--
▼	belajar-golang-level-akses		Today, 5:46 PM	--
►	library		Today, 5:11 PM	--
	main.go		Today, 5:02 PM	Zero bytes
	partial.go		Today, 5:02 PM	Zero bytes

Tulis kode berikut pada file `partial.go`. File ini kita set package-nya **main** (sama dengan nama package file `main.go`).

```
package main

import "fmt"

func sayHello(name string) {
    fmt.Println("halo", name)
}
```

Hapus semua isi file `main.go`, lalu silakan tulis kode berikut.

```
package main

func main() {
    sayHello("ethan")
}
```

Sekarang terdapat 2 file berbeda (`main.go` dan `partial.go`) dengan package adalah sama, `main`. Pada saat **go build** atau **go run**, semua file dengan nama package `main` harus dituliskan sebagai argumen command.

```
$ go run main.go partial.go
```

Fungsi `sayHello` pada file `partial.go` bisa dikenali meski level aksesnya adalah private. Hal ini karena kedua file tersebut (`main.go` dan `partial.go`) memiliki package yang sama.

Cara lain untuk menjalankan program bisa dengan perintah `go run *.go`, dengan cara ini tidak perlu menuliskan nama file nya satu per satu.

```
[novalagung:belajar-golang-level-akses $ go run main.go partial.go
halo ethan
novalagung:belajar-golang-level-akses $ ]
```

26.7. Fungsi `init()`

Selain fungsi `main()`, terdapat juga fungsi spesial, yaitu `init()`. Fungsi ini otomatis dipanggil pertama kali ketika package-dimana-fungsi-berada di-import.

Langsung saja kita praktikkan. Buka file `library.go`, hapus isinya lalu isi dengan kode berikut.

```

package library

import "fmt"

var Student = struct {
    Name string
    Grade int
}{{}

func init() {
    Student.Name = "John Wick"
    Student.Grade = 2

    fmt.Println("--> library/library.go imported")
}

```

Pada package tersebut, variabel `Student` dibuat dengan isi anonymous struct. Dalam fungsi `init()`, nilai `Name` dan `Grade` variabel di-set.

Selanjutnya buka file `main.go`, isi dengan kode berikut.

```

package main

import "belajar-golang-level-akses/library"
import "fmt"

func main() {
    fmt.Printf("Name : %s\n", library.Student.Name)
    fmt.Printf("Grade : %d\n", library.Student.Grade)
}

```

Package `library` di-import, dan variabel `student` dikonsumsi. Pada saat import package, fungsi `init()` yang berada didalamnya langsung dieksekusi.

Property variabel objek `Student` akan diisi dan sebuah pesan ditampilkan ke console.

Dalam sebuah package diperbolehkan ada banyak fungsi `init()` (urutan eksekusinya adalah sesuai file mana yg terlebih dahulu digunakan). Fungsi ini dipanggil sebelum fungsi `main()`, pada saat eksekusi program.

```

novalagung:belajar-golang-import-init $ go run main.go
--> library/library.go imported
Name : John Wick
Grade : 2
novalagung:belajar-golang-import-init $

```


26. Interface

Interface adalah kumpulan definisi method yang tidak memiliki isi (hanya definisi saja), dan dibungkus dengan nama tertentu.

Interface merupakan tipe data. Nilai objek bertipe interface default-nya adalah `nil`.

Interface mulai bisa digunakan jika sudah ada isinya, yaitu objek konkret yang memiliki definisi method minimal sama dengan yang ada di interface-nya.

26.1. Penerapan Interface

Yang pertama perlu dilakukan untuk menerapkan interface adalah menyiapkan interface beserta definisi method nya. Keyword `type` dan `interface` digunakan untuk pendefinisian interface.

```
package main

import "fmt"
import "math"

type hitung interface {
    luas() float64
    keliling() float64
}
```

Pada kode di atas, interface `hitung` memiliki 2 definisi method, `luas()` dan `keliling()`. Interface ini nantinya digunakan sebagai tipe data pada variabel, dimana variabel tersebut akan menampung menampung objek bangun datar hasil dari struct yang akan kita buat.

Dengan memanfaatkan interface `hitung`, perhitungan luas dan keliling bangun datar bisa dilakukan, tanpa perlu tahu jenis bangun datarnya sendiri itu apa.

Siapkan struct bangun datar `lingkaran`, struct ini memiliki method yang beberapa diantaranya terdefinisi di interface `hitung`.

```

type lingkaran struct {
    diameter float64
}

func (l lingkaran) jariJari() float64 {
    return l.diameter / 2
}

func (l lingkaran) luas() float64 {
    return math.Pi * math.Pow(l.jariJari(), 2)
}

func (l lingkaran) keliling() float64 {
    return math.Pi * l.diameter
}

```

Struct `lingkaran` di atas memiliki tiga method, `jariJari()`, `luas()`, dan `keliling()`.

Selanjutnya, siapkan struct bangun datar `persegi`.

```

type persegi struct {
    sisi float64
}

func (p persegi) luas() float64 {
    return math.Pow(p.sisi, 2)
}

func (p persegi) keliling() float64 {
    return p.sisi * 4
}

```

Perbedaan struct `persegi` dengan `lingkaran` terletak pada method `jariJari()`. Struct `persegi` tidak memiliki method tersebut. Tetapi meski demikian, variabel objek hasil cetakan 2 struct ini akan tetap bisa ditampung oleh variabel cetakan interface `hitung`, karena dua method yang ter-definisi di interface tersebut juga ada pada struct `persegi` dan `lingkaran`, yaitu `luas()` dan `keliling()`.

Buat implementasi perhitungan di `main`.

```

func main() {
    var bangunDatar hitung

    bangunDatar = persegi{10.0}
    fmt.Println("===== persegi")
    fmt.Println("luas      :", bangunDatar.luas())
    fmt.Println("keliling   :", bangunDatar.keliling())

    bangunDatar = lingkaran{14.0}
    fmt.Println("===== lingkaran")
    fmt.Println("luas      :", bangunDatar.luas())
    fmt.Println("keliling   :", bangunDatar.keliling())
    fmt.Println("jari-jari  :", bangunDatar.(lingkaran).jariJari())
}

```

Perhatikan kode di atas. Variabel objek `bangunDatar` bertipe interface `hitung`. Variabel tersebut digunakan untuk menampung objek konkret buatan struct `lingkaran` dan `persegi`.

Dari variabel tersebut, method `luas()` dan `keliling()` diakses. Secara otomatis GoLang akan mengarahkan pemanggilan method pada interface ke method asli milik struct yang bersangkutan.

```
[novalagung:belajar-golang $ go run bab26.go
===== persegi
luas      : 100
keliling   : 40
===== lingkaran
luas      : 153.93804002589985
keliling   : 43.982297150257104
jari-jari : 7
novalagung:belajar-golang $ ]
```

Method `jariJari()` pada struct `lingkaran` tidak akan bisa diakses karena tidak terdefinisi dalam interface `hitung`. Pengaksesannya dengan paksa akan menyebabkan error.

Untuk mengakses method yang tidak ter-definisi di interface, variabel-nya harus di-casting terlebih dahulu ke tipe asli variabel konkritisnya (pada kasus ini tipenya `lingkaran`), setelahnya method akan bisa diakses.

Cara casting objek interface sedikit unik, yaitu dengan menuliskan nama tipe tujuan dalam kurung, ditempatkan setelah nama interface dengan menggunakan notasi titik (seperti cara mengakses property, hanya saja ada tanda kurung nya). Contohnya bisa dilihat di kode berikut. Statement `bangunDatar.(lingkaran)` adalah contoh casting pada objek interface.

```

var bangunDatar hitung = lingkaran{14.0}
var bangunLingkaran lingkaran = bangunDatar.(lingkaran)

bangunLingkaran.jariJari()

```

Perlu diketahui juga, jika ada interface yang menampung objek konkret dimana struct-nya tidak memiliki salah satu method yang terdefinisi di interface, error juga akan muncul. Intinya kembali ke aturan awal, variabel interface hanya bisa menampung objek yang minimal memiliki semua method yang terdefinisi di interface-nya.

26.2. Embedded Interface

Interface bisa di-embed ke interface lain, sama seperti struct. Cara penerapannya juga sama, cukup dengan menuliskan nama interface yang ingin di-embed ke dalam interface tujuan.

Pada contoh berikut, disiapkan interface bernama `hitung2d` dan `hitung3d`. Kedua interface tersebut kemudian di-embed ke interface baru bernama `hitung`.

```
package main

import "fmt"
import "math"

type hitung2d interface {
    luas() float64
    keliling() float64
}

type hitung3d interface {
    volume() float64
}

type hitung interface {
    hitung2d
    hitung3d
}
```

Interface `hitung2d` berisikan method untuk kalkulasi luas dan keliling, sedang `hitung3d` berisikan method untuk mencari volume bidang. Kedua interface tersebut diturunkan di interface `hitung`, menjadikannya memiliki kemampuan untuk menghitung luas, keliling, dan volume.

Next, siapkan struct baru bernama `kubus` yang memiliki method `luas()`, `keliling()`, dan `volume()`.

```

type kubus struct {
    sisi float64
}

func (k *kubus) volume() float64 {
    return math.Pow(k.sisi, 3)
}

func (k *kubus) luas() float64 {
    return math.Pow(k.sisi, 2) * 6
}

func (k *kubus) keliling() float64 {
    return k.sisi * 12
}

```

Objek hasil cetakan struct `kubus` di atas, nantinya akan ditampung oleh objek cetakan interface `hitung` yang isinya merupakan gabungan interface `hitung2d` dan `hitung3d`.

Terakhhir, buat implementasi-nya di main.

```

func main() {
    var bangunRuang hitung = &kubus{4}

    fmt.Println("===== kubus")
    fmt.Println("luas      :", bangunRuang.luas())
    fmt.Println("keliling   :", bangunRuang.keliling())
    fmt.Println("volume     :", bangunRuang.volume())
}

```

Bisa dilihat di kode di atas, lewat interface `hitung`, method `luas`, `keliling`, dan `volume` bisa di akses.

Pada bab 24 dijelaskan bahwa method pointer bisa diakses lewat variabel objek biasa dan variabel objek pointer. Variabel objek yang dicetak menggunakan struct yang memiliki method pointer, jika ditampung kedalam variabel interface, harus diambil referensi-nya terlebih dahulu. Contohnya bisa dilihat pada kode di atas `var bangunRuang hitung = &kubus{4}`.

```
[novalagung:belajar-golang $ go run bab26.go
===== kubus
luas      : 96
keliling   : 48
volume     : 64
novalagung:belajar-golang $ ]
```


27. Interface Kosong

Interface kosong atau `interface{}` adalah tipe data yang sangat spesial. Variabel bertipe ini bisa menampung segala jenis data, bahkan array, bisa pointer bisa tidak (konsep ini disebut dengan **dynamic typing**).

27.1. Penggunaan `interface{}`

`interface{}` merupakan tipe data, sehingga cara penggunaannya sama seperti pada tipe data lainnya, hanya saja nilai yang diisikan bisa apa saja. Contoh:

```
package main

import "fmt"

func main() {
    var secret interface{}

    secret = "ethan hunt"
    fmt.Println(secret)

    secret = []string{"apple", "manggo", "banana"}
    fmt.Println(secret)

    secret = 12.4
    fmt.Println(secret)
}
```

Keyword `interface` seperti yang kita tau, digunakan untuk pembuatan interface. Tetapi ketika ditambahkan kurung kurawal (`{}`) di belakangnya (menjadi `interface{}`), maka kegunaannya akan berubah, yaitu sebagai tipe data.

```
[novalagung:belajar-golang $ go run bab27.go
data 1: ethan hunt
data 2: [apple manggo banana]
data 3: 12.4
novalagung:belajar-golang $ ]
```

Agar tidak bingung, coba perhatikan kode berikut.

```
var data map[string]interface{}

data = map[string]interface{}{
    "name":      "ethan hunt",
    "grade":     2,
    "breakfast": []string{"apple", "manggo", "banana"},
}
```

Pada kode di atas, disiapkan variabel `data` dengan tipe `map[string]interface{}`, yaitu sebuah koleksi dengan key bertipe `string` dan nilai bertipe interface kosong `interface{}`.

Kemudian variabel tersebut di-inisialisasi, ditambahkan lagi kurung kurawal setelah keyword deklarasi untuk kebutuhan pengisian data, `map[string]interface{}{ /* data */ }`.

Dari situ terlihat bahwa `interface{}` bukanlah sebuah objek, melainkan tipe data.

27.2. Casting Variabel Interface Kosong

Variabel bertipe `interface{}` bisa ditampilkan ke layar sebagai `string` dengan memanfaatkan fungsi `print`, seperti `fmt.Println()`. Tapi perlu diketahui bahwa nilai yang dimunculkan tersebut bukanlah nilai asli, melainkan bentuk `string` dari nilai aslinya.

Hal ini penting diketahui, karena untuk melakukan operasi yang membutuhkan nilai asli pada variabel yang bertipe `interface{}`, diperlukan casting ke tipe aslinya. Contoh seperti pada kode berikut.

```
package main

import "fmt"
import "strings"

func main() {
    var secret interface{}

    secret = 2
    var number = secret.(int) * 10
    fmt.Println(secret, "multiplied by 10 is :", number)

    secret = []string{"apple", "manggo", "banana"}
    var gruits = strings.Join(secret.([]string), ", ")
    fmt.Println(gruits, "is my favorite fruits")
}
```

Pertama, variabel `secret` menampung nilai bertipe numerik. Ada kebutuhan untuk mengalikan nilai yang ditampung variabel tersebut dengan angka `10`. Maka perlu dilakukan casting ke tipe aslinya, yaitu `int`, setelahnya barulah nilai bisa dioperasikan, yaitu `secret.(int) * 10`.

Pada contoh kedua, `secret` berisikan array string. Kita memerlukan string tersebut untuk digabungkan dengan pemisah tanda koma. Maka perlu di-casting ke `[]string` terlebih dahulu sebelum bisa digunakan di `strings.Join()`, contohnya pada `strings.Join(secret.([]string), ", ")`.

```
[novalagung:belajar-golang $ go run bab27.go
2 multiplied by 10 is : 20
apple, manggo, banana is my favorite fruits
novalagung:belajar-golang $ ]
```

Teknik casting pada interface disebut dengan **type assertions**.

27.3. Casting Variabel Interface Kosong Ke Objek Pointer

Variabel `interface{}` bisa menyimpan data apa saja, termasuk data objek, pointer, ataupun gabungan keduanya. Di bawah ini merupakan contoh penerapan interface untuk menampung data objek pointer.

```
type person struct {
    name string
    age int
}

var secret interface{} = &person{name: "wick", age: 27}
var name = secret.(*person).name
fmt.Println(name)
```

Variabel `secret` dideklarasikan bertipe `interface{}` menampung referensi objek cetakan `struct person`. Cara casting dari `interface{}` ke struct pointer adalah dengan menuliskan nama struct-nya dan ditambahkan tanda asterisk (`*`) di awal, contohnya seperti `secret.(*person)`. Setelah itu barulah nilai asli bisa diakses.

```
[novalagung:belajar-golang $ go run bab27.go
wick
novalagung:belajar-golang $ ]
```

27.4. Kombinasi Slice, map , dan interface{}

Tipe `[]map[string]interface{}` adalah salah satu tipe yang paling sering digunakan (menurut saya), karena tipe data tersebut bisa menjadi alternatif tipe slice struct.

Pada contoh berikut, variabel `person` dideklarasikan berisi data slice `map` berisikan 2 item dengan key adalah `name` dan `age`.

```
var person = []map[string]interface{}{
    {"name": "Wick", "age": 23},
    {"name": "Ethan", "age": 23},
    {"name": "Bourne", "age": 22},
}

for _, each := range person {
    fmt.Println(each["name"], "age is", each["age"])
}
```

Dengan memanfaatkan slice dan `interface{}`, kita bisa membuat data array yang isinya adalah bisa apa saja. Silakan perhatikan contoh berikut.

```
var fruits = []interface{}{
    map[string]interface{}{"name": "strawberry", "total": 10},
    []string{"manggo", "pineapple", "papaya"},
    "orange",
}

for _, each := range fruits {
    fmt.Println(each)
}
```

28. Reflect

Reflection adalah teknik untuk inspeksi variabel, mengambil informasi dari variabel tersebut atau bahkan memanipulasinya. Cakupan informasi yang bisa didapatkan lewat reflection sangat luas, seperti melihat struktur variabel, tipe, nilai pointer, dan banyak lagi.

Golang menyediakan package bernama `reflect`, berisikan banyak sekali fungsi untuk keperluan reflection. Di bab ini, kita akan belajar tentang dasar penggunaan package tersebut.

Dari banyak fungsi yang tersedia di dalam package tersebut, ada 2 fungsi yang paling penting untuk diketahui, yaitu `reflect.ValueOf()` dan `reflect.TypeOf()`.

- Fungsi `reflect.ValueOf()` akan mengembalikan objek dalam tipe `reflect.Value`, yang berisikan informasi yang berhubungan dengan nilai pada variabel yang dicari
- Sedangkan `reflect.TypeOf()` mengembalikan objek dalam tipe `reflect.Type`. Objek tersebut berisikan informasi yang berhubungan dengan tipe data variabel yang dicari

28.1. Mencari Tipe Data & Value Menggunakan Reflect

Dengan reflection, tipe data dan nilai variabel dapat diketahui dengan mudah. Contoh penerapannya bisa dilihat pada kode berikut.

```
package main

import "fmt"
import "reflect"

func main() {
    var number = 23
    var reflectValue = reflect.ValueOf(number)

    fmt.Println("tipe variabel :", reflectValue.Type())

    if reflectValue.Kind() == reflect.Int {
        fmt.Println("nilai variabel :", reflectValue.Int())
    }
}
```

```
[novalagung:belajar-golang $ go run bab28.go
    tipe variabel : int
    nilai variabel : 23
novalagung:belajar-golang $ ]
```

Fungsi `reflect.ValueOf()` memiliki parameter yang bisa menampung segala jenis tipe data. Fungsi tersebut mengembalikan objek dalam tipe `reflect.Value`, yang berisikan informasi mengenai variabel yang bersangkutan.

Objek `reflect.Value` memiliki beberapa method, salah satunya `Type()`. Method ini mengembalikan tipe data variabel yang bersangkutan dalam bentuk `string`.

Statement `reflectValue.Int()` menghasilkan nilai `int` dari variabel `number`. Untuk menampilkan nilai variabel reflect, harus dipastikan dulu tipe datanya. Ketika tipe data adalah `int`, maka bisa menggunakan method `Int()`. Ada banyak lagi method milik struct `reflect.Value` yang bisa digunakan untuk pengambilan nilai dalam bentuk tertentu, contohnya: `reflectValue.String()` digunakan untuk mengambil nilai `string`, `reflectValue.Float64()` untuk nilai `float64`, dan lainnya.

Perlu diketahui, fungsi yang digunakan harus sesuai dengan tipe data nilai yang ditampung variabel. Jika fungsi yang digunakan berbeda dengan tipe data variabelnya, maka akan menghasilkan error. Contohnya pada variabel menampung nilai bertipe `float64`, maka tidak bisa menggunakan method `String()`.

Diperlukan adanya pengecekan tipe data nilai yang disimpan, agar pengambilan nilai bisa tepat. Salah satunya bisa dengan cara seperti kode di atas, yaitu dengan mengecek dahulu apa jenis tipe datanya menggunakan method `Kind()`, setelah itu diambil nilainya dengan method yang sesuai.

Berikut adalah konstanta tipe data dan method yang bisa digunakan dalam refleksi di GoLang.

- Bool
- Int
- Int8
- Int16
- Int32
- Int64
- Uint
- Uint8
- Uint16
- Uint32
- Uint64
- Uintptr
- Float32

- Float64
- Complex64
- Complex128
- Array
- Chan
- Func
- Interface
- Map
- Ptr
- Slice
- String
- Struct
- UnsafePointer

Pengaksesan Nilai Dalam Bentuk `interface{}`

Jika nilai hanya diperlukan untuk ditampilkan ke output, bisa menggunakan `.Interface()`. Lewat method tersebut segala jenis nilai bisa diakses dengan mudah.

```
var number = 23
var reflectValue = reflect.ValueOf(number)

fmt.Println("tipe variabel :", reflectValue.Type())
fmt.Println("nilai variabel :", reflectValue.Interface())
```

Fungsi `Interface()` mengembalikan nilai interface kosong atau `interface{}`. Nilai aslinya sendiri bisa diakses dengan meng-casting interface kosong tersebut.

```
var nilai = reflectValue.Interface().(int)
```

28.2. Pengaksesan Informasi Property Variabel Objek

Reflect bisa digunakan untuk mengambil informasi semua property variabel objek cetakan struct, dengan catatan property-property tersebut bermodifier public. Langsung saja kita praktikan, siapkan sebuah struct bernama `student`.

```
type student struct {
    Name string
    Grade int
}
```

Buat method baru untuk struct tersebut, dengan nama method `getPropertyInfo()`. Method ini berisikan kode untuk mengambil dan menampilkan informasi tiap property milik struct `student`.

```
func (s *student) getPropertyInfo() {
    var reflectValue = reflect.ValueOf(s)

    if reflectValue.Kind() == reflect.Ptr {
        reflectValue = reflectValue.Elem()
    }

    var reflectType = reflectValue.Type()

    for i := 0; i < reflectValue.NumField(); i++ {
        fmt.Println("nama      :", reflectType.Field(i).Name)
        fmt.Println("tipe data :", reflectType.Field(i).Type)
        fmt.Println("nilai     :", reflectValue.Field(i).Interface())
        fmt.Println("")
    }
}
```

Terakhir, lakukan uji coba method di fungsi `main`.

```
func main() {
    var s1 = &student{Name: "wick", Grade: 2}
    s1.getPropertyInfo()
}
```

```
[novalagung:belajar-golang $ go run bab28.go
nama      : Name
tipe data : string
nilai     : wick

nama      : Grade
tipe data : int
nilai     : 2]
```

Dalam method `getPropertyInfo` terjadi beberapa hal. Pertama objek `reflect.Value` dari variabel `s` diambil. Setelah itu dilakukan pengecekan apakah variabel objek tersebut merupakan pointer atau tidak (bisa dilihat dari `if reflectValue.Kind() == reflect.Ptr`, jika bernilai `true` maka variabel adalah pointer). Jika ternyata variabel memang pointer, maka perlu diambil objek `reflect` aslinya dengan cara memanggil method `Elem()`.

Setelah itu, dilakukan perulangan sebanyak jumlah property yang ada pada struct `student`. Method `NumField()` akan mengembalikan jumlah property publik yang ada dalam struct.

Di tiap perulangan, informasi tiap property struct diambil berurutan dengan lewat method `Field()`. Method ini ada pada tipe `reflect.Value` dan `reflect.Type`.

- `reflectType.Field(i).Name` akan mengembalikan nama property
- `reflectType.Field(i).Type` mengembalikan tipe data property
- `reflectValue.Field(i).Interface()` mengembalikan nilai property dalam bentuk `interface{}`

Pengambilan informasi property, selain menggunakan indeks, bisa diambil berdasarkan nama field dengan menggunakan method `FieldByName()`.

28.3. Pengaksesan Informasi Method Variabel Objek

Informasi mengenai method juga bisa diakses lewat reflect, syaratnya masih sama seperti pada pengaksesan property, yaitu harus bermodifier public.

Pada contoh dibawah ini informasi method `SetName` akan diambil lewat reflection. Siapkan method baru di struct `student`, dengan nama `SetName`.

```
func (s *student) SetName(name string) {
    s.Name = name
}
```

Buat contoh penerapannya di fungsi `main`.

```
func main() {
    var s1 = &student{Name: "john wick", Grade: 2}
    fmt.Println("nama :", s1.Name)

    var reflectValue = reflect.ValueOf(s1)
    var method = reflectValue.MethodByName("SetName")
    method.Call([]reflect.Value{
        reflect.ValueOf("wick"),
    })

    fmt.Println("nama :", s1.Name)
}
```

```
[novalagung:belajar-golang $ go run bab28.go
  nama : john wick
  nama : wick
novalagung:belajar-golang $ ]
```

Pada kode di atas, disiapkan variabel `s1` yang merupakan instance struct `student`. Awalnya property `Name` variabel tersebut berisikan string `"john wick"`.

Setelah itu, refleksi nilai objek tersebut diambil, refleksi method `SetName` juga diambil. Pengambilan refleksi method dilakukan menggunakan `MethodByName` dengan argument adalah nama method yang diinginkan, atau bisa juga lewat indeks method-nya (menggunakan `Method(i)`).

Setelah refleksi method yang dicari sudah didapatkan, `call()` dipanggil untuk eksekusi method.

Jika eksekusi method diikuti pengisian parameter, maka parameternya harus ditulis dalam bentuk array `[]reflect.Value` berurutan sesuai urutan deklarasi parameter-nya. Dan nilai yang dimasukkan ke array tersebut harus dalam bentuk `reflect.Value` (gunakan `reflect.ValueOf()` untuk pengambilannya).

```
[]reflect.Value{
    reflect.ValueOf("wick"),
}
```

29. Goroutine

Goroutine mirip dengan thread thread, tapi sebenarnya bukan. Sebuah *native thread* bisa berisikan sangat banyak goroutine. Mungkin lebih pas kalau goroutine disebut sebagai **mini thread**. Goroutine sangat ringan, hanya dibutuhkan sekitar **2kB** memori saja untuk satu buah goroutine. Eksepsi goroutine bersifat *asynchronous*, menjadikannya tidak saling tunggu dengan goroutine lain.

Karena goroutine sangat ringan, maka eksekusi banyak goroutine bukan masalah.

Akan tetapi jika jumlah goroutine sangat banyak sekali (contoh 1 juta goroutine dijalankan pada komputer dengan RAM terbatas), memang proses akan jauh lebih cepat selesai, tapi memory / RAM juga bisa bengkak.

Goroutine merupakan salah satu bagian paling penting dalam Concurrent Programming di Golang. Salah satu yang membuat goroutine sangat istimewa adalah eksekusi-nya dijalankan di multi core processor. Kita bisa tentukan berapa banyak core yang aktif, makin banyak akan makin cepat.

Mulai bab 29 ini hingga bab 34, lalu dilanjut bab 56 dan 57, kita akan membahas tentang fitur-fitur yang disediakan golang untuk kebutuhan Concurrent Programming.

Concurrency atau konkurensi berbeda dengan paralel. Paralel adalah eksekusi banyak proses secara bersamaan. Sedangkan konkurensi adalah komposisi dari sebuah proses. Konkurensi merupakan struktur, sedangkan paralel adalah bagaimana eksekusinya berlangsung.

29.1. Penerapan Goroutine

Untuk menerapkan goroutine, proses yang akan dieksekusi sebagai goroutine harus dibungkus kedalam sebuah fungsi. Pada saat pemanggilan fungsi tersebut, ditambahkan keyword `go` didepannya, dengan itu goroutine baru akan dibuat dengan tugas adalah menjalankan proses yang ada dalam fungsi tersebut.

Berikut merupakan contoh implementasi sederhana tentang goroutine. Program di bawah ini menampilkan 10 baris teks, 5 dieksekusi dengan cara biasa, dan 5 lainnya dieksekusi sebagai goroutine baru.

```
package main

import "fmt"
import "runtime"

func print(till int, message string) {
    for i := 0; i < till; i++ {
        fmt.Println((i + 1), message)
    }
}

func main() {
    runtime.GOMAXPROCS(2)

    go print(5, "halo")
    print(5, "apa kabar")

    var input string
    fmt.Scanln(&input)
}
```

Pada kode di atas, Fungsi `runtime.GOMAXPROCS(n)` digunakan untuk menentukan jumlah core yang diaktifkan untuk eksekusi program.

Pembuatan goroutine baru ditandai dengan keyword `go`. Contohnya pada statement `go print(5, "halo")`, di situ fungsi `print()` dieksekusi sebagai goroutine baru.

Fungsi `fmt.Scanln()` mengakibatkan proses jalannya aplikasi berhenti di baris itu (**blocking**) hingga user menekan tombol enter. Hal ini perlu dilakukan karena ada kemungkinan waktu selesainya eksekusi goroutine `print()` lebih lama dibanding waktu selesainya goroutine utama `main()`, mengingat bahwa keduanya sama-sama asynchronous. Jika itu terjadi, goroutine yang belum selesai secara paksa dihentikan prosesnya karena goroutine utama sudah selesai dijalankan.

```
[novalagung:belajar-golang $ go run bab29.go
1 apa kabar
2 apa kabar
3 apa kabar
1 halo
4 apa kabar
2 halo
5 apa kabar
3 halo
4 halo
5 halo

[novalagung:belajar-golang $ go run bab29.go
1 apa kabar
1 halo
2 apa kabar
3 apa kabar
4 apa kabar
5 apa kabar
2 halo
3 halo
4 halo
5 halo]
```

Bisa dilihat di output, tulisan "halo" dan "apa kabar" bermunculan selang-seling. Ini disebabkan karena statement `print(5, "halo")` dijalankan sebagai goroutine baru, menjadikannya tidak saling tunggu dengan `print(5, "apa kabar")`.

Pada gambar di atas, program dieksekusi 2 kali. Hasil eksekusi pertama berbeda dengan kedua, penyebabnya adalah karena kita menggunakan 2 prosesor. Goroutine mana yang dieksekusi terlebih dahulu tergantung kedua prosesor tersebut.

Berikut adalah penjelasan tambahan tentang beberapa fungsi yang baru kita pelajari di atas.

29.2. Penggunaan Fungsi `runtime.GOMAXPROCS()`

Fungsi ini digunakan untuk menentukan jumlah core atau processor yang digunakan dalam eksekusi program.

Jumlah yang diinputkan secara otomatis akan disesuaikan dengan jumlah asli *logical processor* yang ada. Jika jumlahnya lebih, maka dianggap menggunakan sejumlah prosesor yang ada.

29.3. Penggunaan Fungsi `fmt.Scanln()`

Fungsi ini akan meng-capture semua karakter sebelum user menekan tombol enter, lalu menyimpannya pada variabel.

```
func Scanln(a ...interface{}) (n int, err error)
```

Kode di atas merupakan skema fungsi `fmt.Scanln()`. Fungsi tersebut bisa menampung parameter bertipe `interface{}` berjumlah tak terbatas. Tiap parameter akan menampung karakter-karakter inputan user yang sudah dipisah dengan tanda spasi. Agar lebih jelas, silakan perhatikan contoh berikut.

```
var s1, s2, s3 string
fmt.Scanln(&s1, &s2, &s3)

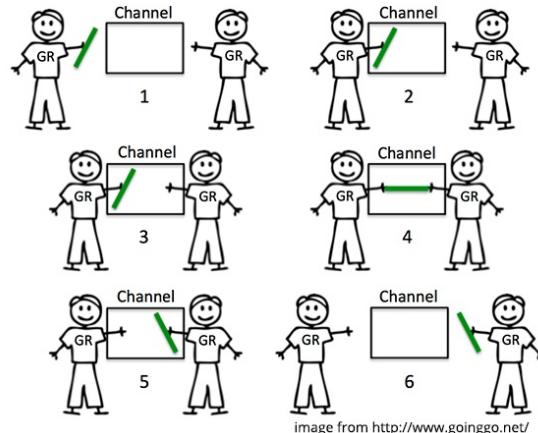
// user inputs: "trafalgar d law"

fmt.Println(s1) // trafalgar
fmt.Println(s2) // d
fmt.Println(s3) // law
```

Bisa dilihat pada kode di atas, untuk menampung inputan text `trafalgar d law`, dibutuhkan 3 buah variabel. Juga perlu diperhatikan bahwa yang disisipkan sebagai parameter pada pemanggilan fungsi `fmt.Scanln()` adalah referensi variabel, bukan nilai aslinya.

30. Channel

Channel digunakan untuk menghubungkan goroutine satu dengan goroutine lain. Mekanismenya adalah serah-terima data lewat channel tersebut. Goroutine pengirim dan penerima harus berada pada channel yang berbeda (konsep ini disebut **buffered channel**). Pengiriman dan penerimaan data pada channel bersifat **blocking** atau **synchronous**.



Pada bab ini kita akan belajar mengenai pemanfaatan channel.

30.1. Penerapan Channel

Channel merupakan sebuah variabel, dibuat dengan menggunakan keyword `make` dan `chan`. Variabel channel memiliki tugas menjadi pengirim dan penerima data.

Program berikut adalah contoh implementasi channel. 3 buah goroutine dieksekusi, di masing-masing goroutine terdapat proses pengiriman data lewat channel. Data tersebut akan diterima 3 kali di goroutine utama `main`.

```

package main

import "fmt"
import "runtime"

func main() {
    runtime.GOMAXPROCS(2)

    var messages = make(chan string)

    var sayHelloTo = func(who string) {
        var data = fmt.Sprintf("hello %s", who)
        messages <- data
    }

    go sayHelloTo("john wick")
    go sayHelloTo("ethan hunt")
    go sayHelloTo("jason bourne")

    var message1 = <-messages
    fmt.Println(message1)

    var message2 = <-messages
    fmt.Println(message2)

    var message3 = <-messages
    fmt.Println(message3)
}

```

Pada kode di atas, variabel `messages` dideklarasikan bertipe `channel string`. Cara pembuatan channel yaitu dengan menuliskan keyword `make` dengan isi keyword `chan` diikuti dengan tipe data channel yang diinginkan.

```
var messages = make(chan string)
```

Selain itu disiapkan juga closure `sayHelloTo` yang menghasilkan data string. Data tersebut dikirim lewat channel `messages`. Tanda `<-` jika dituliskan di sebelah kiri nama variabel, berarti sedang berlangsung proses pengiriman data dari variabel yang berada di kanan lewat channel yang berada di kiri (pada konteks ini, variabel `data` dikirim lewat channel `messages`).

```

var sayHelloTo = func(who string) {
    var data = fmt.Sprintf("hello %s", who)
    messages <- data
}

```

Fungsi `sayHelloTo` dieksekusi tiga kali sebagai goroutine berbeda. Menjadikan tiga proses ini berjalan secara **asynchronous** atau tidak saling tunggu.

```
go sayHelloTo("john wick")
go sayHelloTo("ethan hunt")
go sayHelloTo("jason bourne")
```

Dari ketiga fungsi tersebut, goroutine yang selesai paling awal akan mengirim data lebih dulu, datanya kemudian diterima variabel `message1`. Tanda `<-` jika dituliskan di sebelah kanan channel, menandakan proses penerimaan data dari channel yang di kanan, untuk disimpan ke variabel yang di kiri.

```
var message1 = <-messages
fmt.Println(message1)
```

Penerimaan channel bersifat blocking. Artinya statement `var message1 = <-messages` hingga setelahnya tidak akan dieksekusi sebelum ada data yang dikirim lewat channel.

Ketiga goroutine tersebut datanya akan diterima secara berurutan oleh `message1`, `message2`, `message3`; untuk kemudian ditampilkan.

```
[novalagung:belajar-golang $ go run bab30.go
hello wick
hello bourne
hello hunt
[novalagung:belajar-golang $ go run bab30.go
hello wick
hello hunt
hello bourne
[novalagung:belajar-golang $ go run bab30.go
hello hunt
hello bourne
hello wick
novalagung:belajar-golang $ ]]
```

Dari screenshot output di atas bisa dilihat bahwa text yang dikembalikan oleh `sayHelloTo` tidak selalu berurutan, meskipun penerimaan datanya adalah berurutan. Hal ini dikarenakan, pengiriman data adalah dari 3 goroutine yang berbeda, yang kita tidak tau mana yang prosesnya selesai lebih dulu. Goroutine yang dieksekusi lebih awal belum tentu selesai lebih awal, yang jelas proses yang selesai lebih awal datanya akan diterima lebih awal.

Karena pengiriman dan penerimaan data lewat channel bersifat **blocking**, tidak perlu memanfaatkan sifat blocking dari fungsi `fmt.Scanln()` untuk mengantisipasi goroutine utama `main` selesai sebelum 3 goroutine di atas selesai.

30.2. Channel Sebagai Tipe Data Parameter

Variabel channel bisa di-passing ke fungsi lain sebagai parameter. Caranya dengan menambahkan keyword `chan` ketika deklarasinya.

Siapkan fungsi `printMessage` dengan parameter adalah channel. Lalu ambil data yang dikirimkan lewat channel tersebut untuk ditampilkan.

```
func printMessage(what chan string) {
    fmt.Println(<-what)
}
```

Setelah itu ubah implementasi di fungsi `main`.

```
func main() {
    runtime.GOMAXPROCS(2)

    var messages = make(chan string)

    for _, each := range []string{"wick", "hunt", "bourne"} {
        go func(who string) {
            var data = fmt.Sprintf("hello %s", who)
            messages <- data
        }(each)
    }

    for i := 0; i < 3; i++ {
        printMessage(messages)
    }
}
```

Parameter `what` fungsi `printMessage` bertipe channel `string`, bisa dilihat dari kode `chan string` pada cara deklarasinya. Operasi serah-terima data akan bisa dilakukan pada variabel tersebut, dan akan berdampak juga pada variabel `messages` di fungsi `main`.

Passing data bertipe channel lewat parameter secara implisit adalah **pass by reference**, yang di-passing adalah pointer-nya. Output program di atas adalah sama dengan program sebelumnya.

```
[novalagung:belajar-golang $ go run bab30.go
hello hunt
hello bourne
hello wick
[novalagung:belajar-golang $ go run bab30.go
hello wick
hello bourne
hello hunt
novalagung:belajar-golang $ ]]
```

Berikut merupakan penjelasan tambahan kode di atas.

30.3. Iterasi Data Array Langsung Pada Saat Inisialisasi

Data array yang baru di-inisialisasi bisa langsung di-iterasi, caranya mudah dengan menuliskannya langsung setelah keyword `range`.

```
for _, each := range []string{"wick", "hunt", "bourne"} {
    // ...
}
```

30.4. Eksekusi Goroutine Pada IIFE

Eksekusi goroutine tidak harus pada fungsi atau closure yang sudah terdefinisi. Sebuah IIFE juga bisa dijalankan sebagai goroutine baru. Caranya dengan langsung menambahkan keyword `go` pada waktu deklarasi-eksekusi IIFE-nya.

```
var messages = make(chan string)

go func(who string) {
    var data = fmt.Sprintf("hello %s", who)
    messages <- data
}("wick")

var message = <-messages
fmt.Println(message)
```

31. Buffered Channel

Channel secara default adalah **un-buffered**, tidak di-buffer di memori. Ketika ada goroutine yang mengirimkan data lewat channel, harus ada goroutine lain yang bertugas menerima data dari channel yang sama, dengan proses serah-terima yang bersifat blocking.

Maksudnya, baris kode di bagian pengiriman dan penerimaan data, tidak akan akan diproses sebelum proses serah-terima-nya selesai.

Buffered channel sedikit berbeda. Pada channel jenis ini, ditentukan jumlah buffer-nya. Angka tersebut menjadi penentu jumlah data yang dikirimkan bersamaan. Selama jumlah data yang dikirim tidak melebihi jumlah buffer, maka pengiriman akan berjalan **asynchronous** (tidak blocking).

Ketika jumlah data yang dikirim sudah melewati batas buffer, maka pengiriman data hanya bisa dilakukan ketika salah satu data sudah diambil dari channel, sehingga ada slot channel yang kosong. Dengan proses penerimaan-nya sendiri bersifat blocking.

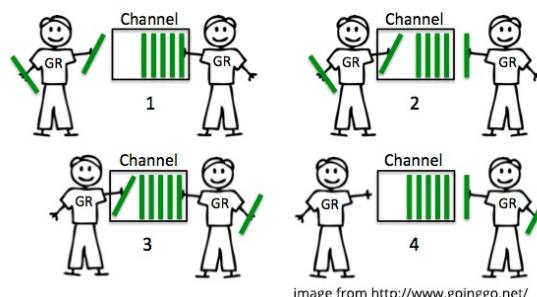


image from <http://www.goinggo.net/>

31.1. Penerapan Buffered Channel

Penerapan buffered channel pada dasarnya mirip seperti channel biasa. Perbedannya pada channel jenis ini perlu disiapkan jumlah buffer-nya.

Berikut adalah contoh penerapan buffered channel. Program dibawah ini merupakan pembuktian bahwa pengiriman data lewat buffered channel adalah asynchronous selama jumlah data yang sedang di-buffer oleh channel tidak melebihi kapasitas buffernya.

```

package main

import "fmt"
import "runtime"

func main() {
    runtime.GOMAXPROCS(2)

    messages := make(chan int, 2)

    go func() {
        for {
            i := <-messages
            fmt.Println("receive data", i)
        }
    }()

    for i := 0; i < 5; i++ {
        fmt.Println("send data", i)
        messages <- i
    }
}

```

Pada kode di atas, parameter kedua fungsi `make` adalah representasi jumlah buffer. Perlu diperhatikan bahwa nilai buffered channel dimulai dari `0`. Ketika nilainya adalah `2`, brarti jumlah buffer maksimal ada **3** (`0, 1, dan 2`).

Pada contoh di atas, terdapat juga sebuah goroutine yang berisikan proses penerimaan data dari channel message, yang selanjutnya akan ditampilkan.

Setelah goroutine untuk penerimaan data dieksekusi, data dikirimkan lewat perulangan `for`. Sejumlah 5 data akan dikirim lewat channel `message` secara sekuensial.

```

[novalagung:belajar-golang $ go run bab31.go
send data 0
send data 1
send data 2
send data 3
receive data 0
receive data 1
receive data 2
send data 4
receive data 3
receive data 4
[novalagung:belajar-golang $ go run bab31.go
send data 0
send data 1
send data 2
send data 3
receive data 0
receive data 1
receive data 2
send data 4
novalagung:belajar-golang $ ]

```

Bisa dilihat hasilnya pada output di atas. Pengiriman data ke-4, diikuti dengan penerimaan data, dan kedua proses tersebut berjalan secara blocking.

Pengiriman data ke 0, 1, 2 dan 3 akan berjalan secara asynchronous, hal ini karena channel ditentukan nilai buffer-nya sebanyak 3 (ingat, dimulai dari 0). Pengiriman selanjutnya (ke-4 dan ke-5) hanya akan terjadi jika ada salah satu data dari 4 data yang sebelumnya telah dikirimkan, sudah diterima (dengan serah terima data yang bersifat blocking). Setelahnya, sesudah slot channel ada yang kosong, serah-terima akan kembali asynchronous.

32. Channel - Select

Adanya channel memang sangat membantu pengontrolan goroutine, jumlah goroutine yang banyak bukan lagi masalah.

Fungsi utama channel bukan untuk mengontrol goroutine, melainkan untuk sharing data antar goroutine. Namun channel memang bisa digunakan untuk mengontrol goroutine.

Ada kalanya dimana kita butuh tak hanya satu channel saja untuk handle komunikasi data pada goroutine yang jumlahnya juga banyak, dibutuhkan beberapa atau mungkin banyak channel.

Disinilah kegunaan dari `select`. Select memudahkan pengontrolan komunikasi data lewat channel. Cara penggunaannya sama seperti seleksi kondisi `switch`.

32.1. Penerapan Keyword `select`

Program pencarian rata-rata dan nilai tertinggi berikut merupakan contoh sederhana penerapan `select` dalam channel. Akan ada 2 buah goroutine yang masing-masing di-handle oleh sebuah channel. Setiap kali goroutine selesai dieksekusi, akan dikirimkan datanya ke channel yang bersangkutan. Lalu dengan menggunakan `select`, akan diatur penerimaan datanya.

Pertama, kita siapkan terlebih dahulu 2 fungsi yang akan dieksekusi sebagai goroutine baru. Fungsi pertama digunakan untuk mencari rata-rata, dan fungsi kedua untuk penentuan nilai tertinggi dari sebuah slice.

```
package main

import "fmt"
import "runtime"

func getAverage(numbers []int, ch chan float64) {
    var sum = 0
    for _, e := range numbers {
        sum += e
    }
    ch <- float64(sum) / float64(len(numbers))
}

func getMax(numbers []int, ch chan int) {
    var max = numbers[0]
    for _, e := range numbers {
        if max < e {
            max = e
        }
    }
    ch <- max
}
```

Kedua fungsi di atas nantinya dijalankan sebagai goroutine baru. Di akhir masing-masing fungsi, dikirimkan data hasil komputasi ke channel yang sudah ditentukan (`ch1` menampung data rata-rata, `ch2` untuk data nilai tertinggi).

Buat implementasinya pada fungsi `main`.

```

func main() {
    runtime.GOMAXPROCS(2)

    var numbers = []int{3, 4, 3, 5, 6, 3, 2, 2, 6, 3, 4, 6, 3}
    fmt.Println("numbers :", numbers)

    var ch1 = make(chan float64)
    go getAverage(numbers, ch1)

    var ch2 = make(chan int)
    go getMax(numbers, ch2)

    for i := 0; i < 2; i++ {
        select {
        case avg := <-ch1:
            fmt.Printf("Avg \t: %.2f \n", avg)
        case max := <-ch2:
            fmt.Printf("Max \t: %d \n", max)
        }
    }
}

```

Pada kode di atas, transaksi pengiriman data pada channel `ch1` dan `ch2` dikontrol menggunakan `select`. Terdapat 2 buah `case` kondisi penerimaan data dari kedua channel tersebut.

- Kondisi `case avg := <-ch1` akan terpenuhi ketika ada penerimaan data dari channel `ch1`, yang kemudian akan ditampung oleh variabel `avg`.
- Kondisi `case max := <-ch2` akan terpenuhi ketika ada penerimaan data dari channel `ch2`, yang kemudian akan ditampung oleh variabel `max`.

Karena ada 2 buah channel, maka perlu disiapkan perulangan 2 kali sebelum penggunaan keyword `select`.

```

[novalagung:belajar-golang $ go run bab32.go
numbers : [3 4 3 5 6 3 2 2 6 3 4 6 3]
Avg      : 3.85
Max      : 6
[novalagung:belajar-golang $ go run bab32.go
numbers : [3 4 3 5 6 3 2 2 6 3 4 6 3]
Max      : 6
Avg      : 3.85
novalagung:belajar-golang $ ]

```

Cukup mudah bukan?

33. Channel - Range dan Close

Penerimaan data lewat channel yang jumlah goroutine-nya banyak bisa lebih mudah dengan memanfaatkan keyword `for - range`.

`for - range` jika diterapkan pada channel berfungsi untuk handle penerimaan data. Setiap kali ada pengiriman data ke channel, perulangan akan berjalan sekali, setelah 1 perulangan selesai akan menunggu lagi penerimaan data selanjutnya. Perulangan hanya akan berhenti jika channel di **close** atau di-non-aktifkan. Fungsi `close` digunakan untuk me-non-aktifkan channel.

Channel yang sudah di-close tidak bisa digunakan lagi untuk menerima maupun mengirim data, itulah kenapa perulangan pada `for - range` juga ikut berhenti.

33.1. Penerapan `for - range - close` Pada Channel

Berikut adalah contoh program yang menggunakan `for - range` untuk penerimaan data dari channel.

Pertama siapkan fungsi `sendMessage()` bertugas untuk mengirim data. Didalam fungsi ini dijalankan perulangan sebanyak 20 kali, di tiap perulangannya data dikirim lewat channel. Setelah semua data terkirim, channel di-close.

```
func sendMessage(ch chan<- string) {
    for i := 0; i < 20; i++ {
        ch <- fmt.Sprintf("data %d", i)
    }
    close(ch)
}
```

Siapkan juga fungsi `printMessage()` untuk handle penerimaan data. Didalamnya, channel akan di-looping menggunakan `for - range`, yang kemudian ditampilkan data-nya.

```
func printMessage(ch <-chan string) {
    for message := range ch {
        fmt.Println(message)
    }
}
```

Buat channel baru dalam `main`, jalankan `sendMessage()` sebagai goroutine. Jalankan juga `printMessage()`. Dengan ini 20 data dikirimkan lewat goroutine baru, dan nantinya diterima di goroutine utama.

```
func main() {
    runtime.GOMAXPROCS(2)

    var messages = make(chan string)
    go sendMessage(messages)
    printMessage(messages)
}
```

Setelah 20 data sukses dikirim dan diterima, channel `ch` di-non-aktifkan (`close(ch)`). Membuat perulangan data channel dalam `printMessage()` juga akan berhenti.

```
[novalagung:belajar-golang $ go run bab33.go
data 0
data 1
data 2
data 3
data 4
data 5
data 6
data 7
data 8
data 9
data 10
data 11
data 12
data 13
data 14
data 15
data 16
data 17
data 18
data 19
novalagung:belajar-golang $ ]
```

Berikut adalah penjelasan tambahan mengenai channel.

33.2. Channel Direction

Ada yang unik dengan fitur parameter channel yang disediakan Golang. Level akses channel bisa ditentukan, apakah hanya sebagai penerima, pengirim, atau penerima sekaligus pengirim. Konsep ini disebut dengan **channel direction**.

Cara pemberian level akses adalah dengan menambahkan tanda `<-` sebelum atau setelah keyword `chan`. Untuk lebih jelasnya bisa dilihat di list berikut.

Sintaks	Penjelasan
ch chan string	Parameter <code>ch</code> bisa digunakan untuk mengirim dan menerima data
ch chan<- string	Parameter <code>ch</code> hanya bisa digunakan untuk mengirim data
ch <-chan string	Parameter <code>ch</code> hanya bisa digunakan untuk menerima data

Pada kode di atas bisa dilihat bahwa secara default channel akan memiliki kemampuan untuk mengirim dan menerima data. Untuk mengubah channel tersebut agar hanya bisa mengirim atau menerima saja, dengan memanfaatkan simbol `<-`.

Sebagai contoh fungsi `sendMessage(ch chan<- string)` yang parameter `ch` dideklarasikan dengan level akses untuk pengiriman data saja. Channel tersebut hanya bisa digunakan untuk mengirim, contohnya: `ch <- fmt.Sprintf("data %d", i)`.

Dan sebaliknya pada fungsi `printMessage(ch <-chan string)`, channel `ch` hanya bisa digunakan untuk menerima data saja.

34. Channel - Timeout

Teknik timeout digunakan untuk mengontrol penerimaan data dari channel berdasarkan waktu diterimanya, dengan durasi timeout bisa kita tentukan sendiri.

Ketika tidak ada aktivitas penerimaan data dalam durasi yang sudah ditentukan, callback akan dijalankan.

34.1. Penerapan Channel Timeout

Berikut adalah program sederhana tentang pengaplikasian timeout pada channel. Sebuah goroutine baru dijalankan dengan tugas mengirimkan data setiap interval tertentu, dengan durasi interval-nya adalah acak/random.

```
package main

import "fmt"
import "math/rand"
import "runtime"
import "time"

func sendData(ch chan<- int) {
    for i := 0; true; i++ {
        ch <- i
        time.Sleep(time.Duration(rand.Int()%10+1) * time.Second)
    }
}
```

Selanjutnya, disiapkan perulangan tanpa henti, yang di tiap perulangannya ada seleksi kondisi channel menggunakan `select`.

```
func retrieveData(ch <-chan int) {
    loop:
    for {
        select {
        case data := <-ch:
            fmt.Print(`receive data `, data, ` `` , `\n`)
        case <-time.After(time.Second * 5):
            fmt.Println("timeout. no activities under 5 seconds")
            break loop
        }
    }
}
```

Ada 2 blok kondisi pada `select` tersebut.

- `case data := <-messages:`, akan terpenuhi ketika ada serah terima data pada channel `messages`
- `case <-time.After(time.Second * 5):`, akan terpenuhi ketika tidak ada aktivitas penerimaan data dari channel dalam durasi 5 detik. Blok inilah yang kita sebut sebagai callback.

Terakhir, kedua fungsi tersebut dipanggil di `main`.

```
func main() {
    rand.Seed(time.Now().Unix())
    runtime.GOMAXPROCS(2)

    var messages = make(chan int)

    go sendData(messages)
    retreiveData(messages)
}
```

Muncul output setiap kali ada penerimaan data dengan delay waktu acak. Ketika tidak ada aktifitas penerimaan dari channel dalam durasi 5 detik, perulangan pengecekan channel diberhentikan.

```
[novalagung:belajar-golang $ go run bab34.go
receive data "0"
receive data "1"
receive data "2"
receive data "3"
timeout. no activities under 5 seconds
[novalagung:belajar-golang $ go run bab34.go
receive data "0"
timeout. no activities under 5 seconds
[novalagung:belajar-golang $ go run bab34.go
receive data "0"
timeout. no activities under 5 seconds
[novalagung:belajar-golang $ go run bab34.go
receive data "0"
receive data "1"
timeout. no activities under 5 seconds
novalagung:belajar-golang $ ]]
```

35. Defer & Exit

Defer digunakan untuk mengakhirkan eksekusi sebuah statement. Sedangkan **Exit** digunakan untuk menghentikan program (ingat, menghentikan program, tidak seperti `return` yang menghentikan blok kode).

35.1. Penerapan keyword `defer`

Seperti yang sudah dijelaskan secara singkat di atas, bahwa defer digunakan untuk mengakhirkan eksekusi baris kode. Ketika eksekusi blok sudah selesai, statement yang di defer baru akan dijalankan.

Defer bisa ditempatkan di mana saja, awal maupun akhir blok.

```
package main

import "fmt"

func main() {
    defer fmt.Println("halo")
    fmt.Println("selamat datang")
}
```

Output:

```
[novalagung:belajar-golang $ go run bab35.go
selamat datang
halo
novalagung:belajar-golang $ ]
```

Keyword `defer` digunakan untuk mengakhirkan statement. Pada kode di atas, `fmt.Println("halo")` di-defer, hasilnya string `"halo"` akan muncul setelah `"selamat datang"`.

Statement yang di-defer akan tetap muncul meskipun blok kode diberhentikan ditengah jalan menggunakan `return`. Contohnya seperti pada kode berikut.

```

func main() {
    orderSomeFood("pizza")
    orderSomeFood("burger")
}

func orderSomeFood(menu string) {
    defer fmt.Println("Terimakasih, silakan tunggu")
    if menu == "pizza" {
        fmt.Print("Pilihan tepat!", " ")
        fmt.Print("Pizza ditempat kami paling enak!", "\n")
        return
    }

    fmt.Println("Pesanan anda:", menu)
}

```

Output:

```

[novalagung:chapter-35 $ go run 2-defer-return.go
Pilihan tepat! Pizza ditempat kami paling enak!
Terimakasih, silakan tunggu
Pesanan anda: burger
Terimakasih, silakan tunggu
novalagung:chapter-35 $ █

```

Info tambahan, ketika ada banyak statement yang di-defer, maka kesemuanya akan dieksekusi di akhir secara berurutan.

35.2. 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("halo")
    os.Exit(1)
    fmt.Println("selamat datang")
}

```

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

```
[novalagung:belajar-golang $ go run bab35.go ]  
exit status 1  
novalagung:belajar-golang $ ]
```

36. Error, Panic, dan Recover

Error merupakan topik yang penting dalam pemrograman go. Di bagian ini kita akan belajar mengenai pemanfaatan error dan cara membuat custom error sendiri.

Kita juga akan belajar tentang penggunaan **panic** untuk memunculkan panic error, dan **recover** untuk mengatasinya.

36.1. Pemanfaatan Error

`error` adalah sebuah tipe. Error memiliki 1 buah property berupa method `Error()`, method ini mengembalikan detail pesan error. Error termasuk tipe yang isinya bisa kosong atau `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()`.

`strconv.Atoi()` berguna untuk mengkonversi 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 nilai balik kedua.

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.Println("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())
    }
}

```

Ketika program dijalankan, muncul tulisan "Type some number: " , ketik sebuah angka lalu enter.

`fmt.Scanln(&input)` bertugas mengambil inputan yang diketik 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 pengecekan, 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` .

```

[novalagung:belajar-golang $ go run bab36.go
 Type some number: 24
 24 is number
[novalagung:belajar-golang $ go run bab36.go
 Type some number: 2a
 2a is not number
 strconv.ParseInt: parsing "2a": invalid syntax
 novalagung:belajar-golang $ ]

```

36.2. Membuat Custom Error

Selain memanfaatkan error hasil kembalian fungsi, kita juga bisa membuat error sendiri dengan menggunakan fungsi `errors.New` (untuk menggunakannya harus import package `errors` terlebih dahulu).

Pada contoh berikut ditunjukkan 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()`.

```
[novalagung:belajar-golang $ go run bab36.go
Type your name: wick
halo wick
[novalagung:belajar-golang $ go run bab36.go
Type your name:
cannot be empty
novalagung:belajar-golang $ ]
```

36.3. Penggunaan panic

Panic digunakan untuk menampilkan *trace* error sekaligus menghentikan flow goroutine (ingat, `main` juga merupakan goroutine). Setelah ada panic, proses selanjutnya tidak di-eksekusi (kecuali proses yang di-defer, akan tetap dijalankan tepat sebelum panic muncul).

Panic memnuculkan pesan di console, sama seperti `fmt.Println()` hanya saja informasi yang ditampilkan sangat mendetail dan heboh.

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 {
        fmt.Println(err.Error())
        fmt.Println("end")
    }
}
```

Coba jalankan program, lalu langsung tekan enter, error panic akan muncul, dan baris kode setelahnya tidak dijalankan.

```
[novalagung:belajar-golang $ go run bab36.go
Type your first name: wick
halo wick
[novalagung:belajar-golang $ go run bab36.go
Type your first name:
panic: cannot be empty

goroutine 1 [running]:
main.main()
    /Users/novalagung/Documents/go/src/belajar-golang/bab36.go:26 +0x37c
exit status 2
novalagung:belajar-golang $ ]
```

36.4. Penggunaan recover

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

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 error panic yang seharusnya muncul. Fungsi yang berisikan `recover()` harus di-defer, karena meskipun muncul panic, defer tetap di-eksekusi.

```
func catch() {
    if r := recover(); r != nil {
        fmt.Println("Error occurred", 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")
    }
}
```

Output:

```
[novalagung:chapter-36 $ go run 4-recover.go
Type your name: john
halo john
Application running perfectly
[novalagung:chapter-36 $
[novalagung:chapter-36 $ go run 4-recover.go
Type your name:
Error occured cannot be empty
novalagung:chapter-36 $ ]
```

37. Layout Format String

Di bab-bab sebelumnya kita telah banyak menggunakan layout format string seperti `%s` , `%d` , `%.2f` , dan lainnya; untuk keperluan menampilkan output ke layar ataupun untuk memformat string.

Layout format string digunakan dalam konversi data ke bentuk string. Contohnya seperti `%.3f` yang untuk konversi nilai `double` ke `string` dengan 3 digit desimal.

Pada bab ini kita akan mempelajari satu per satu layout format string yang tersedia di Golang. Kode berikut adalah sampel data yang akan kita gunakan sebagai contoh.

```
type student struct {
    name      string
    height    float64
    age       int32
    isGraduated bool
    hobbies   []string
}

var data = student{
    name:      "wick",
    height:    182.5,
    age:       26,
    isGraduated: false,
    hobbies:   []string{"eating", "sleeping"},
}
```

37.1. Layout Format `%b`

Digunakan untuk memformat data numerik, menjadi bentuk string numerik berbasis 2 (biner).

```
fmt.Printf("%b\n", data.age)
// 11010
```

37.2. Layout Format `%c`

Digunakan untuk memformat data numerik yang merupakan kode unicode, menjadi bentuk string karakter unicode-nya.

```
fmt.Printf("%c\n", 1400)
// n

fmt.Printf("%c\n", 1235)
// ä
```

37.3. Layout Format %d

Digunakan untuk memformat data numerik, menjadi bentuk string numerik berbasis 10 (basis bilangan yang kita gunakan).

```
fmt.Printf("%d\n", data.age)
// 26
```

37.4. Layout Format %e atau %E

Digunakan untuk memformat data numerik desimal ke dalam bentuk notasi numerik standar [Scientific notation](#).

```
fmt.Printf("%e\n", data.height)
// 1.825000e+02

fmt.Printf("%E\n", data.height)
// 1.825000E+02
```

1.825000E+02 maksudnya adalah **1.825×10^2** , dan hasil operasi tersebut adalah sesuai dengan data asli = **182.5**.

Perbedaan antara `%e` dan `%E` hanya pada bagian huruf besar kecil karakter `e` pada hasil.

37.5. Layout Format %f atau %F

`%F` adalah alias dari `%f`. Keduanya memiliki fungsi yang sama.

Berfungsi untuk memformat data numerik desimal, dengan lebar desimal bisa ditentukan. Secara default lebar digit desimal adalah 6 digit.

```

fmt.Printf("%f\n", data.height)
// 182.500000

fmt.Printf("%.9f\n", data.height)
// 182.500000000

fmt.Printf("%.2f\n", data.height)
// 182.50

fmt.Printf("%.f\n", data.height)
// 182

```

37.6. Layout Format %g atau %G

`%g` adalah alias dari `%g`. Keduanya memiliki fungsi yang sama.

Berfungsi untuk memformat data numerik desimal, dengan lebar desimal bisa ditentukan. Lebar kapasitasnya sangat besar, pas digunakan untuk data yang jumlah digit desimalnya cukup banyak.

Bisa dilihat pada kode berikut perbandingan antara `%e`, `%f`, dan `%g`.

```

fmt.Printf("%e\n", 0.123123123123)
// 1.231231e-01

fmt.Printf("%f\n", 0.123123123123)
// 0.123123

fmt.Printf("%g\n", 0.123123123123)
// 0.123123123123

```

Perbedaan lainnya adalah pada `%g`, lebar digit desimal adalah sesuai dengan datanya, tidak bisa dicustom seperti pada `%f`.

```

fmt.Printf("%g\n", 0.12)
// 0.12

fmt.Printf("%.5g\n", 0.12)
// 0.12

```

37.7. Layout Format %o

Digunakan untuk memformat data numerik, menjadi bentuk string numerik berbasis 8 (oktal).

```
fmt.Printf("%o\n", data.age)
// 32
```

37.8. Layout Format %p

Digunakan untuk memformat data pointer, mengembalikan alamat pointer referensi variabelnya.

Alamat pointer dituliskan dalam bentuk numerik berbasis 16 dengan prefix `0x`.

```
fmt.Printf("%p\n", &data.name)
// 0x2081be0c0
```

37.9. Layout Format %q

Digunakan untuk **escape** string. Meskipun string yang dipakai menggunakan literal `\` akan tetap di-escape.

```
fmt.Printf("%q\n", ` name \ height `)
// "\ name \\ height \"
```

37.10. Layout Format %s

Digunakan untuk memformat data string.

```
fmt.Printf("%s\n", data.name)
// wick
```

37.11. Layout Format %t

Digunakan untuk memformat data boolean, menampilkan nilai `bool`-nya.

```
fmt.Printf("%t\n", data.isGraduated)
// false
```

37.12. Layout Format %T

Berfungsi untuk mengambil tipe variabel yang akan diformat.

```
fmt.Printf("%T\n", data.name)
// string

fmt.Printf("%T\n", data.height)
// float64

fmt.Printf("%T\n", data.age)
// int32

fmt.Printf("%T\n", data.isGraduated)
// bool

fmt.Printf("%T\n", data.hobbies)
// []string
```

37.13. Layout Format %v

Digunakan untuk memformat data apa saja (termasuk data bertipe `interface{}`). Hasil kembalinya adalah string nilai data aslinya.

Jika data adalah objek cetakan `struct`, maka akan ditampilkan semua secara property berurutan.

```
fmt.Printf("%v\n", data)
// {wick 182.5 26 false [eating sleeping]}
```

37.14. Layout Format %+v

Digunakan untuk memformat struct, mengembalikan nama tiap property dan nilainya berurutan sesuai dengan struktur struct.

```
fmt.Printf("%+v\n", data)
// {name:wick height:182.5 age:26 isGraduated:false hobbies:[eating sleeping]}
```

37.15. Layout Format %#v

Digunakan untuk memformat struct, mengembalikan nama dan nilai tiap property sesuai dengan struktur struct dan juga bagaimana objek tersebut dideklarasikan.

```
fmt.Printf("%#v\n", data)
// main.student{name:"wick", height:182.5, age:26, isGraduated:false, hobbies:[]string
{"eating", "sleeping"})
```

Ketika menampilkan objek yang deklarasinya adalah menggunakan teknik *anonymous struct*, maka akan muncul juga struktur anonymous struct nya.

```
var data = struct {
    name    string
    height  float64
} {
    name:    "wick",
    height: 182.5,
}

fmt.Printf("%#v\n", data)
// struct { name string; height float64 }{name:"wick", height:182.5}
```

Format ini juga bisa digunakan untuk menampilkan tipe data lain, dan akan dimunculkan strukturnya juga.

37.16. Layout Format `%x` atau `%X`

Digunakan untuk memformat data numerik, menjadi bentuk string numerik berbasis 16 (heksadesimal).

```
fmt.Printf("%x\n", data.age)
// 1a
```

Jika digunakan pada tipe data string, maka akan mengembalikan kode heksadesimal tiap karakter.

```
var d = data.name

fmt.Printf("%x%x%x%x\n", d[0], d[1], d[2], d[3])
// 7769636b

fmt.Printf("%x\n", d)
// 7769636b
```

`%x` dan `%X` memiliki fungsi yang sama. Perbedaannya adalah `%x` akan mengembalikan string dalam bentuk *uppercase* atau huruf kapital.

37.17. Layout Format `%%`

Cara untuk menulis karakter `%` pada string format.

```
fmt.Printf("%%\n")  
// %
```

38. Time, Parsing Time, & Format Time

Pada bab ini kita akan belajar tentang pemanfaatan data bertipe waktu, method-method yang disediakan, dan juga **format** & **parsing** data `string` ke tipe `time.Time` dan sebaliknya.

Golang menyediakan package `time` yang berisikan banyak sekali komponen yang bisa digunakan untuk keperluan pemanfaatan waktu. Salah satunya adalah `time.Time`, yang merupakan tipe untuk data tanggal dan waktu di Golang.

Time disini maksudnya adalah gabungan `date` dan `time`, bukan hanya waktu saja.

38.1. Penggunaan `time.Time`

`time.Time` adalah tipe data untuk objek waktu. Ada 2 cara yang bisa digunakan untuk membuat data bertipe ini.

- Menjadikan informasi waktu sekarang sebagai objek `time.Time`.
- Membuat objek baru bertipe `time.Time` dengan informasi ditentukan sendiri.

Berikut merupakan contoh penggunannya.

```
package main

import "fmt"
import "time"

func main() {
    var time1 = time.Now()
    fmt.Printf("time1 %v\n", time1)
    // time1 2015-09-01 17:59:31.73600891 +0700 WIB

    var time2 = time.Date(2011, 12, 24, 10, 20, 0, 0, time.UTC)
    fmt.Printf("time2 %v\n", time2)
    // time2 2011-12-24 10:20:00 +0000 UTC
}
```

Fungsi `time.Now()` mengembalikan objek `time.Time` dengan informasi adalah waktu sekarang, waktu tepat ketika statement tersebut dijalankan. Bisa dilihat ketika di tampilkan, informasi yang muncul adalah sesuai dengan tanggal program tersebut dieksekusi.

```
[novalagung:belajar-golang $ go run bab38.go
time1 2015-10-08 10:02:43.584690264 +0700 WIB
time2 2011-12-24 10:20:00 +0000 UTC
novalagung:belajar-golang $ ]
```

Fungsi `time.Date()` digunakan untuk membuat objek `time.Time` baru yang informasi waktunya kita tentukan sendiri. Fungsi ini memiliki 8 buah parameter **mandatory** dengan skema bisa dilihat di kode berikut:

```
time.Date(tahun, bulan, tanggal, jam, menit, detik, nanodetik, timezone)
```

Objek cetakan fungsi `time.Now()`, informasi timezone-nya adalah relatif terhadap lokasi kita. Karena kebetulan penulis berlokasi di Jawa Timur, maka akan terdeteksi masuk dalam **GMT+7** atau **WIB**. Berbeda dengan variabel `time2` yang lokasinya sudah kita tentukan secara eksplisit yaitu **UTC**.

Selain menggunakan `time.UTC` untuk penentuan lokasi, tersedia juga `time.Local` yang nilainya adalah relatif terhadap waktu lokal kita.

38.2. Method Milik `time.Time`

Tipe data `time.Time` merupakan struct, memiliki beberapa method yang bisa dipakai.

```
var now = time.Now()
fmt.Println("year:", now.Year(), "month:", now.Month())
// year: 2015 month: 8
```

Kode di atas adalah contoh penggunaan beberapa method milik objek bertipe `time.Time`. Method `Year()` mengembalikan informasi tahun, dan `Month()` mengembalikan informasi angka bulan.

Selain kedua method di atas, ada banyak lagi yang bisa dimanfaatkan. Tabel berikut merupakan list method yang berhubungan dengan *date*, *time*, dan *location* yang dimiliki tipe `time.Time`.

Method	Return Type	Penjelasan
<code>now.Year()</code>	<code>int</code>	Tahun
<code>now.YearDay()</code>	<code>int</code>	Hari ke-? di mulai awal tahun
<code>now.Month()</code>	<code>int</code>	Bulan
<code>now.Weekday()</code>	<code>string</code>	Nama hari. Bisa menggunakan <code>now.Weekday().String()</code> untuk mengambil bentuk string-nya
<code>now.ISOWeek()</code>	<code>(int , int)</code>	Tahun dan minggu ke-? mulai awal tahun
<code>now.Day()</code>	<code>int</code>	Tanggal
<code>now.Hour()</code>	<code>int</code>	Jam
<code>now.Minute()</code>	<code>int</code>	Menit
<code>now.Second()</code>	<code>int</code>	Detik
<code>now.Nanosecond()</code>	<code>int</code>	Nano detik
<code>now.Local()</code>	<code>time.Time</code>	Waktu dalam timezone lokal
<code>now.Location()</code>	<code>*time.Location</code>	Mengambil informasi lokasi, apakah <i>local</i> atau <i>utc</i> . Bisa menggunakan <code>now.Location().String()</code> untuk mengambil bentuk string-nya
<code>now.Zone()</code>	<code>(string , int)</code>	Mengembalikan informasi <i>timezone offset</i> dalam string dan numerik. Sebagai contoh WIB, 25200
<code>now.IsZero()</code>	<code>bool</code>	Deteksi apakah nilai object <code>now</code> adalah 01 Januari tahun 1, 00:00:00 UTC . Jika iya maka bernilai <code>true</code>
<code>now.UTC()</code>	<code>time.Time</code>	Waktu dalam timezone UTC
<code>now.Unix()</code>	<code>int64</code>	Waktu dalam format <i>unix time</i>
<code>now.UnixNano()</code>	<code>int64</code>	Waktu dalam format <i>unix time</i> . Infomasi nano detik juga dimasukkan
<code>now.String()</code>	<code>string</code>	Waktu dalam string

38.3. Parsing `time.Time`

Data `string` bisa dikonversi menjadi `time.Time` dengan memanfaatkan `time.Parse`. Fungsi ini membutuhkan 2 parameter:

- Parameter ke-1 adalah layout format dari data waktu yang akan diparsing.
- Parameter ke-2 adalah data string yang ingin diparsing.

Contoh penerapannya bisa dilihat di kode berikut.

```
var layoutFormat, value string
var date time.Time

layoutFormat = "2006-01-02 15:04:05"
value = "2015-09-02 08:04:00"
date, _ = time.Parse(layoutFormat, value)
fmt.Println(value, "\t->", date.String())
// 2015-09-02 08:04:00 +0000 UTC

layoutFormat = "02/01/2006 MST"
value = "02/09/2015 WIB"
date, _ = time.Parse(layoutFormat, value)
fmt.Println(value, "\t\t->", date.String())
// 2015-09-02 00:00:00 +0700 WIB
```

```
[novalagung:belajar-golang $ go run bab38.go
2015-09-02 08:04:00      -> 2015-09-02 08:04:00 +0000 UTC
02/09/2015 WIB          -> 2015-09-02 00:00:00 +0700 WIB
novalagung:belajar-golang $ ]
```

Layout format time di golang berbeda dibanding bahasa lain. Umumnya layout format yang digunakan adalah seperti `"DD/MM/YYYY"`, di Golang tidak.

Golang memiliki standar layout format yang cukup unik, contohnya seperti pada kode di atas `"2006-01-02 15:04:05"`. Golang menggunakan `2006` untuk parsing tahun, bukan `YYYY`; `01` untuk parsing bulan; `02` untuk parsing hari; dan seterusnya. Detailnya bisa dilihat di tabel berikut.

Layout Format	Penjelasan	Contoh Data
2006	Tahun 4 digit	2015
006	Tahun 3 digit	015
06	Tahun 2 digit	15
01	Bulan 2 digit	05
1	Bulan 1 digit jika dibawah bulan 10, selainnya 2 digit	5 , 12
January	Nama bulan dalam bahasa inggris	September , August
Jan	Nama bulan dalam bahasa inggris, 3 huruf	Sep , Aug
02	Tanggal 2 digit	02
2	Tanggal 1 digit jika dibawah bulan 10, selainnya 2 digit	8 , 31
Monday	Nama hari dalam bahasa inggris	Saturday , Friday
Mon	Nama hari dalam bahasa inggris, 3 huruf	Sat , Fri
15	Jam dengan format 24 jam	18
03	Jam dengan format 12 jam 2 digit	05 , 11
3	Jam dengan format 12 jam 1 digit jika dibawah jam 11, selainnya 2 digit	5 , 11
PM	AM/PM, biasa digunakan dengan format jam 12 jam	PM , AM
04	Menit 2 digit	08
4	Menit 1 digit jika dibawah menit 10, selainnya 2 digit	8 , 24
05	Detik 2 digit	06
5	Detik 1 digit jika dibawah detik 10, selainnya 2 digit	6 , 36
999999	Nano detik	124006
MST	Lokasi timezone	UTC , WIB , EST
Z0700	Offset timezone	Z , +0700 , -0200

38.4. Predefined Layout Format Untuk Keperluan Parsing Time

Golang juga menyediakan beberapa predefined layout format umum yang bisa dimanfaatkan. Jadi tidak perlu menuliskan kombinasi komponen-komponen layout format.

Salah satu predefined layout yang bisa digunakan adalah `time.RFC822`. Format ini sama dengan `02 Jan 06 15:04 MST`. Berikut adalah contoh penerapannya.

```
var date, _ = time.Parse(time.RFC822, "02 Sep 15 08:00 WIB")
fmt.Println(date.String())
// 2015-09-02 08:00:00 +0700 WIB
```

Ada beberapa layout format lain yang tersedia, silakan lihat tabel berikut.

Predefined Layout Format	Layout Format
<code>time.ANSIC</code>	Mon Jan _2 15:04:05 2006
<code>time.UnixDate</code>	Mon Jan _2 15:04:05 MST 2006
<code>time.RubyDate</code>	Mon Jan 02 15:04:05 -0700 2006
<code>time.RFC822</code>	02 Jan 06 15:04 MST
<code>time.RFC822Z</code>	02 Jan 06 15:04 -0700
<code>time.RFC850</code>	Monday, 02-Jan-06 15:04:05 MST
<code>time.RFC1123</code>	Mon, 02 Jan 2006 15:04:05 MST
<code>time.RFC1123Z</code>	Mon, 02 Jan 2006 15:04:05 -0700
<code>time.RFC3339</code>	2006-01-02T15:04:05Z07:00
<code>time.RFC3339Nano</code>	2006-01-02T15:04:05.999999999Z07:00
<code>time.Kitchen</code>	3:04PM
<code>time.Stamp</code>	Jan _2 15:04:05
<code>time.StampMilli</code>	Jan _2 15:04:05.000
<code>time.StampMicro</code>	Jan _2 15:04:05.000000
<code>time.StampNano</code>	Jan _2 15:04:05.000000000

38.5. Format `time.Time`

Setelah sebelumnya kita belajar tentang cara konversi data dengan tipe `string` ke `time.Time`. Kali ini kita akan belajar kebalikannya, konversi `time.Time` ke `string`.

Method `Format()` milik tipe `time.Time` digunakan untuk membentuk output `string` sesuai dengan layout format yang diinginkan. Contoh bisa dilihat pada kode berikut.

```

var date, _ = time.Parse(time.RFC822, "02 Sep 15 08:00 WIB")

var dateS1 = date.Format("Monday 02, January 2006 15:04 MST")
fmt.Println("dateS1", dateS1)
// Wednesday 02, September 2015 08:00 WIB

var dateS2 = date.Format(time.RFC3339)
fmt.Println("dateS2", dateS2)
// 2015-09-02T08:00:00+07:00

```

Variabel `date` di atas berisikan hasil parsing data dengan format `time.RFC822`. Data tersebut kemudian diformat sebagai string 2 kali dengan layout format berbeda.

```
[novalagung:belajar-golang $ go run bab38.go
dateS1 Wednesday 02, September 2015 08:00 WIB
dateS2 2015-09-02T08:00:00+07:00
novalagung:belajar-golang $ ]
```

38.6. Handle Error Parsing `time.Time`

Ketika parsing `string` ke `time.Time`, sangat memungkinkan bisa terjadi error karena struktur data yang akan diparsing tidak sesuai layout format yang digunakan.

Error tidaknya parsing bisa diketahui lewat nilai kembalian ke-2 fungsi `time.Parse`. Berikut adalah contoh penerapannya.

```

var date, err = time.Parse("06 Jan 15", "02 Sep 15 08:00 WIB")

if err != nil {
    fmt.Println("error", err.Error())
    return
}

fmt.Println(date)

```

Kode di atas menghasilkan error karena format tidak sesuai dengan skema data yang akan diparsing. Layout format yang seharusnya digunakan adalah `06 Jan 15 03:04 MST`.

```
[novalagung:belajar-golang $ go run bab38.go
error parsing time "02 Sep 15 08:00 WIB": extra text: 08:00 WIB
novalagung:belajar-golang $ ]
```

39. Timer

Ada beberapa fungsi dalam package `time` yang bisa dimanfaatkan untuk menunda atau mengatur jadwal eksekusi sebuah proses dalam jeda waktu tertentu.

39.1. Fungsi `time.Sleep()`

Fungsi ini digunakan untuk menghentikan program sejenak. `time.Sleep()` bersifat **blocking**, statement dibawahnya tidak akan dieksekusi sampai waktunya pemberhentian usai. Contoh sederhana penerapan bisa dilihat pada kode berikut.

```
package main

import "fmt"
import "time"

func main () {
    fmt.Println("start")
    time.Sleep(time.Second * 4)
    fmt.Println("after 4 seconds")
}
```

Hasilnya, tulisan `"start"` muncul, lalu 4 detik kemudian tulisan `"after 4 seconds"` muncul.

39.2. Fungsi `time.NewTimer()`

Fungsi ini sedikit berbeda dengan `time.Sleep()`. Fungsi `time.NewTimer()` mengembalikan objek bertipe `*time.Timer` yang memiliki property `c`. Cara kerja fungsi ini, setelah jeda waktu yang ditentukan sebuah data akan dikirimkan lewat channel `c`. Penggunaan fungsi ini harus diikuti dengan statement untuk penerimaan data dari channel `c`.

Untuk lebih jelasnya silakan perhatikan kode berikut.

```
var timer = time.NewTimer(4 * time.Second)
fmt.Println("start")
<-timer.C
fmt.Println("finish")
```

Statement `var timer = time.NewTimer(4 * time.Second)` mengindikasikan bahwa nantinya akan ada data yang dikirimkan ke channel `timer.c` setelah 4 detik berlalu. Baris kode `<- timer.c` menandakan penerimaan data dari channel `timer.c`. Karena penerimaan channel sendiri sifatnya adalah blocking, maka statement `fmt.Println("finish")` baru akan dieksekusi setelah **4 detik**.

Hasil program di atas adalah tulisan `"start"` muncul, lalu setelah 4 detik tulisan `"expired"` muncul.

39.3. Fungsi `time.AfterFunc()`

Fungsi `time.AfterFunc()` memiliki 2 parameter. Parameter pertama adalah durasi timer, dan parameter kedua adalah *callback* nya. Callback tersebut akan dieksekusi jika waktu sudah memenuhi durasi timer.

```
var ch = make(chan bool)

time.AfterFunc(4*time.Second, func() {
    fmt.Println("expired")
    ch <- true
})

fmt.Println("start")
<-ch
fmt.Println("finish")
```

Hasil dari kode di atas, tulisan `"start"` muncul kemudian setelah 4 detik berlalu, tulisan `"expired"` muncul.

Dalam callback terdapat proses transfer data lewat channel, menjadikan tulisan `"finish"` akan muncul tepat setelah tulisan `"expired"` muncul.

Beberapa hal yang perlu diketahui dalam menggunakan fungsi ini:

- Jika tidak ada serah terima data lewat channel, maka eksekusi `time.AfterFunc()` adalah asynchronous dan tidak blocking.
- Jika ada serah terima data lewat channel, maka fungsi akan tetap berjalan asynchronous dan tidak blocking hingga baris kode dimana penerimaan data channel dilakukan.

39.4. Fungsi `time.After()`

Kegunaan fungsi ini mirip seperti `time.Sleep()`. Perbedaannya adalah, fungsi `timer.After()` akan mengembalikan data channel, sehingga perlu menggunakan tanda `<-` dalam penerapannya.

```
<-time.After(4 * time.Second)
fmt.Println("expired")
```

Tulisan `"expired"` akan muncul setelah 4 detik.

39.5. Kombinasi Timer & Goroutine

Berikut merupakan contoh penerapan timer dan goroutine. Program di bawah ini adalah program tanya-jawab sederhana. Sebuah pertanyaan muncul dan user harus menginputkan jawaban dalam waktu tidak lebih dari 5 detik. Jika 5 detik berlalu dan belum ada jawaban, maka akan muncul pesan **time out**.

OK langsung saja, mari kita buat programnya, pertama, import package yang diperlukan.

```
package main

import "fmt"
import "os"
import "time"
```

Buat fungsi `timer()`, nantinya fungsi ini dieksekusi sebagai goroutine. Di dalam fungsi `timer()` terdapat blok kode jika waktu sudah mencapai `timeout`, maka sebuah data dikirimkan lewat channel `ch`.

```
func timer(timeout int, ch chan<- bool) {
    time.AfterFunc(time.Duration(timeout)*time.Second, func() {
        ch <- true
    })
}
```

Siapkan juga fungsi `watcher()`. Fungsi ini juga akan dieksekusi sebagai goroutine. Tugasnya cukup sederhana, yaitu menerima data dari channel `ch` (jika ada penerimaan data, berarti sudah masuk waktu timeout), lalu menampilkan pesan bahwa waktu telah habis.

```
func watcher(timeout int, ch <-chan bool) {
    <-ch
    fmt.Println("\ntime out! no answer more than", timeout, "seconds")
    os.Exit(0)
}
```

Terakhir, buat implementasi di fungsi `main`.

```
func main() {
    var timeout = 5
    var ch = make(chan bool)

    go timer(timeout, ch)
    go watcher(timeout, ch)

    var input string
    fmt.Print("what is 725/25 ? ")
    fmt.Scan(&input)

    if input == "29" {
        fmt.Println("the answer is right!")
    } else {
        fmt.Println("the answer is wrong!")
    }
}
```

Ketika user tidak menginputkan apa-apa dalam kurun waktu 5 detik, maka akan muncul pesan `timeout`, lalu program dihentikan.

```
[novalagung:belajar-golang $ go run bab39.go
what is 725/25 ? 34
the answer is wrong!
[novalagung:belajar-golang $ go run bab39.go
what is 725/25 ?
time out! no answer more than 5 seconds
[novalagung:belajar-golang $ go run bab39.go
what is 725/25 ? 29
the answer is right!
novalagung:belajar-golang $ ]]
```

40. Konversi Antar Tipe Data

Di bab-bab sebelumnya kita sudah mengaplikasikan beberapa cara konversi data, contohnya seperti konversi `string` ↔ `int` menggunakan `strconv`, dan `time.Time` ↔ `string`. Di bab ini kita akan belajar lebih banyak.

40.1. Konversi Menggunakan `strconv`

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

40.1.a. 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
    }
}
```

40.1.b. 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"
```

40.1.c. 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
}
```

40.1.d. 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
```

40.1.e. 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](#).

40.1.f. 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`.

Ada beberapa format eksponen yang bisa digunakan. Detailnya bisa dilihat di tabel berikut.

Format Eksponen	Penjelasan
b	-ddddp±ddd, a, eksponen biner (basis 2)
e	-d.ddde±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 <code>e</code> untuk eksponen besar dan <code>f</code> untuk selainnya
G	Akan menggunakan format eksponen <code>E</code> untuk eksponen besar dan <code>f</code> untuk selainnya

40.1.g. 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
}
```

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

Digunakan untuk konversi `bool` ke `string`.

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

fmt.Println(str) // 124
```

40.2. Konversi Data Menggunakan Casting

Keyword tipe data bisa digunakan untuk casting. Cara penggunaannya adalah dengan menuliskan tipe data sebagai fungsi dan menyisipkan data yang akan dikonversi sebagai parameternya.

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

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

40.3. 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, tiap 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

```

40.4. Konversi Data `interface{}` Menggunakan Teknik Type Assertions

Type assertions merupakan teknik casting data `interface{}` ke segala jenis tipe (dengan syarat data tersebut memang bisa di-casting ke tipe tujuan).

Berikut merupakan contoh penerapannya. Variabel `data` disiapkan bertipe `map[string]interface{}`, berisikan beberapa item dengan tipe data value nya berbeda satu sama lain.

```

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"]` dicasting sebagai `string`.

Tipe asli data pada variabel `interface{}` bisa diketahui dengan cara meng-casting ke tipe `type`. Namun casting ke tipe `type` 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 asli sebuah data bertipe `interface{}`, contoh penerapannya seperti pada kode di atas.

41. Fungsi String

Golang menyediakan package `strings`, berisikan cukup banyak fungsi untuk keperluan pengolahan data string. Bab ini berisi pembahasan mengenai beberapa fungsi yang ada di dalam package tersebut.

41.1. Fungsi `strings.Contains()`

Dipakai untuk deteksi apakah string (parameter kedua) merupakan bagian dari string lain (parameter pertama). Nilai kembalinya berupa `bool`.

```
package main

import "fmt"
import "strings"

func main() {
    var isExists = strings.Contains("john wick", "wick")
    fmt.Println(isExists)
}
```

Variabel `isExists` akan bernilai `true`, karena string `"wick"` merupakan bagian dari `"john wick"`.

41.2. Fungsi `strings.HasPrefix()`

Digunakan untuk deteksi apakah sebuah string (parameter pertama) diawali string tertentu (parameter kedua).

```
var isPrefix1 = strings.HasPrefix("john wick", "jo")
fmt.Println(isPrefix1) // true

var isPrefix2 = strings.HasPrefix("john wick", "wi")
fmt.Println(isPrefix2) // false
```

41.3. Fungsi `strings.HasSuffix()`

Digunakan untuk deteksi apakah sebuah string (parameter pertama) diakhiri string tertentu (parameter kedua).

```
var isSuffix1 = strings.HasPrefix("john wick", "ic")
fmt.Println(isSuffix1) // false

var isSuffix2 = strings.HasPrefix("john wick", "ck")
fmt.Println(isSuffix2) // true
```

41.4. Fungsi strings.Count()

Memiliki kegunaan untuk menghitung jumlah karakter tertentu (parameter kedua) dari sebuah string (parameter pertama). Nilai kembalian fungsi ini adalah jumlah karakternya.

```
var howMany = strings.Count("ethan hunt", "t")
fmt.Println(howMany) // 2
```

Nilai yang dikembalikan `2`, karena pada string `"ethan hunt"` terdapat dua buah karakter `"t"`.

41.5. Fungsi strings.Index()

Digunakan untuk mencari posisi indeks sebuah string (parameter kedua) dalam string (parameter pertama).

```
var index1 = strings.Index("ethan hunt", "ha")
fmt.Println(index1) // 2
```

String `"ha"` berada pada posisi ke `2` dalam string `"ethan hunt"` (indeks dimulai dari 0). Jika diketemukan dua substring, maka yang diambil adalah yang pertama, contoh:

```
var index2 = strings.Index("ethan hunt", "n")
fmt.Println(index2) // 4
```

String `"n"` berada pada indeks `4` dan `8`. Yang dikembalikan adalah yang paling kiri (paling kecil), yaitu `4`.

41.6. Fungsi strings.Replace()

Fungsi ini digunakan untuk replace atau mengganti bagian dari string dengan string tertentu. Jumlah substring yang di-replace bisa ditentukan, apakah hanya 1 string pertama, 2 string, atau kesemuanya.

```
var text = "banana"
var find = "a"
var replaceWith = "o"

var newText1 = strings.Replace(text, find, replaceWith, 1)
fmt.Println(newText1) // "bonana"

var newText2 = strings.Replace(text, find, replaceWith, 2)
fmt.Println(newText2) // "bonona"

var newText3 = strings.Replace(text, find, replaceWith, -1)
fmt.Println(newText3) // "bonono"
```

Penjelasan:

1. Pada contoh di atas, substring "a" pada string "banana" akan di-replace dengan string "o".
2. Pada `newText1`, hanya 1 huruf `o` saja yang tereplace karena maksimal substring yang ingin di-replace ditentukan 1.
3. Angka `-1` akan menjadikan proses replace berlaku pada semua substring. Contoh bisa dilihat pada `newText3`.

41.7. Fungsi `strings.Repeat()`

Digunakan untuk mengulang string (parameter pertama) sebanyak data yang ditentukan (parameter kedua).

```
var str = strings.Repeat("na", 4)
fmt.Println(str) // "nananana"
```

Pada contoh di atas, string "na" diulang sebanyak 4 kali. Hasilnya adalah: "nananana"

41.8. Fungsi `strings.Split()`

Digunakan untuk memisah string (parameter pertama) dengan tanda pemisah bisa ditentukan sendiri (parameter kedua). Hasilnya berupa array string.

```
var string1 = strings.Split("the dark knight", " ")
fmt.Println(string1) // ["the", "dark", "knight"]

var string2 = strings.Split("batman", "")
fmt.Println(string2) // ["b", "a", "t", "m", "a", "n"]
```

String `"the dark knight"` dipisah oleh karakter spasi `" "`, hasilnya kemudian ditampung oleh `string1`.

Untuk memisah string menjadi array tiap 1 string, gunakan pemisah string kosong `""`. Bisa dilihat contohnya pada variabel `string2`.

41.9. Fungsi `strings.Join()`

Memiliki kegunaan berkebalikan dengan `strings.Split()`. Digunakan untuk menggabungkan array string (parameter pertama) menjadi sebuah string dengan pemisah tertentu (parameter kedua).

```
var data = []string{"banana", "papaya", "tomato"}
var str = strings.Join(data, "-")
fmt.Println(str) // "banana-papaya-tomato"
```

Array `data` digabungkan menjadi satu dengan pemisah tanda *dash* (`-`).

41.10. Fungsi `strings.ToLower()`

Mengubah huruf-huruf string menjadi huruf kecil.

```
var str = strings.ToLower("aLAy")
fmt.Println(str) // "alay"
```

41.11. Fungsi `strings.ToUpper()`

Mengubah huruf-huruf string menjadi huruf besar.

```
var str = strings.ToUpper("eat!")
fmt.Println(str) // "EAT!"
```


42. Regex

Regex atau **regular expression** adalah suatu teknik yang digunakan untuk pencocokan string dengan pola tertentu. Regex biasa dimanfaatkan untuk pencarian dan pengubahan data string.

Golang mengadopsi standar regex **RE2**, untuk melihat sintaks yang di-support engine ini bisa langsung merujuk ke dokumentasinya di <https://github.com/google/re2/wiki/Syntax>.

Pada bab ini kita akan belajar mengenai pengaplikasian regex dengan memanfaatkan fungsi-fungsi dalam package `regexp`.

42.1. Penerapan Regexp

Fungsi `regexp.Compile()` digunakan untuk mengkompilasi ekspresi regex. Fungsi tersebut mengembalikan objek bertipe `regexp.*Regexp`.

Berikut merupakan contoh penerapan regex untuk pencarian karakter.

```
package main

import "fmt"
import "regexp"

func main() {
    var text = "banana burger soup"
    var regex, err = regexp.Compile(`[a-z]+`)

    if err != nil {
        fmt.Println(err.Error())
    }

    var res1 = regex.FindAllString(text, 2)
    fmt.Printf("%#v \n", res1)
    // ["banana", "burger"]

    var res2 = regex.FindAllString(text, -1)
    fmt.Printf("%#v \n", res2)
    // ["banana", "burger", "soup"]
}
```

Ekspresi `[a-z]+` maknanya adalah, semua string yang merupakan alphabet yang hurufnya kecil. Ekspresi tersebut di-compile oleh `regexp.Compile()` lalu disimpan ke variabel objek `regex` bertipe `regexp.*Regexp`.

Struct `regexp.Regexp` memiliki banyak method, salah satunya adalah `FindAllString()`, berfungsi untuk mencari semua string yang sesuai dengan ekspresi regex, dengan kembalian berupa array string.

Jumlah hasil pencarian dari `regex.FindAllString()` bisa ditentukan. Contohnya pada `res1`, ditentukan maksimal `2` data saja pada nilai kembalian. Jika batas di set `-1`, maka akan mengembalikan semua data.

Ada cukup banyak method struct `regexp.*Regexp` yang bisa kita manfaatkan untuk keperluan pengelolaan string. Berikut merupakan pembahasan tiap method-nya.

42.2. Method `MatchString()`

Method ini digunakan untuk mendeteksi apakah string memenuhi sebuah pola regexp.

```
var text = "banana burger soup"
var regex, _ = regexp.Compile(`[a-z]+`)

var isMatch = regex.MatchString(text)
fmt.Println(isMatch)
// true
```

Pada contoh di atas `isMatch` bernilai `true` karena string `"banana burger soup"` memenuhi pola regex `[a-z]+`.

42.3. Method `FindString()`

Digunakan untuk mencari string yang memenuhi kriteria regexp yang telah ditentukan.

```
var text = "banana burger soup"
var regex, _ = regexp.Compile(`[a-z]+`)

var str = regex.FindString(text)
fmt.Println(str)
// "banana"
```

Fungsi ini hanya mengembalikan 1 buah hasil saja. Jika ada banyak substring yang sesuai dengan ekspresi regexp, akan dikembalikan yang pertama saja.

42.4. Method `FindStringIndex()`

Digunakan untuk mencari index string kembalian hasil dari operasi regexp.

```
var text = "banana burger soup"
var regex, _ = regexp.MustCompile(`[a-z]+`)

var idx = regex.FindStringIndex(text)
fmt.Println(idx)
// [0, 6]

var str = text[0:6]
fmt.Println(str)
// "banana"
```

Method ini sama dengan `FindString()` hanya saja yang dikembalikan indeks-nya.

42.5. Method `FindAllString()`

Digunakan untuk mencari banyak string yang memenuhi kriteria regexp yang telah ditentukan.

```
var text = "banana burger soup"
var regex, _ = regexp.MustCompile(`[a-z]+`)

var str1 = regex.FindAllString(text, -1)
fmt.Println(str1)
// ["banana", "burger", "soup"]

var str2 = regex.FindAllString(text, 1)
fmt.Println(str2)
// ["banana"]
```

Jumlah data yang dikembalikan bisa ditentukan. Jika diisi dengan `-1`, maka akan mengembalikan semua data.

42.6. Method `ReplaceAllString()`

Berguna untuk me-replace semua string yang memenuhi kriteria regexp, dengan string lain.

```

var text = "banana burger soup"
var regex, _ = regexp.MustCompile(`[a-z]+`)

var str = regex.ReplaceAllString(text, "potato")
fmt.Println(str)
// "potato potato potato"

```

42.7. Method ReplaceAllStringFunc()

Digunakan untuk me-replace semua string yang memenuhi kriteria regexp, dengan kondisi yang bisa ditentukan untuk setiap substring yang akan di replace.

```

var text = "banana burger soup"
var regex, _ = regexp.MustCompile(`[a-z]+`)

var str = regex.ReplaceAllStringFunc(text, func(each string) string {
    if each == "burger" {
        return "potato"
    }
    return each
})
fmt.Println(str)
// "banana potato soup"

```

Pada contoh di atas, jika salah satu substring yang *match* adalah `"burger"` maka akan diganti dengan `"potato"`, string selainnya tidak di replace.

42.8. Method Split()

Digunakan untuk memisah string dengan pemisah adalah substring yang memenuhi kriteria regexp yang telah ditentukan.

Jumlah karakter yang akan di split bisa ditentukan dengan mengisi parameter kedua fungsi `regex.Split()`. Jika di-isi `-1` maka semua karakter yang memenuhi regex akan di-replace. Contoh lain, jika di-isi `2`, maka hanya 2 karakter pertama yang memenuhi regex akan di-replace.

```

var text = "banana,burger,soup"
var regex, _ = regexp.MustCompile(`[a-z]+`)

var str = regex.Split(text, -1)
fmt.Printf("%#v \n", str)
// [ "", "", "", "", "" ]

```


43. Encode - Decode Base64

Golang memiliki package `encoding/base64`, berisikan fungsi-fungsi untuk kebutuhan **encode** dan **decode** data ke base64 dan sebaliknya. Data yang akan di-encode harus bertipe `[]byte`, perlu dilakukan casting untuk data-data yang belum sesuai tipenya.

Ada beberapa cara yang bisa digunakan untuk encode dan decode data, dan di bab ini kita akan mempelajarinya.

43.1. Penerapan Fungsi `EncodeToString()` & `DecodeString()`

Fungsi `EncodeToString()` digunakan untuk encode data dari bentuk string ke base46.

Fungsi `DecodeString()` melakukan kebalikan dari `EncodeToString()`. Berikut adalah contoh penerapannya.

```
package main

import "encoding/base64"
import "fmt"

func main() {
    var data = "john wick"

    var encodedString = base64.StdEncoding.EncodeToString([]byte(data))
    fmt.Println("encoded:", encodedString)

    var decodedByte, _ = base64.StdEncoding.DecodeString(encodedString)
    var decodedString = string(decodedByte)
    fmt.Println("decoded:", decodedString)
}
```

Variabel `data` yang bertipe `string`, harus di-casting terlebih dahulu kedalam bentuk `[]byte` sebelum di-encode menggunakan fungsi `base64.StdEncoding.EncodeToString()`. Hasil encode adalah data base64 bertipe `string`.

Sedangkan pada fungsi decode `base64.StdEncoding.DecodeString()`, data base64 bertipe `string` di-decode kembali ke string aslinya, tapi bertipe `[]byte`. Ekspresi `string(decodedByte)` menjadikan data `[]byte` tersebut berubah menjadi `string`.

```
[novalagung:belajar-golang $ go run bab43.go
encoded: am9obiB3awNr
decoded: john wick
novalagung:belajar-golang $ ]
```

43.2. Penerapan Fungsi `Encode()` & `Decode()`

Kedua fungsi ini kegunaannya sama dengan fungsi yang sebelumnya kita bahas, salah satu pembedanya adalah data yang akan dikonversi dan hasilnya bertipe `[]byte`. Penggunaan cara ini cukup panjang karena variabel penyimpan hasil encode maupun decode harus disiapkan terlebih dahulu, dan harus memiliki lebar data sesuai dengan hasil yang akan ditampung (yang nilainya bisa dicari menggunakan fungsi `EncodedLen()` dan `DecodedLen()`).

Lebih jelasnya silakan perhatikan contoh berikut.

```
var data = "john wick"

var encoded = make([]byte, base64.StdEncoding.EncodedLen(len(data)))
base64.StdEncoding.Encode(encoded, []byte(data))
var encodedString = string(encoded)
fmt.Println(encodedString)

var decoded = make([]byte, base64.StdEncoding.DecodedLen(len(encoded)))
var _, err = base64.StdEncoding.Decode(decoded, encoded)
if err != nil {
    fmt.Println(err.Error())
}
var decodedString = string(decoded)
fmt.Println(decodedString)
```

Fungsi `base64.StdEncoding.EncodedLen(len(data))` menghasilkan informasi lebar data-ketika-sudah-di-encode. Nilai tersebut kemudian ditentukan sebagai lebar alokasi tipe `[]byte` pada variabel `encoded` yang nantinya digunakan untuk menampung hasil encoding.

Fungsi `base64.StdEncoding.DecodedLen()` memiliki kegunaan sama dengan `EncodedLen()`, hanya saja digunakan untuk keperluan decoding.

Dibanding 2 fungsi sebelumnya, fungsi `Encode()` dan `Decode()` memiliki beberapa perbedaan. Selain lebar data penampung encode/decode harus dicari terlebih dahulu, terdapat perbedaan lainnya, yaitu pada fungsi ini hasil encode/decode tidak didapat dari nilai kembalian, melainkan dari parameter. Variabel yang digunakan untuk menampung hasil, disisipkan pada parameter fungsi tersebut.

Pada pemanggilan fungsi encode/decode, variabel `encoded` dan `decoded` tidak disisipkan nilai pointer-nya, cukup di-pass dengan cara biasa, tipe datanya sudah dalam bentuk `[]byte`.

43.3. Encode & Decode Data URL

Khusus encode data string yang isinya merupakan URL, lebih efektif menggunakan `URLEncoding` dibandingkan `StdEncoding`.

Cara penerapannya kurang lebih sama, bisa menggunakan metode pertama maupun metode kedua yang sudah dibahas di atas. Cukup ganti `StdEncoding` menjadi `URLEncoding`.

```
var data = "https://kalipare.com/"

var encodedString = base64.URLEncoding.EncodeToString([]byte(data))
fmt.Println(encodedString)

var decodedByte, _ = base64.URLEncoding.DecodeString(encodedString)
var decodedString = string(decodedByte)
fmt.Println(decodedString)
```

44. Hash SHA1

Hash adalah algoritma enkripsi untuk mengubah text menjadi deretan karakter acak. Jumlah karakter hasil hash selalu sama. Hash termasuk *one-way encryption*, membuat hasil dari hash tidak bisa dikembalikan ke text asli.

SHA1 atau **Secure Hash Algorithm 1** merupakan salah satu algoritma hashing yang sering digunakan untuk enkripsi data. Hasil dari sha1 adalah data dengan lebar **20 byte** atau **160 bit**, biasa ditampilkan dalam bentuk bilangan heksadesimal 40 digit.

Di bab ini kita akan belajar tentang pemanfaatan sha1 dan teknik salting dalam hash.

44.1. Penerapan Hash SHA1

Golang menyediakan package `crypto/sha1`, berisikan library untuk keperluan *hashing*. Cara penerapannya cukup mudah, contohnya bisa dilihat pada kode berikut.

```
package main

import "crypto/sha1"
import "fmt"

func main() {
    var text = "this is secret"
    var sha = sha1.New()
    sha.Write([]byte(text))
    var encrypted = sha.Sum(nil)
    var encryptedString = fmt.Sprintf("%x", encrypted)

    fmt.Println(encryptedString)
    // f4ebfd7a42d9a43a536e2bed9ee4974abf8f8dc8
}
```

Variabel hasil dari `sha1.New()` adalah objek bertipe `hash.Hash`, memiliki dua buah method `Write()` dan `Sum()`.

- Method `Write()` digunakan untuk menge-set data yang akan di-hash. Data harus dalam bentuk `[]byte`.
- Method `Sum()` digunakan untuk eksekusi proses hash, menghasilkan data yang sudah di-hash dalam bentuk `[]byte`. Method ini membutuhkan sebuah parameter, isi dengan nil.

Untuk mengambil bentuk heksadesimal string dari data yang sudah di-hash, bisa memanfaatkan fungsi `fmt.Sprintf` dengan layout format `%x`.

```
[novalagung:belajar-golang $ go run bab44.go
original : this is secret
hashed    : f4ebfd7a42d9a43a536e2bed9ee4974abf8f8dc8
novalagung:belajar-golang $ ]
```

44.2. Metode Salting Pada Hash SHA1

Salt dalam konteks kriptografi adalah data acak yang digabungkan pada data asli sebelum proses hash dilakukan.

Hash merupakan enkripsi satu arah dengan lebar data yang sudah pasti, sangat mungkin sekali kalau hasil hash untuk beberapa data adalah sama. Disinilah kegunaan **salt**, teknik ini berguna untuk mencegah serangan menggunakan metode pencocokan data-data yang hasil hash-nya adalah sama (*dictionary attack*).

Langsung saja kita praktikan. Pertama import package yang dibutuhkan. Lalu buat fungsi untuk hash menggunakan salt dari waktu sekarang.

```
package main

import "crypto/sha1"
import "fmt"
import "time"

func doHashUsingSalt(text string) (string, string) {
    var salt = fmt.Sprintf("%d", time.Now().UnixNano())
    var saltedText = fmt.Sprintf("text: '%s', salt: %s", text, salt)
    fmt.Println(saltedText)
    var sha = sha1.New()
    sha.Write([]byte(saltedText))
    var encrypted = sha.Sum(nil)

    return fmt.Sprintf("%x", encrypted), salt
}
```

Salt yang digunakan adalah hasil dari ekspresi `time.Now().UnixNano()`. Hasilnya akan selalu unik setiap detiknya, karena scope terendah waktu pada fungsi tersebut adalah *nano second* atau nano detik.

Selanjutnya test fungsi yang telah dibuat beberapa kali.

```

func main() {
    var text = "this is secret"
    fmt.Printf("original : %s\n\n", text)

    var hashed1, salt1 = doHashUsingSalt(text)
    fmt.Printf("hashed 1 : %s\n\n", hashed1)
    // 929fd8b1e58afca1ebbe30beac3b84e63882ee1a

    var hashed2, salt2 = doHashUsingSalt(text)
    fmt.Printf("hashed 2 : %s\n\n", hashed2)
    // cda603d95286f0aece4b3e1749abe7128a4eed78

    var hashed3, salt3 = doHashUsingSalt(text)
    fmt.Printf("hashed 3 : %s\n\n", hashed3)
    // 9e2b514bca911cb76f7630da50a99d4f4bb200b4

    _, _, _ = salt1, salt2, salt3
}

```

Hasil ekripsi fungsi `doHashUsingSalt` akan selalu beda, karena salt yang digunakan adalah waktu.

```

[novalagung:belajar-golang $ go run bab44.go
original : this is secret

text: 'this is secret', salt: 1444813038167909774
hashed 1 : 3b3b3dc90547617236aeeb8d349eeb79d8b658cb

text: 'this is secret', salt: 1444813038167928750
hashed 2 : f7de5b412793a7f8b11f05849fab18eb137c20ca

text: 'this is secret', salt: 1444813038167934830
hashed 3 : 9ee315e39c142648888054ad1e821e7a21f3a583
]

```

Metode ini sering dipakai untuk enkripsi password user. Salt dan data hasil hash harus disimpan pada database, karena digunakan dalam pencocokan password setiap user melakukan login.

45. Arguments & Flag

Arguments adalah data opsional yang disisipkan ketika eksekusi program. Sedangkan **flag** merupakan ekstensi dari argument. Dengan flag, penulisan argument menjadi lebih rapi dan terstruktur.

Di bab ini kita akan belajar tentang penggunaan arguments dan flag.

45.1. Penggunaan Arguments

Data arguments bisa didapat lewat variabel `os.Args` (package `os` perlu di-import terlebih dahulu). Data tersebut tersimpan dalam bentuk array dengan pemisah adalah tanda spasi.

Berikut merupakan contoh penggunaannya.

```
package main

import "fmt"
import "os"

func main() {
    var argsRaw = os.Args
    fmt.Printf("-> %#v\n", argsRaw)
    // []string{.../bab45, "banana", "potato", "ice cream"}

    var args = argsRaw[1:]
    fmt.Printf("-> %#v\n", args)
    // []string{"banana", "potato"}
}
```

Pada saat eksekusi program disisipkan juga argument-nya. Sebagai contoh disisipkan 3 buah data sebagai argumen, yaitu: `banana` , `potato` , dan `ice cream` .

Untuk eksekusinya sendiri bisa menggunakan `go run` ataupun dengan cara build-execute.

- Menggunakan `go run`

```
$ go run bab45.go banana potato "ice cream"
```

- Menggunakan `go build`

```
$ go build bab45.go
$ ./bab45 banana potato "ice cream"
```

Variabel `os.Args` mengembalikan tak hanya arguments saja, tapi juga path file executable (jika eksekusi-nya menggunakan `go run` maka path akan merujuk ke folder temporary).

Gunakan `os.Args[1:]` untuk mengambil slice arguments-nya saja.

```
[novalagung:belajar-golang $ go run bab45.go banana potato "ice cream"
-> []string{"/var/folders/_2/sdbvcqxd0pq3jz14pwvf_rz0000gn/T/go-build918678092
/command-line-arguments/_obj/exe/bab45", "banana", "potato", "ice cream"}
-> []string{"banana", "potato", "ice cream"}]
[novalagung:belajar-golang $ ]
[novalagung:belajar-golang $ go build bab45.go
[novalagung:belajar-golang $ ./bab45 banana potato "ice cream"
-> []string{".bab45", "banana", "potato", "ice cream"}
-> []string{"banana", "potato", "ice cream"}]
novalagung:belajar-golang $ ]
```

Bisa dilihat pada kode di atas, bahwa untuk data argumen yang ada karakter spasi (), maka harus diapit tanda petik ("), agar tidak dideteksi sebagai 2 argumen.

45.2. Penggunaan Flag

Flag memiliki kegunaan yang sama seperti arguments, yaitu untuk *parameterize* eksekusi program, dengan penulisan dalam bentuk key-value. Berikut merupakan contoh penerapannya.

```
package main

import "flag"
import "fmt"

func main() {
    var name = flag.String("name", "anonymous", "type your name")
    var age = flag.Int64("age", 25, "type your age")

    flag.Parse()
    fmt.Printf("name\t: %s\n", *name)
    fmt.Printf("age\t: %d\n", *age)
}
```

Cara penulisan arguments menggunakan flag:

```
$ go run bab45.go -name="john wick" -age=28
```

Tiap argument harus ditentukan key, tipe data, dan nilai default-nya. Contohnya seperti pada `flag.String()` di atas. Agar lebih mudah dipahami, mari kita bahas kode berikut.

```
var dataName = flag.String("name", "anonymous", "type your name")
fmt.Println(*dataName)
```

Kode tersebut maksudnya adalah, disiapkan flag bertipe `string`, dengan key adalah `name`, dengan nilai default `"anonymous"`, dan keterangan `"type your name"`. Nilai flag nya sendiri akan disimpan kedalam variabel `dataName`.

Nilai balik fungsi `flag.String()` adalah string pointer, jadi perlu di-*dereference* terlebih dahulu agar bisa mendapatkan nilai aslinya (`*dataName`).

```
[novalagung:belajar-golang $ go run bab45.go -name="john wick" -age=28
name      : john wick
age       : 28
[novalagung:belajar-golang $ go run bab45.go -age=27
name      : anonymous
age       : 27
novalagung:belajar-golang $ ]
```

Flag yang nilainya tidak di set, secara otomatis akan mengembalikan nilai default.

Tabel berikut merupakan macam-macam fungsi flag yang tersedia untuk tiap jenis tipe data.

Nama Fungsi	Return Value
<code>flag.Bool(name, defaultValue, usage)</code>	<code>*bool</code>
<code>flag.Duration(name, defaultValue, usage)</code>	<code>*time.Duration</code>
<code>flag.Float64(name, defaultValue, usage)</code>	<code>*float64</code>
<code>flag.Int(name, defaultValue, usage)</code>	<code>*int</code>
<code>flag.Int64(name, defaultValue, usage)</code>	<code>*int64</code>
<code>flag.String(name, defaultValue, usage)</code>	<code>*string</code>
<code>flag.Uint(name, defaultValue, usage)</code>	<code>*uint</code>
<code>flag.Uint64(name, defaultValue, usage)</code>	<code>*uint64</code>

45.3. Deklarasi Flag Dengan Cara Passing Reference Variabel Penampung Data

Sebenarnya ada 2 cara deklarasi flag yang bisa digunakan, dan cara di atas merupakan cara pertama.

Cara kedua mirip dengan cara pertama, perbedannya adalah kalau di cara pertama nilai pointer flag dikembalikan lalu ditampung variabel; sedangkan pada cara kedua, nilainya diambil lewat parameter pointer.

Agar lebih jelas perhatikan contoh berikut.

```
// cara ke-1
var data1 = flag.String("name", "anonymous", "type your name")
fmt.Println(*data1)

// cara ke-2
var data2 string
flag.StringVar(&data2, "gender", "male", "type your gender")
fmt.Println(data2)
```

Tinggal tambahkan suffix `var` pada pemanggilan nama fungsi `flag` yang digunakan (contoh `flag.IntVar()`, `flag.BoolVar()`, dll), lalu disisipkan referensi variabel penampung `flag` sebagai parameter pertama.

Kegunaan dari parameter terakhir method-method `flag` adalah untuk memunculkan hints atau petunjuk arguments apa saja yang bisa dipakai, ketika argument `--help` ditambahkan saat eksekusi program.

```
[novalagung:chapter-45 $ go build 2-flag.go
[novalagung:chapter-45 $ ./2-flag --help
Usage of ./2-flag:
-age int
    type your age (default 25)
-name string
    type your name (default "anonymous")
```

46. Exec

Exec digunakan untuk eksekusi perintah command line lewat kode program. Command yang bisa dieksekusi adalah semua command yang bisa dieksekusi di terminal (CMD untuk pengguna Windows).

46.1. Penggunaan Exec

Golang menyediakan package `exec` berisikan banyak fungsi untuk keperluan eksekusi perintah CLI.

Cara untuk eksekusi command cukup mudah, yaitu dengan menuliskan command dalam bentuk string, diikuti arguments-nya (jika ada) sebagai parameter variadic pada fungsi `exec.Command()`.

```
package main

import "fmt"
import "os/exec"

func main() {
    var output1, _ = exec.Command("ls").Output()
    fmt.Printf(" -> ls\n%s\n", string(output1))

    var output2, _ = exec.Command("pwd").Output()
    fmt.Printf(" -> pwd\n%s\n", string(output2))

    var output3, _ = exec.Command("git", "config", "user.name").Output()
    fmt.Printf(" -> git config user.name\n%s\n", string(output3))
}
```

Fungsi `exec.Command()` digunakan untuk menjalankan command. Fungsi tersebut bisa langsung di-chain dengan method `Output()`, jika ingin mendapatkan outputnya. Output yang dihasilkan berbentuk `[]byte`, gunakan cast ke string untuk mengambil bentuk string-nya.

```
[novalagung:belajar-golang $ go run bab46.go ]  
-> ls  
bab43.go  
bab44.go  
bab45.go  
bab46.go  
  
-> pwd  
/Users/novalagung/Documents/go/src/belajar-golang  
  
-> git config user.name  
novalagung
```

47. File

Ada beberapa cara yang bisa digunakan untuk operasi file di Golang. Pada bab ini kita akan mempelajari teknik yang paling dasar, yaitu dengan memanfaatkan `os.File`.

47.1. Membuat File Baru

Pembuatan file di Golang sangatlah mudah, cukup dengan memanggil fungsi `os.Create()` lalu memasukkan path file ingin dibuat sebagai parameter fungsi tersebut.

Jika file yang akan dibuat sudah ada, maka akan ditimpas. Bisa memanfaatkan `os.IsNotExist()` untuk mendeteksi apakah file sudah dibuat atau belum.

Berikut merupakan contoh pembuatan file.

```

package main

import "fmt"
import "os"

var path = "/Users/novalagung/Documents/temp/test.txt"

func isError(err error) bool {
    if err != nil {
        fmt.Println(err.Error())
    }

    return (err != nil)
}

func createFile() {
    // deteksi apakah file sudah ada
    var _, err = os.Stat(path)

    // buat file baru jika belum ada
    if os.IsNotExist(err) {
        var file, err = os.Create(path)
        if isError(err) { return }
        defer file.Close()
    }

    fmt.Println("==> file berhasil dibuat", path)
}

func main() {
    createFile()
}

```

Fungsi `os.Stat()` mengembalikan 2 data, yaitu informasi tentang path yang dicari, dan error (jika ada). Masukkan error kembalian fungsi tersebut sebagai parameter fungsi `os.IsNotExist()`, untuk mendeteksi apakah file yang akan dibuat sudah ada. Jika belum ada, maka fungsi tersebut akan mengembalikan nilai `true`.

Fungsi `os.Create()` digunakan untuk membuat file pada path tertentu. Fungsi ini mengembalikan objek `*os.File` dari file yang bersangkutan. File yang baru terbuat statusnya adalah otomatis **open**, maka dari itu perlu untuk di-**close** menggunakan method `file.Close()` setelah file tidak digunakan lagi.

Membiarakan file terbuka ketika sudah tak lagi digunakan bukan hal yang baik, karena efeknya ke memory dan akses ke file itu sendiri, file akan di-lock sehingga tidak bisa digunakan oleh proses lain selama status file masih open atau belum di-close.

```
[novalagung:chapter-47 $ ls /Users/novalagung/Documents/temp/
[novalagung:chapter-47 $ go run 1-membuat-file.go
==> file berhasil dibuat /Users/novalagung/Documents/temp/test.txt
[novalagung:chapter-47 $ ls /Users/novalagung/Documents/temp/
test.txt
```

47.2. Mengedit Isi File

Untuk mengedit file, yang perlu dilakukan pertama adalah membuka file dengan level akses **write**. Setelah mendapatkan objek file-nya, gunakan method `WriteString()` untuk pengisian data. Terakhir panggil method `Sync()` untuk menyimpan perubahan.

```
func writeFile() {
    // buka file dengan level akses READ & WRITE
    var file, err = os.OpenFile(path, os.O_RDWR, 0644)
    if isError(err) { return }
    defer file.Close()

    // tulis data ke file
    _, err = file.WriteString("halo\n")
    if isError(err) { return }
    _, err = file.WriteString("mari belajar golang\n")
    if isError(err) { return }

    // simpan perubahan
    err = file.Sync()
    if isError(err) { return }

    fmt.Println("==> file berhasil di isi")
}

func main() {
    writeFile()
}
```

Pada program di atas, file dibuka dengan level akses **read** dan **write** dengan kode permission **0664**. Setelah itu, beberapa string diisikan kedalam file tersebut menggunakan `WriteString()`. Di akhir, semua perubahan terhadap file akan disimpan dengan dipanggilnya `Sync()`.

```
[novalagung:chapter-47 $ cat /Users/novalagung/Documents/temp/test.txt
[novalagung:chapter-47 $ go run 2-mengedit-file.go
==> file berhasil di isi
[novalagung:chapter-47 $ cat /Users/novalagung/Documents/temp/test.txt
halo
mari belajar golang
novalagung:chapter-47 $
```

47.3. Membaca Isi File

File yang ingin dibaca harus dibuka terlebih dahulu menggunakan fungsi `os.OpenFile()` dengan level akses minimal adalah **read**. Setelah itu, gunakan method `Read()` dengan parameter adalah variabel, yang dimana hasil proses baca akan disimpan ke variabel tersebut.

```
// tambahkan di bagian import package io
import "io"

func readFile() {
    // buka file
    var file, err = os.OpenFile(path, os.O_RDONLY, 0644)
    if isError(err) { return }
    defer file.Close()

    // baca file
    var text = make([]byte, 1024)
    for {
        n, err := file.Read(text)
        if err != io.EOF {
            if isError(err) { break }
        }
        if n == 0 {
            break
        }
    }
    if isError(err) { return }

    fmt.Println("==> file berhasil dibaca")
    fmt.Println(string(text))
}

func main() {
    readFile()
}
```

Pada kode di atas `os.OpenFile()` digunakan untuk membuka file. Fungsi tersebut memiliki beberapa parameter.

1. Parameter pertama adalah path file yang akan dibuka.
2. Parameter kedua adalah level akses. `os.O_RDONLY` maksudnya adalah **read only**.
3. Parameter ketiga adalah permission file-nya.

Variabel `text` disiapkan bertipe slice `[]byte` dengan alokasi elemen 1024. Variabel tersebut bertugas menampung data hasil statement `file.Read()`. Proses pembacaan file akan dilakukan terus menerus, berurutan dari baris pertama hingga akhir.

Error yang muncul ketika eksekusi `file.Read()` akan di-filter, ketika error tersebut adalah selain `io.EOF` maka proses baca file akan berlanjut. Error `io.EOF` sendiri menandakan bahwa file yang sedang dibaca adalah baris terakhir isi atau **end of file**.

```
[novalagung:chapter-47 $ go run 3-membaca-file.go
==> file berhasil dibaca
halo
mari belajar golang
```

47.4. Menghapus File

Cara menghapus file sangatlah mudah, cukup panggil fungsi `os.Remove()`, masukan path file yang ingin dihapus sebagai parameter.

```
func deleteFile() {
    var err = os.Remove(path)
    if isError(err) { return }

    fmt.Println("==> file berhasil di delete")
}

func main() {
    deleteFile()
}
```

```
[novalagung:chapter-47 $ ls /Users/novalagung/Documents/temp/
test.txt
[novalagung:chapter-47 $ go run 4-menghapus-file.go
==> file berhasil di delete
[novalagung:chapter-47 $ ls /Users/novalagung/Documents/temp/
novalagung:chapter-47 $ ]
```

48. Web

Golang menyediakan package `net/http`, berisi berbagai macam fitur untuk keperluan pembuatan aplikasi berbasis web. Termasuk didalamnya routing, server, templating, dan lainnya, semua tinggal pakai.

Golang memiliki web server sendiri, dan web server tersebut berada di dalam golang, tidak seperti bahasa lain yang server nya terpisah dan perlu di-instal sendiri (seperti PHP yang memerlukan Apache, .NET yang memerlukan IIS).

Di bab ini kita akan belajar cara pembuatan aplikasi web sederhana dan pemanfaatan template untuk mendesain view.

48.1. Membuat Aplikasi Web Sederhana

Package `net/http` memiliki banyak sekali fungsi yang bisa dimanfaatkan. Di bagian ini kita akan mempelajari beberapa fungsi penting seperti *routing* dan *start server*.

Program dibawah ini merupakan contoh sederhana untuk memunculkan text di web ketika url tertentu diakses.

```
package main

import "fmt"
import "net/http"

func index(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "apa kabar!")
}

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintln(w, "halo!")
    })

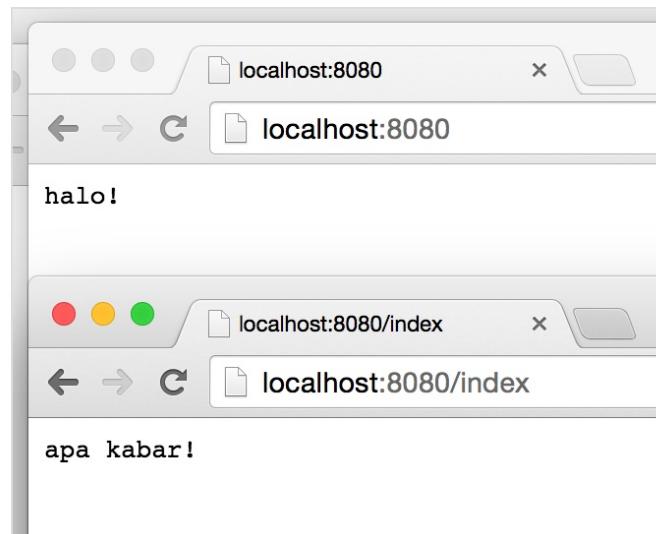
    http.HandleFunc("/index", index)

    fmt.Println("starting web server at http://localhost:8080/")
    http.ListenAndServe(":8080", nil)
}
```

Jalankan program tersebut.

```
[novalagung:belajar-golang $ go run bab48.go
starting web server at http://localhost:8080/]
```

Jika muncul dialog **Do you want the application “bab48” to accept incoming network connections?** atau sejenis, pilih allow. Setelah itu, buka url <http://localhost/> dan <http://localhost/index> lewat browser.



Fungsi `http.HandleFunc()` digunakan untuk routing aplikasi web. Maksud dari routing adalah penentuan aksi ketika url tertentu diakses oleh user.

Pada kode di atas 2 rute didaftarkan, yaitu `/` dan `/index`. Aksi dari rute `/` adalah menampilkan text `"halo"` di halaman website. Sedangkan `/index` menampilkan text `"apa kabar!"`.

Fungsi `http.HandleFunc()` memiliki 2 buah parameter yang harus diisi. Parameter pertama adalah rute yang diinginkan. Parameter kedua adalah *callback* atau aksi ketika rute tersebut diakses. Callback tersebut bertipe fungsi `func(w http.ResponseWriter, r *http.Request)`.

Pada pendaftaran rute `/index`, callback-nya adalah fungsi `index()`, hal seperti ini diperbolehkan asalkan tipe dari fungsi tersebut sesuai.

Fungsi `http.ListenAndServe()` digunakan untuk menghidupkan server sekaligus menjalankan aplikasi menggunakan server tersebut. Di golang, 1 web aplikasi adalah 1 buah server berbeda.

Pada contoh di atas, server dijalankan pada port `8080`.

Perlu diingat, setiap ada perubahan pada file `.go`, `go run` harus dipanggil lagi.

Untuk menghentikan web server, tekan **CTRL+C** pada terminal atau CMD, dimana pengeksekusian aplikasi berlangsung.

48.2. Penggunaan Template Web

Template engine memberikan kemudahan dalam mendesain tampilan view aplikasi website. Dan kabar baiknya golang menyediakan engine template sendiri, dengan banyak fitur yang tersedia didalamnya.

Di sini kita akan belajar contoh sederhana penggunaan template untuk menampilkan data. Pertama siapkan dahulu template nya. Buat file `template.html` lalu isi dengan kode berikut.

```
<html>
  <head>
    <title>Golang learn net/http</title>
  </head>
  <body>
    <p>Hello {{.Name}} !</p>
    <p>{{.Message}}</p>
  </body>
</html>
```

Kode `{{.Name}}` artinya memunculkan isi data property `Name` yang dikirim dari router. Kode tersebut nantinya di-replace dengan isi variabel `Name`.

Selanjutnya ubah isi file `.go` dengan kode berikut.

```

package main

import "fmt"
import "html/template"
import "net/http"

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        var data = map[string]string{
            "Name":      "john wick",
            "Message":   "have a nice day",
        }

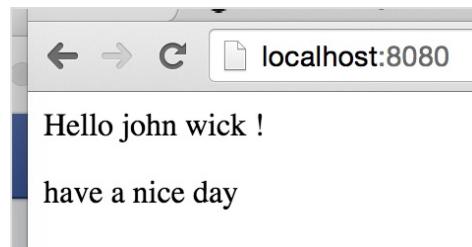
        var t, err = template.ParseFiles("template.html")
        if err != nil {
            fmt.Println(err.Error())
            return
        }

        t.Execute(w, data)
    })

    fmt.Println("starting web server at http://localhost:8080/")
    http.ListenAndServe(":8080", nil)
}

```

Jalankan, lalu buka <http://localhost:8080/>, maka data `Nama` dan `Message` akan muncul di view.



Fungsi `template.ParseFiles()` digunakan untuk parsing template, mengembalikan 2 data yaitu instance template-nya dan error (jika ada). Pemanggilan method `Execute()` akan membuat hasil parsing template ditampilkan ke layar web browser.

Pada kode di atas, variabel `data` disisipkan sebagai parameter ke-2 method `Execute()`. Isi dari variabel tersebut bisa diakses di-view dengan menggunakan notasi `{{.NAMA_PROPERTY}}` (nama variabel sendiri tidak perlu dituliskan, langsung nama property didalamnya).

Pada contoh di atas, statement di view `{{.Name}}` akan menampilkan isi dari `data.Name`.

49. URL Parsing

Data string url bisa dikonversi kedalam bentuk `url.URL`. Dengan menggunakan tipe tersebut akan ada banyak informasi yang bisa kita manfaatkan, diantaranya adalah jenis protokol yang digunakan, path yang diakses, query, dan lainnya.

Berikut adalah contoh sederhana konversi string ke `url.URL`.

```
package main

import "fmt"
import "net/url"

func main() {
    var urlString = "http://depeloper.com:80/hello?name=john wick&age=27"
    var u, e = url.Parse(urlString)
    if e != nil {
        fmt.Println(e.Error())
        return
    }

    fmt.Printf("url: %s\n", urlString)

    fmt.Printf("protocol: %s\n", u.Scheme) // http
    fmt.Printf("host: %s\n", u.Host)       // depeloper.com:80
    fmt.Printf("path: %s\n", u.Path)       // /hello

    var name = u.Query()["name"][0] // john wick
    var age = u.Query()["age"][0]   // 27
    fmt.Printf("name: %s, age: %s\n", name, age)
}
```

Fungsi `url.Parse()` digunakan untuk parsing string ke bentuk url. Mengembalikan 2 data, variabel objek bertipe `url.URL` dan error (jika ada). Lewat variabel objek tersebut pengaksesan informasi url akan menjadi lebih mudah, contohnya seperti nama host bisa didapatkan lewat `u.Host`, protokol lewat `u.Scheme`, dan lainnya.

Selain itu, query yang ada pada url akan otomatis diparsing juga, menjadi bentuk `map[string][]string`, dengan key adalah nama elemen query, dan value array string yang berisikan value elemen query.

```
[novalagung:belajar-golang $ go run bab49.go ]  
url: http://localhost:8080/hello?name=john wick&age=27  
protocol: http  
host: localhost:8080  
path: /hello  
name: john wick, age: 27  
novalagung:belajar-golang $ █
```

50. JSON

JSON atau *Javascript Object Notation* adalah notasi standar yang umum digunakan untuk komunikasi data via web. JSON merupakan subset dari *javascript*.

Golang menyediakan package `encoding/json` yang berisikan banyak fungsi untuk kebutuhan operasi json.

Di bab ini, kita akan belajar cara untuk konverstri string yang berbentuk json menjadi objek golang, dan sebaliknya.

50.1. Decode JSON Ke Variabel Objek Cetakan Struct

Di Golang data json tipenya adalah `string`. Dengan menggunakan `json.Unmarshal`, json string bisa dikonversi menjadi bentuk objek, entah itu dalam bentuk `map[string]interface{}` ataupun variabel objek hasil `struct`.

Program berikut ini adalah contoh cara decoding json ke bentuk objek. Pertama import package yang dibutuhkan, lalu siapkan struct `User`.

```
package main

import "encoding/json"
import "fmt"

type User struct {
    FullName string `json:"Name"`
    Age      int
}
```

Hasil decode nantinya akan disimpan ke variabel objek cetakan struct `User`.

Selanjutnya siapkan data json string sederhan. Perlu dilakukan casting ke tipe `[]byte`, karena fungsi `json.Unmarshal` hanya menerima data bertipe `[]byte`.

```

func main() {
    var jsonString = `{"Name": "john wick", "Age": 27}`
    var jsonData = []byte(jsonString)

    var data User

    var err = json.Unmarshal(jsonData, &data)
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    fmt.Println("user :", data.FullName)
    fmt.Println("age  :", data.Age)
}

```

Dalam penggunaan fungsi `json.Unmarshal`, variabel penampung hasil decode harus di-pass dalam bentuk pointer, contohnya seperti `&data`.

```
[novalagung:belajar-golang $ go run bab50.go
user : john wick
age  : 27
novalagung:belajar-golang $ ]
```

Bisa dilihat bahwa salah satu property struct `User`, yaitu `FullName` memiliki tag `json:"Name"`. Tag tersebut digunakan untuk mapping data json ke property yang bersangkutan.

Data json yang akan diparsing memiliki 2 property yaitu `Name` dan `Age`. Kebetulan penulisan `Age` pada data json dan pada struktur struct adalah sama, berbeda dengan `Name` yang tidak ada pada struct.

Property `FullName` struct tersebut kemudian ditugaskan untuk menampung data json property `Name`, ditandai dengan penambahan tag `json:"Name"` pada saat deklarasi structnya.

Perlu diketahui bahwa untuk decode data json ke variabel objek hasil struct, semua level akses property struct-nya harus publik.

50.2. Decode JSON Ke `map[string]interface{}` & `interface{}`

Tak hanya ke objek cetakan struct, target decoding data json juga bisa berupa variabel bertipe `map[string]interface{}`.

```

var data1 map[string]interface{}
json.Unmarshal(jsonData, &data1)

fmt.Println("user :", data1["Name"])
fmt.Println("age  :", data1["Age"])

```

Variabel bertipe `interface{}` juga bisa digunakan untuk menampung hasil decode. Dengan catatan pada pengaksesan nilai property, harus dilakukan casting terlebih dahulu ke `map[string]interface{} .`

```

var data2 interface{}
json.Unmarshal(jsonData, &data2)

var decodedData = data2.(map[string]interface{})
fmt.Println("user :", decodedData["Name"])
fmt.Println("age  :", decodedData["Age"])

```

50.3. Decode Array JSON Ke Array Objek

Decode data dari array json ke slice/array objek masih sama, siapkan saja variabel penampung hasil decode dengan tipe slice struct. Contohnya bisa dilihat pada kode berikut.

```

var jsonString = `[
    {"Name": "john wick", "Age": 27},
    {"Name": "ethan hunt", "Age": 32}
]` 

var data []User

var err = json.Unmarshal([]byte(jsonString), &data)
if err != nil {
    fmt.Println(err.Error())
    return
}

fmt.Println("user 1:", data[0].FullName)
fmt.Println("user 2:", data[1].FullName)

```

50.4. Encode Objek Ke JSON

Setelah sebelumnya dijelaskan beberapa cara decode data dari json json ke objek, sekarang kita akan belajar cara **encode** data objek ke bentuk json string.

Fungsi `json.Marshal` digunakan untuk decoding data ke json string. Sumber data bisa berupa variabel objek cetakan struct, `map[string]interface{}`, atau slice.

Pada contoh berikut, data slice struct dikonversi ke dalam bentuk json string. Hasil konversi berupa `[]byte`, casting terlebih dahulu ke tipe `string` agar bisa ditampilkan bentuk json string-nya.

```
var object = []User{{"john wick", 27}, {"ethan hunt", 32}}
var jsonData, err = json.Marshal(object)
if err != nil {
    fmt.Println(err.Error())
    return
}

var jsonString = string(jsonData)
fmt.Println(jsonString)
```

Output:

```
[novalagung:belajar-golang $ go run bab50.go
[{"Name":"john wick","Age":27}, {"Name":"ethan hunt","Age":32}]
novalagung:belajar-golang $ ]
```

51. Web API JSON

Pada bab ini kita akan mengkombinasikan pembahasan 2 bab sebelumnya, yaitu web dan JSON, untuk membuat sebuah web API dengan tipe data reponse berbentuk JSON.

Web API adalah sebuah web yang menerima request dari client dan menghasilkan response, biasa berupa JSON/XML. Di bab ini kita akan buat tanpa authentikasi.

51.1. Pembuatan Web API

Pertama siapkan terlebih dahulu struct dan beberapa data sample.

```
package main

import "encoding/json"
import "net/http"
import "fmt"

type student struct {
    ID      string
    Name    string
    Grade   int
}

var data = []student{
    student{"E001", "ethan", 21},
    student{"W001", "wick", 22},
    student{"B001", "bourne", 23},
    student{"B002", "bond", 23},
}
```

Struct `student` di atas digunakan sebagai tipe elemen array sample data, ditampung variabel `data`.

Selanjutnya buat fungsi `users()` untuk handle rute `/users`. Didalam fungsi tersebut ada proses deteksi jenis request lewat property `r.Method()`, untuk mencari tahu apakah jenis request adalah **POST** atau **GET** atau lainnya.

```
func users(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "application/json")

    if r.Method == "POST" {
        var result, err = json.Marshal(data)

        if err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
            return
        }

        w.Write(result)
        return
    }

    http.Error(w, "", http.StatusBadRequest)
}
```

Jika request adalah POST, maka data yang di-encode ke JSON dijadikan sebagai response.

Statement `w.Header().Set("Content-Type", "application/json")` digunakan untuk menentukan tipe response, yaitu sebagai JSON. Sedangkan `r.write()` digunakan untuk mendaftarkan data sebagai response.

Selebihnya, jika request tidak valid, response di set sebagai error menggunakan fungsi `http.Error()`.

Siapkan juga handler untuk rute `/user`. Perbedaan rute ini dengan rute `/users` di atas adalah:

- `/users` menghasilkan semua sample data yang ada (array).
- `/user` menghasilkan satu buah data saja, diambil dari data sample berdasarkan `ID`-nya. Pada rute ini, client harus mengirimkan juga informasi `ID` data yang dicari.

```

func user(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "application/json")

    if r.Method == "POST" {
        var id = r.FormValue("id")
        var result []byte
        var err error

        for _, each := range data {
            if each.ID == id {
                result, err = json.Marshal(each)

                if err != nil {
                    http.Error(w, err.Error(), http.StatusInternalServerError)
                    return
                }

                w.Write(result)
                return
            }
        }

        http.Error(w, "User not found", http.StatusBadRequest)
        return
    }

    http.Error(w, "", http.StatusBadRequest)
}

```

Method `r.FormValue()` digunakan untuk mengambil data form yang dikirim dari client, pada konteks ini data yang dimaksud adalah `ID`.

Dengan menggunakan `ID` tersebut dicarilah data yang relevan. Jika ada, maka dikembalikan sebagai response. Jika tidak ada maka error **400, Bad Request** dikembalikan dengan pesan **User Not Found**.

Terakhir, implementasikan kedua handler di atas.

```

func main() {
    http.HandleFunc("/users", users)
    http.HandleFunc("/user", user)

    fmt.Println("starting web server at http://localhost:8080/")
    http.ListenAndServe(":8080", nil)
}

```

Jalankan program, sekarang web server sudah live dan bisa dikonsumsi datanya.

```
[novalagung:belajar-golang $ go run bab51.go
starting web server at http://localhost:8080/]
```

51.2. Test API

Setelah web server sudah berjalan, web API yang telah dibuat perlu untuk di tes. Di sini saya menggunakan Google Chrome plugin bernama [Postman](#) untuk mengetes API yang sudah dibuat.

- Test `/users`, apakah data yang dikembalikan sudah benar.

http://localhost:8080/users

POST

Body Cookies (3) Headers (3) STATUS 200 OK TIME 10 ms

Pretty Raw Preview JSON XML

```

1 [
2   {
3     "ID": "E001",
4     "Name": "ethan",
5     "Grade": 21
6   },
7   {
8     "ID": "W001",
9     "Name": "wick",
10    "Grade": 22
11  },
12  {
13    "ID": "B001",
14    "Name": "bourne",
15    "Grade": 23
16  },
17  {
18    "ID": "B002",
19    "Name": "bond",
20    "Grade": 23
21  }
22 ]

```

- Test `/user`, isi form data `id` dengan nilai `E001`.

http://localhost:8080/user

POST

id E001

Body Cookies (3) Headers (3) STATUS 200 OK TIME 10 ms

Pretty Raw Preview JSON XML

```

1 {
2   "ID": "E001",
3   "Name": "ethan",
4   "Grade": 21
5 }

```


52. HTTP Request

Di bab sebelumnya kita telah belajar tentang bagaimana membuat Web API yang mem-provide data JSON, pada bab ini kita akan belajar mengenai cara untuk mengkonsumsi data tersebut.

Pastikan anda sudah mempraktekkan apa-apa yang ada pada bab sebelumnya (bab 51), karena web api server yang sudah dibuat pada bab sebelumnya kita juga pada bab ini.

```
[novalagung:belajar-golang $ go run bab51.go
starting web server at http://localhost:8080/]
```

52.1. Penggunaan HTTP Request

Package `net/http`, selain berisikan tools untuk keperluan pembuatan web, juga berisikan fungsi-fungsi untuk melakukan http request. Salah satunya adalah `http.NewRequest()` yang akan kita bahas di sini.

Sebelumnya, import package yang dibutuhkan. Dan siapkan struct `student` yang nantinya akan dipakai sebagai tipe data reponse dari web API. Struk tersebut skema nya sama dengan yang ada pada bab 51.

```
package main

import "fmt"
import "net/http"
import "encoding/json"

var baseURL = "http://localhost:8080"

type student struct {
    ID     string
    Name   string
    Grade int
}
```

Setelah itu buat fungsi `fetchusers()`. Fungsi ini bertugas melakukan request ke <http://localhost:8080/users>, menerima response dari request tersebut, lalu menampilkannya.

```

func fetchUsers() ([]student, error) {
    var err error
    var client = &http.Client{}
    var data []student

    request, err := http.NewRequest("POST", baseURL+"/users", nil)
    if err != nil {
        return nil, err
    }

    response, err := client.Do(request)
    if err != nil {
        return nil, err
    }
    defer response.Body.Close()

    err = json.NewDecoder(response.Body).Decode(&data)
    if err != nil {
        return nil, err
    }

    return data, nil
}

```

Statement `&http.Client{}` menghasilkan instance `http.Client`. Objek ini nantinya diperlukan untuk eksekusi request.

Fungsi `http.NewRequest()` digunakan untuk membuat request baru. Fungsi tersebut memiliki 3 parameter yang wajib diisi.

1. Parameter pertama, berisikan tipe request **POST** atau **GET** atau lainnya
2. Parameter kedua, adalah URL tujuan request
3. Parameter ketiga, form data request (jika ada)

Fungsi tersebut menghasilkan instance bertipe `http.Request`. Objek tersebut nantinya disisipkan pada saat eksekusi request.

Cara eksekusi request sendiri adalah dengan memanggil method `Do()` pada instance `http.Client` yang sudah dibuat, dengan parameter adalah instance request-nya. Contohnya seperti pada `client.Do(request)`.

Method tersebut mengembalikan instance bertipe `http.Response`, yang didalamnya berisikan informasi yang dikembalikan dari web API.

Data response bisa diambil lewat property `Body` dalam bentuk string. Gunakan JSON Decoder untuk mengkonversinya menjadi bentuk JSON. Contohnya bisa dilihat di kode di atas, `json.NewDecoder(response.Body).Decode(&data)`. Setelah itu barulah kita bisa menampilkannya.

Perlu diketahui, data response perlu di-**close** setelah tidak dipakai. Caranya seperti pada kode `defer response.Body.Close()`.

Implementasikan fungsi `fetchUsers()` di atas pada `main`.

```
func main() {
    var users, err = fetchUsers()
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }

    for _, each := range users {
        fmt.Printf("ID: %s\t Name: %s\t Grade: %d\n", each.ID, each.Name, each.Grade)
    }
}
```

Jalankan program untuk mengetes hasilnya.

```
[novalagung:belajar-golang $ go run bab52.go
ID: E001      Name: ethan      Grade: 21
ID: W001      Name: wick      Grade: 22
ID: B001      Name: bourne    Grade: 23
ID: B002      Name: bond      Grade: 23
novalagung:belajar-golang $ ]
```

52.3. HTTP Request Dengan Form Data

Untuk menyisipkan data pada sebuah request, ada beberapa hal yang perlu ditambahkan. Yang pertama, import beberapa package lagi, `bytes` dan `net/url`.

```
import "bytes"
import "net/url"
```

Buat fungsi baru, isinya request ke <http://localhost:8080/user> dengan data yang disisipkan adalah `ID`.

```

func fetchUser(ID string) (student, error) {
    var err error
    var client = &http.Client{}
    var data student

    var param = url.Values{}
    param.Set("id", ID)
    var payload = bytes.NewBufferString(param.Encode())

    request, err := http.NewRequest("POST", baseURL+"/user", payload)
    if err != nil {
        return data, err
    }
    request.Header.Set("Content-Type", "application/x-www-form-urlencoded")

    response, err := client.Do(request)
    if err != nil {
        return data, err
    }
    defer response.Body.Close()

    err = json.NewDecoder(response.Body).Decode(&data)
    if err != nil {
        return data, err
    }

    return data, nil
}

```

Isi fungsi di atas bisa dilihat memiliki beberapa kemiripan dengan fungsi `fetchusers()` sebelumnya.

Statement `url.Values{}` akan menghasilkan objek yang nantinya digunakan sebagai form data request. Pada objek tersebut perlu di set data apa saja yang ingin dikirimkan menggunakan fungsi `Set()` seperti pada `param.Set("id", ID)`.

Statement `bytes.NewBufferString(param.Encode())` maksudnya, objek form data di-encode lalu diubah menjadi bentuk `bytes.Buffer`, yang nantinya disisipkan pada parameter ketiga pemanggilan fungsi `http.NewRequest()`.

Karena data yang akan dikirim di-encode, maka pada header perlu di set tipe konten request-nya. Kode `request.Header.Set("Content-Type", "application/x-www-form-urlencoded")` artinya tipe konten request di set sebagai `application/x-www-form-urlencoded`.

Pada konteks HTML, HTTP Request yang di trigger dari tag `<form></form>` secara default tipe konten-nya sudah di set `application/x-www-form-urlencoded`. Lebih detailnya bisa merujuk ke spesifikasi HTML form <http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.1>

Response dari rute `/user` bukan berupa array objek, melainkan sebuah objek. Maka pada saat decode pastikan tipe variabel penampung hasil decode data response adalah `student` (bukan `[]student`).

Terakhir buat implementasinya pada fungsi `main`.

```
func main() {
    var user1, err = fetchUser("E001")
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }

    fmt.Printf("ID: %s\t Name: %s\t Grade: %d\n", user1.ID, user1.Name, user1.Grade)
}
```

Pada kode di atas `ID` ditentukan nilainya `"E001"`. Jalankan program untuk mengetes apakah data yang dikembalikan sesuai.

```
[novalagung:belajar-golang $ go run bab52.go
ID: E001      Name: ethan      Grade: 21
novalagung:belajar-golang $ ] E
```

53. SQL

Golang menyediakan package `database/sql` berisikan generic interface untuk keperluan interaksi dengan database sql. Package ini hanya bisa digunakan ketika **driver** database engine yang dipilih juga ada.

Ada cukup banyak sql driver yang tersedia untuk Golang, detailnya bisa diakses di <https://github.com/golang/go/wiki/SQLDrivers>. Beberapa diantaranya:

- MySql
- Oracle
- MS Sql Server
- dan lainnya

Driver-driver tersebut merupakan project open source yang diinisiasi oleh komunitas di Github. Artinya kita selaku developer juga bisa ikut berkontribusi didalamnya.

Pada bab ini kita akan belajar bagaimana berkomunikasi dengan database MySQL menggunakan driver [Go MySQL Driver](#).

53.1. Instalasi Driver

Unduh driver mysql menggunakan `go get .`

```
go get github.com/go-sql-driver/mysql
```

```
[novalagung:belajar-golang $ go get github.com/go-sql-driver/mysql
[novalagung:belajar-golang $ ls $GOPATH/src/github.com/go-sql-driver/mysql ]
 AUTHORS           collations.go      packets.go
 CHANGELOG.md     connection.go    result.go
 CONTRIBUTING.md  const.go        rows.go
 LICENSE          driver.go       statement.go
 README.md        driver_test.go  transaction.go
 appengine.go     errors.go      utils.go
 benchmark_test.go errors_test.go utils_test.go
 buffer.go        infile.go
 novalagung:belajar-golang $ ]
```

53.2. Setup Database

Sebelumnya pastikan sudah ada [mysql server](#) yang terinstal di lokal anda.

Buat database baru bernama `db_belajar_golang`, dan tabel baru bernama `tb_student`.

```

CREATE TABLE IF NOT EXISTS `tb_student` (
  `id` varchar(5) NOT NULL,
  `name` varchar(255) NOT NULL,
  `age` int(11) NOT NULL,
  `grade` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO `tb_student` (`id`, `name`, `age`, `grade`) VALUES
('B001', 'Jason Bourne', 29, 1),
('B002', 'James Bond', 27, 1),
('E001', 'Ethan Hunt', 27, 2),
('W001', 'John Wick', 28, 2);

ALTER TABLE `tb_student` ADD PRIMARY KEY (`id`);

```

53.3. Membaca Data Dari MySQL Server

Import package yang dibutuhkan, lalu disiapkan struct dengan skema yang sama seperti pada tabel `tb_student` di database. Nantinya struct ini digunakan sebagai tipe data penampung hasil query.

```

package main

import "fmt"
import "database/sql"
import _ "github.com/go-sql-driver/mysql"

type student struct {
    id    string
    name  string
    age   int
    grade int
}

```

Driver database yang digunakan perlu di-import menggunakan tanda `_`, karena meskipun dibutuhkan oleh package `database/sql`, kita tidak langsung berinteraksi dengan driver tersebut.

Selanjutnya buat fungsi untuk koneksi ke database.

```

func connect() (*sql.DB, error) {
    db, err := sql.Open("mysql", "root:@tcp(127.0.0.1:3306)/db_belajar_golang")
    if err != nil {
        return nil, err
    }

    return db, nil
}

```

Fungsi `sql.Open()` digunakan untuk memulai koneksi dengan database. Fungsi tersebut memiliki 2 parameter mandatory, nama driver dan **connection string**.

Skema connection string untuk driver mysql yang kita gunakan cukup unik, `root@tcp(127.0.0.1:3306)/db_belajar_golang`. Dibawah ini merupakan skema connection string yang bisa digunakan pada driver Go MySQL Driver. Jika anda menggunakan driver mysql lain, skema koneksinya bisa saja berbeda tergantung driver yang digunakan.

```

user:password@tcp(host:port)/dbname
user@tcp(host:port)/dbname

```

Di bawah ini adalah penjelasan mengenai connection string yang digunakan pada fungsi `connect()`.

```

root@tcp(127.0.0.1:3306)/db_belajar_golang
// user      => root
// password =>
// host      => 127.0.0.1 atau localhost
// port      => 3306
// dbname   => db_belajar_golang

```

Setelah fungsi untuk konektivitas dengan database sudah dibuat, saatnya untuk mempraktekan proses pembacaan data dari server database. Siapkan fungsi `sqlQuery()` dengan isi adalah kode berikut.

```

func sqlQuery() {
    db, err := connect()
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    defer db.Close()

    var age = 27
    rows, err := db.Query("select id, name, grade from tb_student where age = ?", age)
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    defer rows.Close()

    var result []student

    for rows.Next() {
        var each = student{}
        var err = rows.Scan(&each.id, &each.name, &each.grade)

        if err != nil {
            fmt.Println(err.Error())
            return
        }

        result = append(result, each)
    }

    if err = rows.Err(); err != nil {
        fmt.Println(err.Error())
        return
    }

    for _, each := range result {
        fmt.Println(each.name)
    }
}

```

Setiap kali terbuat koneksi baru, jangan lupa untuk selalu **close** instance konesinya. Bisa menggunakan keyword `defer` seperti pada kode di atas, `defer db.Close()`.

Fungsi `db.Query()` digunakan untuk eksekusi sql query. Fungsi tersebut parameter keduanya adalah variadic, sehingga boleh tidak diisi. Pada kode di atas bisa dilihat bahwa nilai salah satu clause `where` adalah tanda tanya (`?`). Tanda tersebut kemudian akan ter-replace oleh nilai pada parameter setelahnya (nilai variabel `age`). Teknik penulisan query sejenis ini sangat dianjurkan, untuk mencegah [sql injection](#).

Fungsi tersebut menghasilkan instance bertipe `sql.*Rows`, yang juga perlu di **close** ketika sudah tidak digunakan (`defer rows.Close()`).

Selanjutnya, sebuah array dengan tipe elemen struct `student` disiapkan dengan nama `result`. Nantinya hasil query akan ditampung ke variabel tersebut.

Kemudian dilakukan perulangan dengan acuan kondisi adalah `rows.Next()`. Perulangan dengan cara ini dilakukan sebanyak jumlah total record yang ada, berurutan dari record pertama hingga akhir, satu per satu.

Method `Scan()` milik `sql.Rows` berfungsi untuk mengambil nilai record yang sedang diiterasi, untuk disimpan pada variabel pointer. Variabel yang digunakan untuk menyimpan field-field record dituliskan berurutan sebagai parameter variadic, sesuai dengan field yang di select pada query. Silakan lihat perbandingan dibawah ini untuk lebih jelasnya.

```
// query
select id, name, grade ...

// scan
rows.Scan(&each.id, &each.name, &each.grade ...)
```

Data record yang didapat kemudian di-append ke slice `result`, lewat statement `result = append(result, each)`.

OK, sekarang tinggal panggil fungsi `sqlQuery()` di `main`, lalu jalankan program.

```
func main() {
    sqlQuery()
}
```

Output:

```
[novalagung:belajar-golang $ go run bab53.go
James Bond
Ethan Hunt
novalagung:belajar-golang $ ]
```

53.4. Membaca 1 Record Data Menggunakan Method `QueryRow()`

Untuk query yang menghasilkan 1 baris record saja, bisa gunakan method `QueryRow()`, dengan metode ini kode menjadi lebih ringkas. Chain dengan method `Scan()` untuk mendapatkan value-nya.

```

func sqlQueryRow() {
    var db, err = connect()
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    defer db.Close()

    var result = student{}
    var id = "E001"
    err = db.
        QueryRow("select name, grade from tb_student where id = ?", id).
        Scan(&result.name, &result.grade)
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    fmt.Printf("name: %s\ngrade: %d\n", result.name, result.grade)
}

func main() {
    sqlQueryRow()
}

```

Dari kode di atas ada statement yang dituliskan cukup unik, chain statement boleh dituliskan dalam beberapa baris, contohnya:

```

err = db.
    QueryRow("select name, grade from tb_student where id = ?", id).
    Scan(&result.name, &result.grade)

```

Sekarang jalankan program. Outputnya akan muncul data record sesuai id.

```

[novalagung:belajar-golang $ go run bab53.go
name: Ethan Hunt
grade: 2
novalagung:belajar-golang $ ]

```

53.5. Eksekusi Query Menggunakan Prepare()

Teknik **prepared statement** adalah teknik penulisan query di awal dengan kelebihan bisa di re-use atau digunakan banyak kali untuk eksekusi yang berbeda-beda.

Metode ini bisa digabung dengan `Query()` maupun `QueryRow()`. Berikut merupakan contoh penerapannya.

```

func sqlPrepare() {
    db, err := connect()
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    defer db.Close()

    stmt, err := db.Prepare("select name, grade from tb_student where id = ?")
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    var result1 = student{}
    stmt.QueryRow("E001").Scan(&result1.name, &result1.grade)
    fmt.Printf("name: %s\ngrade: %d\n", result1.name, result1.grade)

    var result2 = student{}
    stmt.QueryRow("W001").Scan(&result2.name, &result2.grade)
    fmt.Printf("name: %s\ngrade: %d\n", result2.name, result2.grade)

    var result3 = student{}
    stmt.QueryRow("B001").Scan(&result3.name, &result3.grade)
    fmt.Printf("name: %s\ngrade: %d\n", result3.name, result3.grade)
}

func main() {
    sqlPrepare()
}

```

Method `Prepare()` digunakan untuk deklarasi query, yang mengembalikan objek bertipe `sql.*Stmt`. Dari objek tersebut, dipanggil method `QueryRow()` beberapa kali dengan isi value untuk `id` berbeda-beda untuk tiap pemanggilannya.

```
[novalagung:belajar-golang $ go run bab53.go
name: Ethan Hunt
grade: 2
name: John Wick
grade: 2
name: Jason Bourne
grade: 1
novalagung:belajar-golang $ ]
```

53.6. Insert, Update, & Delete Data Menggunakan `Exec()`

Untuk operasi **insert**, **update**, dan **delete**; dianjurkan untuk tidak menggunakan fungsi `sql.Query()` ataupun `sql.QueryRow()` untuk eksekusinya. Direkomendasikan eksekusi perintah-perintah tersebut lewat fungsi `Exec()`, contohnya seperti pada kode berikut.

```
func sqlExec() {
    db, err := connect()
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    defer db.Close()

    _, err = db.Exec("insert into tb_student values (?, ?, ?, ?)", "G001", "Galahad",
29, 2)
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    fmt.Println("insert success!")

    _, err = db.Exec("update tb_student set age = ? where id = ?", 28, "G001")
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    fmt.Println("update success!")

    _, err = db.Exec("delete from tb_student where id = ?", "G001")
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    fmt.Println("delete success!")
}

func main() {
    sqlExec()
}
```

Teknik prepared statement juga bisa digunakan pada metode ini. Berikut adalah perbandingan eksekusi `Exec()` menggunakan `Prepare()` dan cara biasa.

```
// menggunakan metode prepared statement
stmt, err := db.Prepare("insert into tb_student values (?, ?, ?, ?)")
stmt.Exec("G001", "Galahad", 29, 2)

// menggunakan metode biasa
_, err := db.Exec("insert into tb_student values (?, ?, ?, ?)", "G001", "Galahad", 29,
2)
```

53.7. Koneksi Dengan Engine Database Lain

Karena package `database/sql` merupakan interface generic, maka cara untuk koneksi ke engine database lain (semisal Oracle, Postgres, SQL Server) adalah sama dengan cara koneksi ke MySQL. Cukup dengan meng-import driver yang digunakan, lalu mengganti nama driver pada saat pembuatan koneksi baru.

```
sql.Open(driverName, connectionString)
```

Sebagai contoh saya menggunakan driver `pq` untuk koneksi ke server Postgres, maka connection string-nya:

```
sql.Open("pq", "user=postgres password=secret dbname=test sslmode=disable")
```

Selengkapnya mengenai driver yang tersedia bisa dilihat di
<https://github.com/golang/go/wiki/SQLDrivers>.

-
- [Go MySQL Driver](#), by Julien Schmidt, MPL-2.0 license

54. NoSQL MongoDB

Golang tidak menyediakan interface generic untuk NoSQL, jadi implementasi driver tiap brand NoSQL di Golang bisa berbeda satu dengan lainnya.

Dari sekian banyak teknologi NoSQL yang ada, yang terpilih untuk dibahas di buku ini adalah MongoDB. Dan pada bab ini kita akan belajar cara berkomunikasi dengan MongoDB menggunakan driver [mgo](#).

54.1. Persiapan

Ada beberapa hal yang perlu disiapkan sebelum mulai masuk ke bagian coding.

1. Instal mgo menggunakan `go get` .

```
go get gopkg.in/mgo.v2
```

```
[novalagung:belajar-golang $ go get gopkg.in/mgo.v2
novalagung:belajar-golang $ ]
```

2. Pastikan sudah terinstal MongoDB di komputer anda, dan jangan lupa untuk menjalankan daemon-nya. Jika belum, [download](#) dan install terlebih dahulu.
3. Instal juga MongoDB GUI untuk mempermudah browsing data. Bisa menggunakan [MongoChef](#), [Robomongo](#), atau lainnya.

54.2. Insert Data

Cara insert data lewat mongo tidak terlalu sulit. Kita akan praktekan bagaimana caranya.

Import package yang dibutuhkan, lalu siapkan struct model.

```
package main

import "fmt"
import "gopkg.in/mgo.v2"

type student struct {
    Name string `bson:"name"`
    Grade int    `bson:"Grade"`
}
```

Tag `bson` pada property struct dalam konteks `mgo`, digunakan sebagai penentu nama field ketika data disimpan kedalam collection. Jika sebuah property tidak memiliki tag `bson`, secara default nama field adalah sama dengan nama property hanya saja lowercase. Untuk customize nama field, gunakan tag `bson`.

Pada contoh di atas, property `Name` ditentukan nama field nya sebagai `name`, dan `Grade` sebagai `grade`.

Selanjutnya siapkan fungsi untuk membuat session baru.

```
func connect() (*mgo.Session, error) {
    var session, err = mgo.Dial("localhost")
    if err != nil {
        return nil, err
    }
    return session, nil
}
```

Fungsi `mgo.Dial()` digunakan untuk membuat objek session baru, dengan tipe `*mgo.Session`. Fungsi ini memiliki satu parameter yang harus diisi, yaitu connection string dari server mongo yang akan diakses.

Secara default jenis konsistensi session yang digunakan adalah `mgo.Primary`. Anda bisa mengubahnya lewat method `SetMode()` milik struct `mgo.Session`. Lebih jelasnya silakan merujuk <https://godoc.org/gopkg.in/mgo.v2#Session.SetMode>.

Terkahir buat fungsi `insert` yang didalamnya berisikan kode untuk insert data ke mongodb, lalu implementasikan di `main`.

```

func insert() {
    var session, err = connect()
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }
    defer session.Close()

    var collection = session.DB("belajar_golang").C("student")
    err = collection.Insert(&student{"Wick", 2}, &student{"Ethan", 2})
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }

    fmt.Println("Insert success!")
}

func main() {
    insert()
}

```

Session di mgo juga harus di close ketika sudah tidak digunakan, seperti pada instance connection di bab SQL. Statement `defer session.Close()` akan mengakhirkan proses close session dalam fungsi `insert()`.

Struct `mgo.Session` memiliki method `DB()`, digunakan untuk memilih database, dan bisa langsung di chain dengan fungsi `C()` untuk memilih collection.

Setelah mendapatkan instance collection-nya, gunakan method `Insert()` untuk insert data ke database. Method ini memiliki parameter variadic, harus diisi pointer data yang ingin di-insert.

Jalankan program tersebut, lalu cek menggunakan mongo GUI untuk melihat apakah data sudah masuk.

Key	Value	Type
▼ (1) {_id : 562b58c30f04d83...}	{ 3 fields }	Document
▀ _id	562b58c30f04d83cdd873ad2	ObjectId
▀ name	Wick	String
▀ Grade	2	Int32
▼ (2) {_id : 562b58c30f04d83...}	{ 3 fields }	Document
▀ _id	562b58c30f04d83cdd873ad3	ObjectId
▀ name	Ethan	String
▀ Grade	2	Int32

54.3. Membaca Data

method `Find()` milik tipe collection `mgo.Collection` digunakan untuk melakukan pembacaan data. Query selectornya dituliskan menggunakan `bson.M` lalu disisipkan sebagai parameter fungsi `Find()`.

Untuk menggunakan `bson.M`, package `gopkg.in/mgo.v2/bson` harus di-import terlebih dahulu.

```
import "gopkg.in/mgo.v2/bson"
```

Setelah itu buat fungsi `find` yang didalamnya terdapat proses baca data dari database.

```
func find() {
    var session, err = connect()
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }
    defer session.Close()
    var collection = session.DB("belajar_golang").C("student")

    var result = student{}
    var selector = bson.M{"name": "Wick"}
    err = collection.Find(selector).One(&result)
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }

    fmt.Println("Name : ", result.Name)
    fmt.Println("Grade : ", result.Grade)
}

func main() {
    find()
}
```

Variabel `result` di-inisialisasi menggunakan struct `student`. Variabel tersebut nantinya digunakan untuk menampung hasil pencarian data.

query selector ditulis dalam tipe `bson.M`. Tipe ini sebenarnya adalah alias dari `map[string]interface{}`.

Selector tersebut kemudian dimasukan sebagai parameter method `Find()`, yang kemudian di chain langsung dengan method `One()` untuk mengambil 1 baris datanya. Pointer variabel `result` disisipkan sebagai parameter method tersebut.

```
[novalagung:belajar-golang $ go run bab54.go
Name : Wick
Grade : 2
novalagung:belajar-golang $ ]
```

54.4. Update Data

Method `Update()` milik struct `mgo.Collection` digunakan untuk update data. Ada 2 parameter yang harus diisi:

1. Parameter pertama adalah query selector data yang ingin di update
2. Parameter kedua adalah data perubahannya

Di bawah ini adalah contoh implementasi method `Update()`.

```
func update() {
    var session, err = connect()
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }
    defer session.Close()
    var collection = session.DB("belajar_golang").C("student")

    var selector = bson.M{"name": "Wick"}
    var changes = student{"John Wick", 2}
    err = collection.Update(selector, changes)
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }

    fmt.Println("Update success!")
}

func main() {
    update()
}
```

Jalankan kode di atas, lalu cek lewat Mongo GUI apakah data berubah.

Key	Value	Type
▼ (1) {_id : 562b594f0f04d83c...}	{ 3 fields }	Document
`_id	562b594f0f04d83cdd873ad6	ObjectId
"_name	John Wick	String
`Grade	2	Int32
▼ (2) {_id : 562b594f0f04d83c...}	{ 3 fields }	Document
`_id	562b594f0f04d83cdd873ad7	ObjectId
"_name	Ethan	String
`Grade	2	Int32

54.5. Menghapus Data

Cara menghapus document pada collection cukup mudah, tinggal gunakan method `Remove()` dengan isi parameter adalah query selector document yang ingin dihapus.

```
func remove() {
    var session, err = connect()
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }
    defer session.Close()
    var collection = session.DB("belajar_golang").C("student")

    var selector = bson.M{"name": "John Wick"}
    err = collection.Remove(selector)
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }

    fmt.Println("Remove success!")
}

func main() {
    remove()
}
```

2 data yang sebelumnya sudah di-insert kini tinggal satu saja.

Key	Value	Type
▼ (1) {_id : 562b594f0f04d83c...}	{ 3 fields }	Document
_id	562b594f0f04d83cdd873ad7	ObjectId
name	Ethan	String
Grade	2	Int32

55. Unit Test

Golang menyediakan package `testing`, yang berisikan banyak sekali tools untuk keperluan unit testing.

Pada bab ini kita akan belajar mengenai testing, benchmark, dan juga testing menggunakan [testify](#).

55.1. Persiapan

Pertama siapkan terlebih dahulu sebuah struct `Kubus`. Variabel object hasil struct ini nantinya kita gunakan sebagai bahan testing.

```
package main

import "math"

type Kubus struct {
    Sisi float64
}

func (k Kubus) Volume() float64 {
    return math.Pow(k.Sisi, 3)
}

func (k Kubus) Luas() float64 {
    return math.Pow(k.Sisi, 2) * 6
}

func (k Kubus) Keliling() float64 {
    return k.Sisi * 12
}
```

Simpan kode di atas dengan nama `bab55.go`.

55.2. Testing

File untuk keperluan testing dipisah dengan file utama, namanya harus berakhiran `_test.go`, dan package-nya harus sama. Pada bab ini, file utama adalah `bab55.go`, maka file testing harus bernama `bab55_test.go`.

Unit test di Golang dituliskan dalam bentuk fungsi, yang memiliki parameter yang bertipe `*testing.T`, dengan nama fungsi harus diawali kata **Test** (pastikan sudah meng-import package `testing` sebelumnya). Lewat parameter tersebut, kita bisa mengakses method-method untuk keperluan testing.

Pada contoh di bawah ini disiapkan 3 buah fungsi test, yang masing-masing digunakan untuk mengecek apakah hasil kalkulasi volume, luas, dan keliling kubus adalah benar.

```
package main

import "testing"

var (
    kubus           Kubus    = Kubus{4}
    volumeSeharusnya float64 = 64
    luasSeharusnya float64 = 96
    kelilingSeharusnya float64 = 48
)

func TestHitungVolume(t *testing.T) {
    t.Logf("Volume : %.2f", kubus.Volume())

    if kubus.Volume() != volumeSeharusnya {
        t.Errorf("SALAH! harusnya %.2f", volumeSeharusnya)
    }
}

func TestHitungLuas(t *testing.T) {
    t.Logf("Luas : %.2f", kubus.Luas())

    if kubus.Luas() != luasSeharusnya {
        t.Errorf("SALAH! harusnya %.2f", luasSeharusnya)
    }
}

func TestHitungKeliling(t *testing.T) {
    t.Logf("Keliling : %.2f", kubus.Keliling())

    if kubus.Keliling() != kelilingSeharusnya {
        t.Errorf("SALAH! harusnya %.2f", kelilingSeharusnya)
    }
}
```

Method `t.Logf()` digunakan untuk memunculkan log. Method ini equivalen dengan `fmt.Printf()`.

Method `Errorf()` digunakan untuk memunculkan log dengan diikuti keterangan bahwa terjadi **fail** pada saat testing.

Cara eksekusi testing adalah menggunakan command `go test`. Karena struct yang diuji berada dalam file `bab55.go`, maka pada saat eksekusi test menggunakan `go test`, nama file `bab55_test.go` dan `bab55.go` perlu dituliskan sebagai argument.

Argument `-v` atau verbose digunakan menampilkan semua output log pada saat pengujian.

Jalankan aplikasi seperti gambar dibawah ini, terlihat bahwa tidak ada test yang fail.

```
[novalagung:belajar-golang $ go test bab55.go bab55_test.go -v
    === RUN TestHitungVolume
    --- PASS: TestHitungVolume (0.00s)
        bab55_test.go:13: Volume : 64.00
    === RUN TestHitungLuas
    --- PASS: TestHitungLuas (0.00s)
        bab55_test.go:21: Luas : 96.00
    === RUN TestHitungKeliling
    --- PASS: TestHitungKeliling (0.00s)
        bab55_test.go:29: Keliling : 48.00
PASS
ok      command-line-arguments  0.005s
novalagung:belajar-golang $ ]
```

OK, selanjutnya coba ubah rumus kalkulasi method `Keliling()`. Tujuan dari pengubahan ini adalah untuk mengetahui bagaimana penanda fail muncul ketika ada test yang gagal.

```
func (k Kubus) Keliling() float64 {
    return k.Sisi * 15
}
```

Setelah itu jalankan lagi test.

```
[novalagung:belajar-golang $ go test bab55.go bab55_test.go -v
    === RUN TestHitungVolume
    --- PASS: TestHitungVolume (0.00s)
        bab55_test.go:13: Volume : 64.00
    === RUN TestHitungLuas
    --- PASS: TestHitungLuas (0.00s)
        bab55_test.go:21: Luas : 96.00
    === RUN TestHitungKeliling
    --- FAIL: TestHitungKeliling (0.00s)
        bab55_test.go:29: Keliling : 60.00
        bab55_test.go:32: SALAH! harusnya 48.00
FAIL
exit status 1
FAIL      command-line-arguments  0.005s
novalagung:belajar-golang $ ]
```

55.3. Method Test

Table berikut berisikan method standar testing yang bisa digunakan di Golang.

Method	Kegunaan
Log()	Menampilkan log
Logf()	Menampilkan log menggunakan format
Fail()	Menandakan terjadi Fail() dan proses testing fungsi tetap diteruskan
FailNow()	Menandakan terjadi Fail() dan proses testing fungsi dihentikan
Failed()	Menampilkan laporan fail
Error()	Log() diikuti dengan Fail()
Errorf()	Logf() diikuti dengan Fail()
Fatal()	Log() diikuti dengan failNow()
Fatalf()	Logf() diikuti dengan failNow()
Skip()	Log() diikuti dengan SkipNow()
Skipf()	Logf() diikuti dengan SkipNow()
SkipNow()	Menghentikan proses testing fungsi, dilanjutkan ke testing fungsi setelahnya
Skipped()	Menampilkan laporan skip
Parallel()	Menge-set bahwa eksekusi testing adalah parallel

55.4. Benchmark

Package `testing` selain berisikan tools untuk testing juga berisikan tools untuk benchmarking. Cara pembuatan benchmark sendiri cukup mudah yaitu dengan membuat fungsi yang namanya diawali dengan **Benchmark** dan parameternya bertipe `*testing.B`.

Sebagai contoh, kita akan mengetes performa perhitungan luas kubus. Siapkan fungsi dengan nama `BenchmarkHitungLuas()` dengan isi adalah kode berikut.

```
func BenchmarkHitungLuas(b *testing.B) {
    for i := 0; i < b.N; i++ {
        kubus.Luas()
    }
}
```

Jalankan test menggunakan argument `-bench=.`, argumen ini digunakan untuk menandai bahwa selain testing terdapat juga benchmark yang perlu diuji.

```
[novalagung:belajar-golang $ go test bab55.go bab55_test.go -bench=.
]
PASS
BenchmarkHitungLuas      30000000          51.4 ns/op
ok   command-line-arguments 1.598s
novalagung:belajar-golang $ ]
```

Arti dari `30000000 51.1 ns/op` adalah, fungsi di atas di-test sebanyak **30 juta** kali, hasilnya membutuhkan waktu rata-rata **51 nano detik** untuk run satu fungsi.

55.5. Testing Menggunakan `testify`

Package **testify** berisikan banyak sekali tools yang bisa dimanfaatkan untuk keperluan testing di Golang.

Testify bisa di-download pada github.com/stretchr/testify menggunakan `go get`.

Didalam `testify` terdapat 5 package dengan kegunaan berbeda-beda satu dengan lainnya. Detailnya bisa dilihat pada tabel berikut.

Package	Kegunaan
assert	Berisikan tools standar untuk testing
http	Berisikan tools untuk keperluan testing http
mock	Berisikan tools untuk mocking object
require	Sama seperti assert, hanya saja jika terjadi fail pada saat test akan menghentikan eksekusi program
suite	Berisikan tools testing yang berhubungan dengan struct dan method

Pada bab ini akan kita contohkan bagaimana penggunaan package `assert`. Silakan perhatikan contoh berikut.

```
import "github.com/stretchr/testify/assert"

...

func TestHitungVolume(t *testing.T) {
    assert.Equal(t, kubus.Volume(), volumeSeharusnya, "perhitungan volume salah")
}

func TestHitungLuas(t *testing.T) {
    assert.Equal(t, kubus.Luas(), luasSeharusnya, "perhitungan luas salah")
}

func TestHitungKeliling(t *testing.T) {
    assert.Equal(t, kubus.Keliling(), kelilingSeharusnya, "perhitungan keliling salah")
}
```

Fungsi `assert.Equal()` digunakan untuk uji perbandingan. Parameter ke-2 dibandingkan nilainya dengan parameter ke-3. Jika tidak sama, maka pesan parameter ke-3 akan dimunculkan.

```
[novalagung:belajar-golang $ go test bab55.go bab55_test.go -v
==== RUN TestHitungVolume
--- PASS: TestHitungVolume (0.00s)
==== RUN TestHitungLuas
--- PASS: TestHitungLuas (0.00s)
==== RUN TestHitungKeliling
--- FAIL: TestHitungKeliling (0.00s)
    Error Trace:    bab55_test.go:24
    Error:          Not equal: 60 (expected)
                  != 48 (actual)
    Messages:       perhitungan keliling salah

FAIL
exit status 1
FAIL      command-line-arguments  0.008s
novalagung:belajar-golang $ ]
```

- [Testify](#), by "Stetchr, Inc"

56. WaitGroup

Sebelumnya kita telah belajar banyak mengenai channel, yang fungsi utama-nya adalah untuk sharing data antar goroutine. Selain untuk komunikasi data, channel secara tidak langsung bisa dimanfaatkan untuk mengontrol goroutine.

Golang menyediakan package `sync`, berisi cukup banyak API untuk handle masalah multiprocessing (goroutine), salah satunya diantaranya adalah yang kita bahas di bab ini, yaitu `sync.WaitGroup`.

Kegunaan `sync.WaitGroup` adalah untuk sinkronisasi goroutine. Berbeda dengan channel, `sync.WaitGroup` memang didesain khusus untuk maintain goroutine, penggunaannya lebih mudah dan lebih efektif dibanding channel.

Sebenarnya kurang pas membandingkan `sync.WaitGroup` dan channel, karena fungsi utama dari keduanya adalah berbeda. Channel untuk keperluan sharing data antar goroutine, sedangkan `sync.WaitGroup` untuk sinkronisasi goroutine.

56.1. Penerapan `sync.WaitGroup`

`sync.WaitGroup` digunakan untuk menunggu goroutine. Cara penggunaannya sangat mudah, tinggal masukan jumlah goroutine yang dieksekusi, sebagai parameter method `Add()` object cetakan `sync.WaitGroup`. Dan pada akhir tiap-tiap goroutine, panggil method `Done()`. Juga, pada baris kode setelah eksekusi goroutine, panggil method `Wait()`.

Agar lebih jelas, silakan coba kode berikut.

```

package main

import "sync"
import "runtime"
import "fmt"

func doPrint(wg *sync.WaitGroup, message string) {
    defer wg.Done()
    fmt.Println(message)
}

func main() {
    runtime.GOMAXPROCS(2)

    var wg sync.WaitGroup

    for i := 0; i < 5; i++ {
        var data = fmt.Sprintf("data %d", i)

        wg.Add(1)
        go doPrint(&wg, data)
    }

    wg.Wait()
}

```

Kode di atas merupakan contoh penerapan `sync.WaitGroup` untuk pengelolahan goroutine. Fungsi `doPrint()` akan dijalankan sebagai goroutine, dengan tugas menampilkan isi variabel `message`.

Variabel `wg` disiapkan bertipe `sync.WaitGroup`, dibuat untuk sinkronisasi goroutines yang dijalankan.

Di tiap perulangan statement `wg.Add(1)` dipanggil. Kode tersebut akan memberikan informasi kepada `wg` bahwa jumlah goroutine yang sedang diproses ditambah 1 (karena dipanggil 5 kali, maka `wg` akan sadar bahwa terdapat 5 buah goroutine sedang berjalan).

Di baris selanjutnya, fungsi `doPrint()` dieksekusi sebagai goroutine. Didalam fungsi tersebut, sebuah method bernama `Done()` dipanggil. Method ini digunakan untuk memberikan informasi kepada `wg` bahwa goroutine dimana method itu dipanggil sudah selesai. Sejumlah 5 buah goroutine dijalankan, maka method tersebut harus dipanggil 5 kali.

Statement `wg.Wait()` bersifat blocking, proses eksekusi program tidak akan diteruskan ke baris selanjutnya, sebelum sejumlah 5 goroutine selesai. Jika `Add(1)` dipanggil 5 kali, maka `Done()` juga harus dipanggil 5 kali.

Output program di atas.

```
[novalagung:belajar-golang $ go run bab56.go
data 4
data 2
data 3
data 0
data 1
[novalagung:belajar-golang $ go run bab56.go
data 4
data 1
data 2
data 3
data 0
novalagung:belajar-golang $ ]]
```

56.2. Perbedaan WaitGroup Dengan Channel

Beberapa perbedaan antara channel dan `sync.WaitGroup` :

- Channel tergantung kepada goroutine tertentu dalam penggunaannya, tidak seperti `sync.WaitGroup` yang dia tidak perlu tahu goroutine mana saja yang dijalankan, cukup tahu jumlah goroutine yang harus selesai
- Penerapan `sync.WaitGroup` lebih mudah dibanding channel
- Kegunaan utama channel adalah untuk komunikasi data antar goroutine. Sifatnya yang blocking bisa kita manfaatkan untuk manage goroutine; sedangkan WaitGroup khusus digunakan untuk sinkronisasi goroutine
- Performa `sync.WaitGroup` lebih baik dibanding channel, sumber:
<https://groups.google.com/forum/#topic/golang-nuts/wphCEk9yLhc>

Kombinasi yang tepat antara `sync.WaitGroup` dan channel sangat penting, dibutuhkan untuk handle proses concurrent agar bisa maksimal.

57. Mutex

Sebelum kita membahas mengenai apa itu **mutex**? ada baiknya untuk mempelajari terlebih dahulu apa itu **race condition**, karena kedua konsep ini berhubungan erat satu sama lain.

Race condition adalah kondisi dimana lebih dari satu goroutine, mengakses data yang sama pada waktu yang bersamaan (benar-benar bersamaan). Ketika hal ini terjadi, nilai data tersebut akan menjadi kacau. Dalam **concurrency programming** situasi seperti ini ini sering terjadi.

Mutex melakukan pengubahan level akses sebuah data menjadi eksklusif, menjadikan data tersebut hanya dapat dikonsumsi (read / write) oleh satu buah goroutine saja. Ketika terjadi race condition, maka hanya goroutine yang beruntung saja yang bisa mengakses data tersebut. Goroutine lain (yang waktu running nya kebetulan bersamaan) akan dipaksa untuk menunggu, hingga goroutine yang sedang memanfaatkan data tersebut selesai.

Golang menyediakan `sync.Mutex` yang bisa dimanfaatkan untuk keperluan **lock** dan **unlock** data. Pada bab ini kita akan membahas mengenai race condition, dan menanggulanginya menggunakan mutex.

57.1. Persiapan

Pertama siapkan struct baru bernama `counter`, dengan isi satu buah property `val` bertipe `int`. Property ini nantinya dikonsumsi dan diolah oleh banyak goroutine.

Lalu buat beberapa method struct `counter`.

1. Method `Add()`, untuk increment nilai.
2. Method `value()`, untuk mengembalikan nilai.

```
package main

import (
    "fmt"
    "runtime"
    "sync"
)

type counter struct {
    val int
}

func (c *counter) Add(x int) {
    c.val++
}

func (c *counter) Value() (x int) {
    return c.val
}
```

Kode di atas kita gunakan sebagai template contoh source code yang ada pada bab ini.

57.2. Contoh Race Condition

Program berikut merupakan contoh program yang didalamnya memungkinkan terjadi race condition atau kondisi goroutine balapan.

Pastikan jumlah core prosesor komputer anda adalah lebih dari satu. Karena contoh pada bab ini hanya akan berjalan sesuai harapan jika `GOMAXPROCS > 1`.

```

func main() {
    runtime.GOMAXPROCS(2)

    var wg sync.WaitGroup
    var meter counter

    for i := 0; i < 1000; i++ {
        wg.Add(1)

        go func() {
            for j := 0; j < 1000; j++ {
                meter.Add(1)
            }
        }

        wg.Done()
    }()

    wg.Wait()
    fmt.Println(meter.Value())
}

```

Pada kode diatas, disiapkan sebuah instance `sync.WaitGroup` bernama `wg`, dan variabel object `meter` bertipe `counter` (nilai property `val` default-nya adalah `0`).

Setelahnya dijalankan perulangan sebanyak 1000 kali, yang ditiap perulangannya dijalankan sebuah goroutine baru. Didalam goroutine tersebut, terdapat perulangan lagi, sebanyak 1000 kali. Dalam perulangan tersebut nilai property `val` dinaikkan sebanyak 1 lewat method `Add()`.

Dengan demikian, ekspektasi nilai akhir `meter.val` harusnya adalah 1000000.

Di akhir, `wg.Wait()` dipanggil, dan nilai variabel counter `meter` diambil lewat `meter.Value()` untuk kemudian ditampilkan.

Jalankan program, lihat hasilnya.

```

[novalagung:belajar-golang $ go run bab57.go
754541
[novalagung:belajar-golang $ go run bab57.go
753642
[novalagung:belajar-golang $ go run bab57.go
729100
novalagung:belajar-golang $ ]

```

Nilai `meter.val` tidak genap 1000000? kenapa bisa begitu? Padahal seharusnya tidak ada masalah dalam kode yang kita tulis di atas.

Inilah yang disebut dengan race condition, data yang diakses bersamaan dalam 1 waktu menjadi kacau.

57.3. Deteksi Race Condition Menggunakan Golang Race Detector

Golang menyediakan fitur untuk [deteksi race condition](#). Cara penggunaannya adalah dengan menambahkan flag `-race` pada saat eksekusi aplikasi.

```
[novalagung:belajar-golang $ go run -race bab57.go
=====
WARNING: DATA RACE
Read by goroutine 7:
  main.main.func1()
    /Users/novalagung/Documents/go/src/belajar-golang/bab57.go:30 +0x46

Previous write by goroutine 6:
  main.main.func1()
    /Users/novalagung/Documents/go/src/belajar-golang/bab57.go:30 +0x5a

Goroutine 7 (running) created at:
  main.main()
    /Users/novalagung/Documents/go/src/belajar-golang/bab57.go:34 +0xe8

Goroutine 6 (finished) created at:
  main.main()
    /Users/novalagung/Documents/go/src/belajar-golang/bab57.go:34 +0xe8
=====
679625
Found 1 data race(s)
exit status 66
novalagung:belajar-golang $ ]
```

Terlihat pada gambar diatas, ada pesan memberitahu terdapat kemungkinan data race pada program yang kita jalankan.

57.4. Penerapan sync.Mutex

Sekarang kita tahu bahwa program di atas menghasilkan bug, ada kemungkinan data race didalamnya. Untuk mengatasi masalah ini ada beberapa cara yang bisa digunakan, dan disini kita akan menggunakan `sync.Mutex`.

Ubah kode di atas, embed struct `sync.Mutex` kedalam struct `counter`, agar lewat objek cetakan `counter` kita bisa melakukan lock dan unlock dengan mudah. Tambahkan method `Lock()` dan `Unlock()` didalam method `Add()`.

```

type counter struct {
    sync.Mutex
    val int
}

func (c *counter) Add(x int) {
    c.Lock()
    c.val++
    c.Unlock()
}

func (c *counter) Value() (x int) {
    return c.val
}

```

Method `Lock()` digunakan untuk menandai bahwa semua operasi pada baris setelah kode tersebut adalah bersifat eksklusif. Hanya ada satu buah goroutine yang bisa melakukannya dalam satu waktu. Jika ada banyak goroutine yang eksekusinya bersamaan, harus antri.

Pada kode di atas terdapat kode untuk increment nilai `meter.val`. Maka property tersebut hanya bisa diakses oleh satu goroutine saja.

Method `Unlock()` akan membuka kembali akses operasi ke property/variabel yang di lock, proses mutual exclusion terjadi diantara method `Lock()` dan `Unlock()`.

Di contoh di atas, pada saat bagian pengambilan nilai, mutex tidak dipasang, karena kebetulan pengambilan nilai terjadi setelah semua goroutine selesai. Data Race bisa terjadi saat pengubahan maupun pengambilan data, jadi penggunaan mutex harus disesuaikan dengan kasus.

Coba jalankan program, dan lihat hasilnya.

```

[novalagung:belajar-golang $ go run bab57.go
1000000
[novalagung:belajar-golang $ go run bab57.go
1000000
[novalagung:belajar-golang $ go run bab57.go
1000000
novalagung:belajar-golang $ ]

```

Penggunaan `sync.Mutex` yang dianjurkan adalah dengan cara langsung di embed ke struct dimana proses yang memungkin race condition berada. Kita bisa saja menggunakan mutex dengan cara biasa, berikut adalah contohnya.

```
func (c *counter) Add(x int) {
    c.val++
}

func (c *counter) Value() (x int) {
    return c.val
}

func main() {
    runtime.GOMAXPROCS(2)

    var wg sync.WaitGroup
    var mtx sync.Mutex
    var meter counter

    for i := 0; i < 1000; i++ {
        wg.Add(1)

        go func() {
            for j := 0; j < 1000; j++ {
                mtx.Lock()
                meter.Add(1)
                mtx.Unlock()
            }
            wg.Done()
        }()
    }

    wg.Wait()
    fmt.Println(meter.Value())
}
```