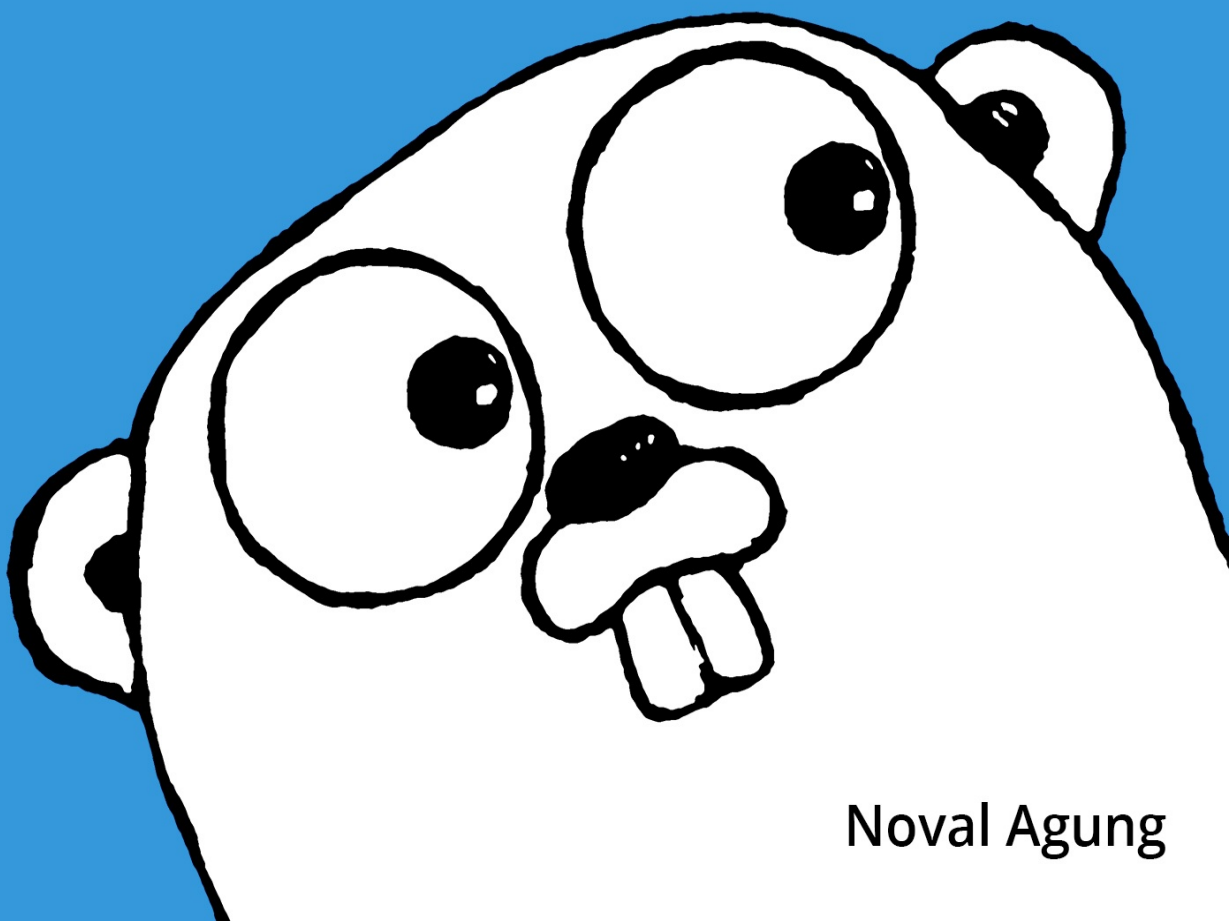
DASAR PEMROGRAMAN
GOLANG



Noval Agung

Table of Contents

Introduction	1.1
Berkenalan Dengan Golang	1.2
Instalasi Golang	1.3
GOPATH Dan Workspace	1.4
Instalasi Editor	1.5
Command	1.6
Program Pertama: Hello World	1.7
Komentar	1.8
Variabel	1.9
Tipe Data	1.10
Konstanta	1.11
Operator	1.12
Seleksi Kondisi	1.13
Perulangan	1.14
Array	1.15
Slice	1.16
Map	1.17
Fungsi	1.18
Fungsi Multiple Return	1.19
Fungsi Variadic	1.20
Fungsi Closure	1.21
Fungsi Sebagai parameter	1.22
Pointer	1.23
Struct	1.24
Method	1.25
Property Public & Private	1.26
Interface	1.27
Interface Kosong	1.28
Reflect	1.29
Goroutine	1.30

Channel	1.31
Buffered Channel	1.32
Channel - Select	1.33
Channel - Range & Close	1.34
Channel - Timeout	1.35
Defer & Exit	1.36
Error & Panic	1.37
Layout Format String	1.38
Time, Parsing Time, & Format Time	1.39
Timer	1.40
Konversi Data	1.41
Fungsi String	1.42
Regexp	1.43
Encode - Decode Base64	1.44
Hash Sha1	1.45
Arguments & Flag	1.46
Exec	1.47
File	1.48
Web	1.49
URL Parsing	1.50
JSON	1.51
Web JSON API	1.52
HTTP Request	1.53
SQL	1.54
MongoDB	1.55
Automated Test	1.56
WaitGroup	1.57
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 Github. Dianjurkan untuk tidak copy-paste dari source code dalam belajar, usahakan untuk menulis sendiri kode program, agar cepat terbiasa dengan bahasa Golang.

Versi 1.2016.05.23

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.

Jika berkenan, anda bisa membantu penulis dengan cara donasi lewat paypal.




DASAR PEMROGRAMAN
GOLANG

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.

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 *(pararel processing)*
- Memiliki garbage collector
- Proses kompilasi sangat cepat
- Bukan bahasa pemrograman yang hirarkial, 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

Meskipun bahasa ini masih berumur sekitar 5 tahunan, sudah banyak industri dan perusahaan yg menggunakan Golang sampai level production, termasuk diantaranya adalah Google sendiri.

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



Instalasi Golang

Hal pertama yang perlu dilakukan sebelum bisa menggunakan Golang adalah menginstalnya terlebih dahulu. Cara instalasinya berbeda-beda untuk tiap jenis sistem operasi. Panduan instalasi Golang sebenarnya sudah disediakan, bisa dilihat di situs official-nya http://golang.org/doc/install#install.

Bab ini merupakan ringkasan dari panduan instalasi yang disediakan oleh Golang.

> Di buku ini versi Golang yang digunakan adalah **1.4.2**

Instalasi Golang di Windows

1. Download terlebih dahulu installer-nya. Pilih sesuai jenis bit prosesor yang digunakan.
    - 32bit => go1.4.2.wind*ws-386.msi
    - 64bit => go1.4.2.wind*ws-amd64.msi
1. Setelah ter-download, jalankan installer, klik **next** sampai proses instalasi selesai. Default-nya Golang akan terinstal di `c:\go\bin`. Path tersebut akan secara otomatis terdaftar di **path variable**.
2. Buka **Command Prompt** / **CMD**, eksekusi perintah berikut untuk mengetes apakah Golang sudah terinstal dengan benar.

    ```
    $ go version
    ```

3. Jika output command di atas adalah versi Golang yang di-instal, maka instalasi berhasil.

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

Instalasi Golang di Mac OSX

Cara termudah instalasi Golang di **M*c OSX** adalah dengan menggunakan homebrew. Homebrew sendiri adalah **package manager** khusus untuk M*c OSX (mirip seperti `apt-get` milik Ubuntu).

Di bawah ini merupakan langkah instalasi Golang menggunakan homebrew.

1. Install terlebih dahulu homebrew (jika belum ada), dengan cara mengeksekusi perintah berikut di **terminal**.

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

2. Install Golang menggunakan command `brew`.

```
$ brew install go
```

3. Selanjutnya, eksekusi perintah di bawah ini untuk mengetes apakah golang sudah terinstal dengan benar.

```
$ go version
```

4. Jika output command di atas adalah versi Golang yang di-instal, maka instalasi berhasil.

Instalasi Golang di Ubuntu

Cara menginstal Golang di **Ub*ntu** bisa dengan memanfaatkan `apt-get`. Silakan ikuti petunjuk di bawah ini.

1. Jalankan command berikut di **terminal**.

```
$ sudo add-apt-repository ppa:gophers/go
$ sudo apt-get update
$ sudo apt-get install Golang-stable
```

2. Setelah instalasi selesai, eksekusi perintah di bawah ini untuk mengetes apakah Golang sudah terinstal dengan benar.

```
$ go version
```

3. Jika output command di atas adalah versi Golang yang di-instal, maka instalasi berhasil.

Instalasi Golang di Distro Linux Lain

1. Download archive berikut, pilih sesuai jenis bit komputer anda.
    - 32bit => go1.4.2.lin*x-386.tar.gz
    - 64bit => go1.4.2.lin*x-amd64.tar.gz

1. Buka **terminal**, ekstrak archive tersebut ke `/usr/local`. Setelah itu export path-nya. Gunakan command di bawah ini untuk melakukan hal tersebut.

```
$ tar zxvf go1.4.2.lin*x-....tar.gz -C /usr/local
$ export PATH=$PATH:/usr/local/go/bin
```

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

```
$ go version
```

3. Jika output command di atas adalah versi Golang yang di-instal, maka instalasi berhasil.

GOPATH Dan Workspace

Setelah Golang berhasil di-instal, ada hal yang perlu disiapkan sebelum bisa masuk ke sesi pembuatan aplikasi, yaitu setup workspace untuk proyek-proyek yang akan dibuat. Dan di bab ini kita akan belajar bagaimana caranya.

Variabel `GOPATH`

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

Proyek di Golang harus ditempatkan pada path `$GOPATH/src`. Sebagai contoh kita ingin membuat proyek dengan nama `belajar`, maka harus dibuatkan sebuah folder dengan nama `belajar` ditempatkan dalam `src` (`$GOPATH/src/belajar`). Nantinya semua file untuk keperluan proyek yang bersangkutan ditempatkan disana.

> Path separator yang digunakan sebagai contoh di buku ini adalah slash `/`. Khusus pengguna Wind*ws, path separator adalah backslah `\`.

Setup Workspace

Lokasi atau alamat folder yang akan dijadikan sebagai workspace bisa ditentukan sendiri. Anda bisa menggunakan alamat folder mana saja, bebas. Lokasi tersebut perlu disimpan kedalam path variable dengan nama `GOPATH`. Sebagai contoh, saya memilih path `$HOME/Documents/go`, maka saya daftarkan alamat tersebut. Caranya:

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

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

- Bagi pengguna non-**Wind*ws**, gunakan keyword `export` untuk mendaftarkan `GOPATH`.

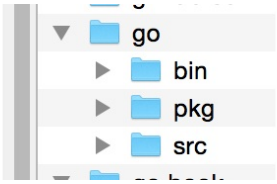
  ```
  $ export GOPATH=$HOME/Documents/go
  ```

  Setelah variabel didaftarkan, cek apakah path sudah terdaftar dengan benar.

```
[novalagung:~ $ source ~/.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 proyek golang 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.

Instalasi Editor

Pembuatan program menggunakan bahasa Golang, akan lebih maksimal jika didukung editor atau **IDE** yang pas. Ada cukup banyak pilihan bagus yang bisa dipertimbangkan, diantaranya: Intellij, Netbeans, Atom, Brackets, dan lainnya.

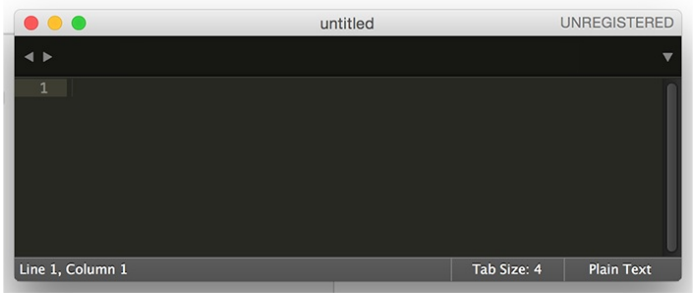
Pada saat menulis buku ini, editor yang saya gunakan adalah **Sublime Text 3**. Editor ini ringan, mudah didapat, dan memiliki cukup banyak plugin. Anda bisa memilih editor yang sama dengan yang digunakan di buku ini, atau editor lainnya, bebas, yang penting nyaman ketika digunakan.

Bagi yang memilih Sublime, saya sarankan untuk menginstall 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), perapian kode otomatis, dan lainnya.

Di bab ini akan dijelaskan bagaimana cara instalasi editor Sublime, package control, dan plugin GoSublime.

Instalasi Editor Sublime Text

1. Download **Sublime Text versi 3** di http://www.sublimetext.com/3, pilih sesuai dengan sistem operasi yang digunakan.
2. Setelah ter-download, buka file tersebut untuk memulai instalasi.
3. Setelah instalasi selesai, jalankan aplikasi.



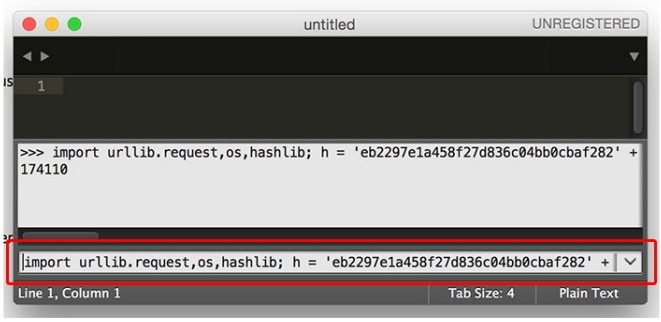
Instalasi Package Control

Package control merupakan aplikasi *3rd party* untuk Sublime, digunakan untuk mempermudah instalasi plugin. Default-nya Sublime tidak menyediakan aplikas ini, kita perlu menginstalnya 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).



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



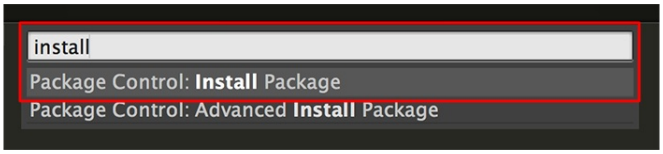
3. Tunggu hingga proses instalasi selesai. Perhatikan karakter sama dengan *(=)* di bagian kiri bawah editor yang bergerak-gerak. Jika karakter tersebut menghilang, menandakan bahwa proses instalasi sudah selesai.

4. Setelah selesai, tutup aplikasi, lalu buka kembali. Package control sudah berhasil di-install.

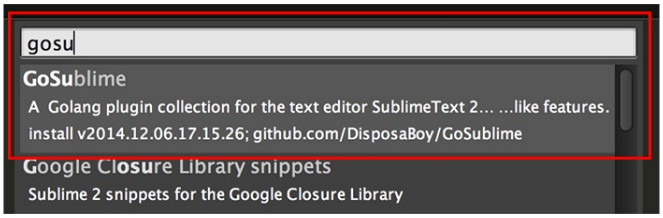
Instalasi Plugin GoSublime

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 *SX), akan muncul sebuah input dialog. Ketikan disana `install`, lalu tekan enter.



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



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

Command

Pengembangan aplikasi Golang tak jauh dari hal-hal yang berbau command line interface. Seperti kompilasi, testing, eksekusi program, semua dilakukan lewat command line.

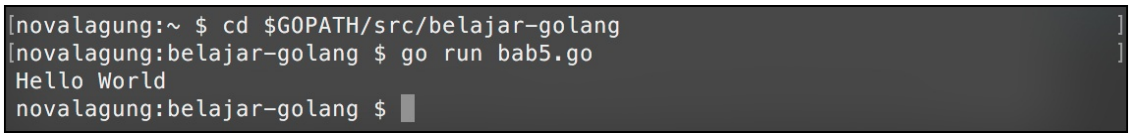
Golang menyediakan command `go` untuk keperluan pengembangan aplikasi. Di bab ini kita akan belajar mengenai pemanfaatannya.

Command `go run`

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

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

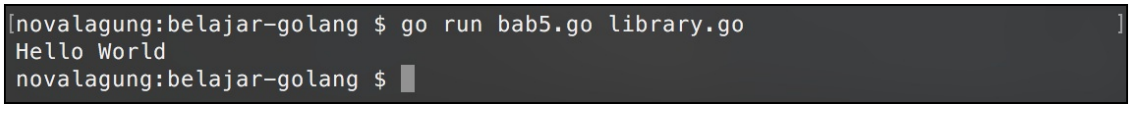
```
$ cd $GOPATH/src/belajar-golang
$ go run bab5.go
```



Command `go run` hanya bisa digunakan pada file yang package-nya adalah **main**. Untuk lebih jelasnya akan 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
```

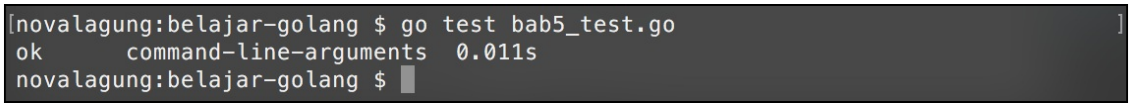


Command `go test`

Golang menyediakan package `testing` yang bisa dimanfaatkan 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
```

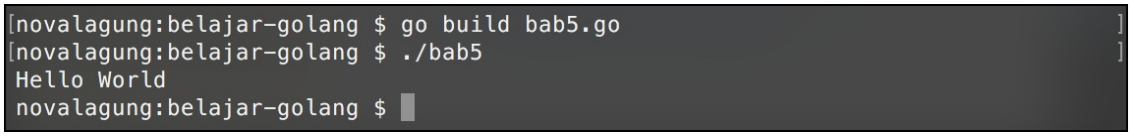


Command `go build`

Command ini digunakan untuk mengkompilasi file program.

Sebenarnya ketika eksekusi program menggunakan `go run`, terjadi proses kompilasi juga, hanya saja 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.



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

> Pada pengguna Wind*ws, file executable ber-ekstensi `.exe`.

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`

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
```

```
[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                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
 novalagung:src $
```

gopkg.in/mgo.v2 adalah alamat url package mgo. Package yang sudah ter-download akan tersimpan pada `$GOPATH/src` dengan struktur folder sesuai dengan url package-nya. Sebagai contoh, package mgo di atas tersimpan di `$GOPATH/src/gopkg.in/mgo.v2`.

Program Pertama: Hello World

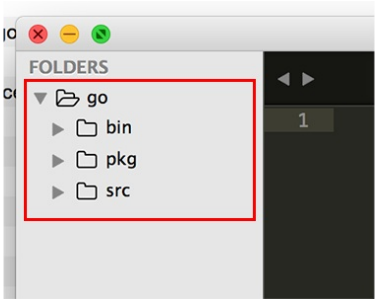
Semua persiapan sudah selesai, saatnya mulai masuk pada sesi pembuatan program. Program pertama yang akan dibuat adalah aplikasi sederhana untuk memunculkan tulisan **Hello World**.

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

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

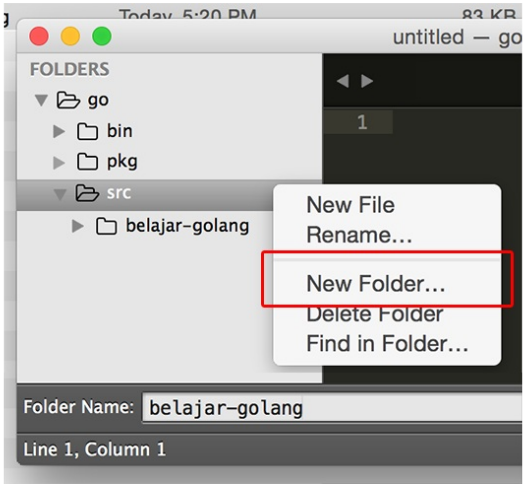


> Nama variabel di sistem operasi non-Wind*ws diawali dengan tanda dollar `$`, sebagai contoh `$GOPATH`. Sedangkan di Wind*ws, nama variabel diapit karakter persen `%`, contohnya seperti `%GOPATH%`.

Menyiapkan Folder Proyek

Selanjutnya kita siapkan sebuah proyek untuk keperluan pembuatan program. Buat direktori baru dalam `$GOPATH/src` dengan nama folder silakan ditentukan sendiri (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



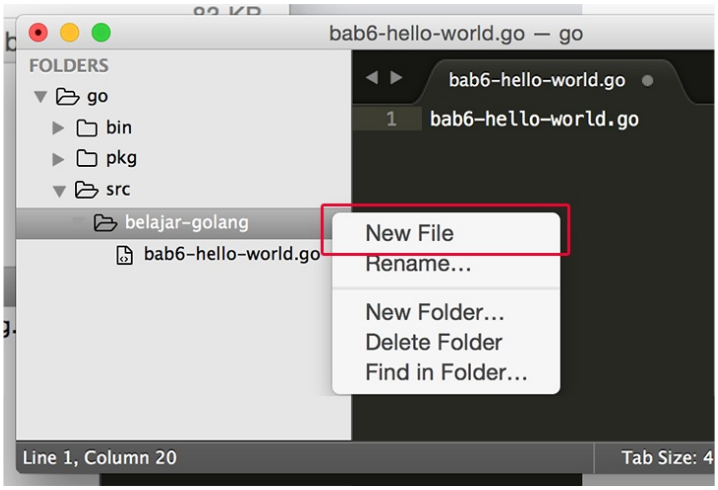
Menyiapkan File Program

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

Di dalam proyek yang telah dibuat (`$GOPATH/src/belajar-golang/`), perlu disiapkan 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
5. File akan terbuat



Program Pertama: Hello Word

Setelah folder proyek dan file program sudah siap, saatnya untuk mulai masuk ke sesi penulisan program atau **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 pemrograman Golang.

```
package main

import "fmt"

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

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

- Wind*ws

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

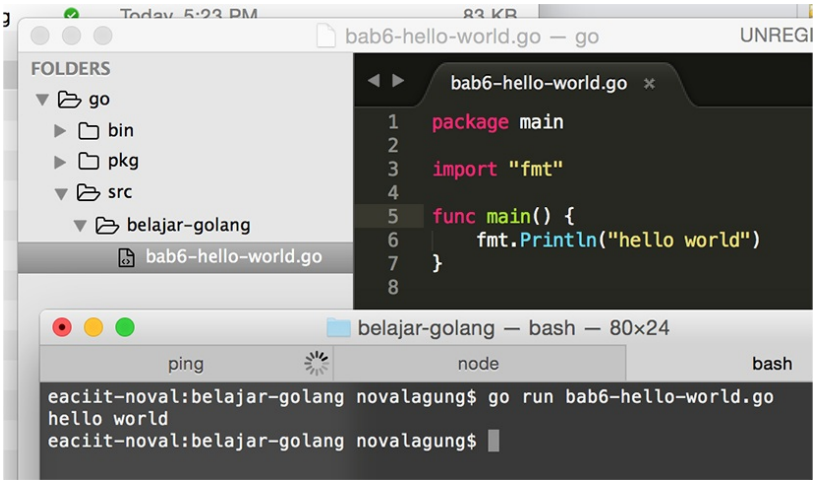
- Non-Wind*ws

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

Setelah itu jalankan program dengan perintah `go run` .

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

Akan muncul tulisan **hello world** di layar console.



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 mengenai kode di atas.

Penggunaan Keyword `package`

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

Cara penentuan package adalah menggunakan keyword `package` , berikut adalah contoh penggunaannya.

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

Penggunaan Keyword `import`

Keyword `import` digunakan untuk meng-include 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.

Berikut adalah skema penulisan keyword `import` beserta contohnya.

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

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.

Berikut merupakan contoh penulisannya.

```
func main() {

}
```

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**.

Berikut adalah skema penulisan fungsi `fmt.Println()` beserta contohnya.

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

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

Fungsi ini bisa menampung parameter yang tidak terbatas jumlahnya. Semua data parameter akan dimunculkan dengan pemisah tanda spasi. Contohnya bisa dilihat di kode berikut.

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

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

Komentar

Komentar biasa dimanfaatkan untuk menyisipkan catatan pada kode program, menulis penjelasan atau deskripsi mengenai suatu blok kode, atau bisa juga digunakan untuk me-remark 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.

Komentar Inline

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

Berikut ini merupakan contoh penulisan komentar inline.

```
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 praktekan kode di atas. Siapkan file program baru di proyek `belajar-golang` dengan nama bisa apa saja. 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.

Komentar Multiline

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.

Variabel

Golang mengadopsi dua jenis penulisan variabel, yang dituliskan tipe data-nya dan yang tidak. Kedua cara tersebut intinya adalah sama, pembedanya hanyalah cara penulisannya saja.

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

Deklarasi Variabel Dengan Tipe Data

Golang memiliki aturan cukup ketat dalam hal penulisan variabel. Pada saat deklarasinya, 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` 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. Cara tersebut diperbolehkan di Golang.

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

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 diisi langsung pada saat deklarasi variabel.

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.

Agar lebih mudah dipahami, silakan perhatikan kode berikut. 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 (*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 end line (baris baru) di akhir.

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, kontradiktif jika dibandingkan dengan cara pertama. Dengan metode jenis ini, keyword `var` dan tipe data tidak perlu ditulis. Agar lebih jelas, silakan perhatikan kode berikut.

```
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, dengan ketentuan tidak menggunakan tanda `:=`, melainkan tetap menggunakan `=`. Contohnya bisa dilihat pada kode berikut.

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

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

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

Tanda `:=` hanya digunakan sekali di awal pada saat deklarasi saja. Setelah itu, untuk assignment nilai selanjutnya harus menggunakan tanda `=`. Contoh:

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

Deklarasi Multi Variabel

Golang mendukung deklarasi bantak variabel secara bersamaan, caranya dengan menuliskan variabel-variabel-nya dengan pembatas tanda koma (`,`). Untuk pengisian nilainya-pun diperbolehkan secara bersamaan. Contoh:

```
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 (`,`). Contohnya seperti pada kode berikut.

```
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. Contoh:

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

Istimewa bukan? Istimewa sekali.

Variabel Underscore `_`

Golang memiliki aturan unik yang tidak 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, program akan gagal dikompilasi.

```
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. Berikut adalah contoh penggunaan variabel tersebut.

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

Pada contoh di atas, variabel `name` akan berisikan text `john` , sedang nilai `wick` akan 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 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 :-)

Deklarasi Variabel Menggunakan Keyword `new`

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

```
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, namun tak apa, karena nantinya di bab 22 akan dikupas habis tentang apa itu pointer dan dereference.

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, tapi tak apa. Ketika sudah masuk ke pembahasan masing-masing poin tersebut, akan terlihat apa kegunaan dari keyword ini.

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 menerapkan beberapa tipe data, seperti `string` dan tipe numerik `int`.

Pada bab ini, akan dijelaskan beberapa macam tipe data standar yang disediakan oleh Golang, dan bagaiman cara penggunaannya.

Tipe Data Numerik Non-Desimal

Tipe data numerik non-desimal atau **non floating point** di Golang ada beberapa macam. Secara umum ada 2 tipe data yang perlu diketahui, yaitu:

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

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

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

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.

Contoh penggunaan variabel numerik non-desimal bisa dilihat di kode berikut.

```
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.

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. Contoh penggunaan tipe data ini bisa dilihat di kode berikut.

```
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.

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). Contoh sederhana penggunaan `bool`:

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

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

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 akan di escape**, termasuk `\n`, tanda petik dua dan tanda petik satu, baris baru, dan lainnya. Semua akan terdeteksi sebagai string. Berikut adalah contoh penerapannya.

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

fmt.Println(message)
```

Ketika dijalankan, output akan muncul sama persisi 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 $
```

Nilai `nil` Dan Nilai Default Tipe Data

`nil` bukan merupakan tipe data, melainkan sebuah nilai. Variabel yang isi nilainya `nil`, berarti variabel tersebut 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 sudah 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` ketika sudah masuk pada pembahasan bab-bab tersebut.

Konstanta

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

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`. Contohnya:

```
const firstName string = "john"
fmt.Print("halo ", firstName, "!\n")
```

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

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

Penggunaan Fungsi `fmt.Print()`

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. Agar lebih mudah dipahami, perhatikan kode berikut.

```
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 di situ, karena fungsi ini tidak menambahkan spasi di sela-sela nilai parameter yang digabungkan.

Operator

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

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 penggunaannya:

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

Operator Perbandingan

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

Tabel berikut berisikan operator perbandingan yang bisa digunakan di Golang.

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

Contoh penggunaannya:

```
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`. Selanjurnya, 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 memakai layout format `%t`.

Operator Logika

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

Beberapa operator logika standar yang bisa digunakan:

Tanda	Penjelasan
`&&`	kiri **dan** kanan
`||`	kiri **atau** kanan
`!`	negasi / nilai kebalikan

Contoh penggunaannya:

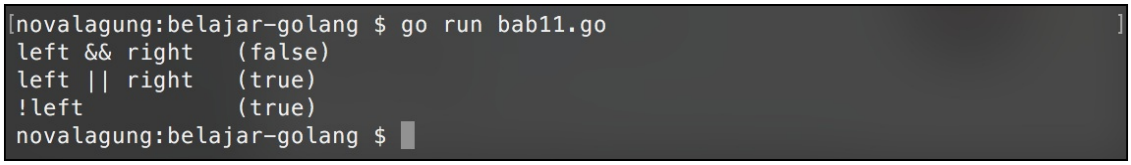
```
var left = false
var right = true

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.



Berikut adalah 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.

Seleksi Kondisi

Seleksi kondisi digunakan untuk mengontrol alur program. Kalau dianalogikan, fungsinya mirip seperti 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 akan diatur.

Yang dijadikan acuan oleh seleksi kondisi adalah nilai bertipe `bool`, bisa berasa dari variabel, ataupun hasil operasi perbandingan. Nilai tersebut akan 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")` akan menghasilkan error.

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

Cara penerapan if-else di Golang sama dengan pada bahasa pemrograman lain. Yang membedakan hanya tanda kurung kurawalnya, 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 ke-empat 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), text `"lulus"` akan muncul di console.

```
[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.

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.

Berikut merupakan contoh penerapannya.

```
var point = 8840.0

if percent := point / 100; percent >= 100 {
    fmt.Printf("%.1f%s perfect!\n", percent, "%")
} else if percent >= 70 {
    fmt.Printf("%.1f%s good\n", percent, "%")
} else {
    fmt.Printf("%.1f%s 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.

Seleksi Kondisi Menggunakan Keyword `switch`

Switch merupakan seleksi kondisi yang sifatnya fokus pada satu variabel. Contoh sederhananya seperti penentuan apakah nilai variabel `x` adalah: `1`, `2`, `3`, atau lainnya. Agar lebih jelas, silakan melihat contoh di bawah ini.

```
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` adalah `6`. Ketika hal seperti ini terjadi, blok kondisi `default` akan 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 pengecekkan 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`.

Pemanfaatan 1 `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 (`,`). Contoh bisa dilihat pada kode berikut.

```
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.

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.

Berikut adalah contoh penggunaannya. 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!")
    }
}
```

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")
    }
}
```

Penggunaan Keyword `fallthrough` Dalam `switch`

Seperti yang kita sudah singgung di atas, bahwa switch pada Golang memiliki beberapa perbedaan dengan bahasa lain. Ketika sebuah `case` terpenuhi, pengecekkan kondisi tidak akan diteruskan ke case-case setelahnya.

Keyword `fallthrough` digunakan untuk memaksa proses pengecekkan diteruskan ke `case` selanjutnya. Contoh berikut merupakan penerapan keyword ini.

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

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

Seleksi Kondisi Bersarang

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

```
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!")
        }
    }
}
```

Perulangan

Perulangan merupakan 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.

Perulangan Menggunakan Keyword `for`

Ada beberapa cara standar menggunakan `for`. Cara pertama dengan memasukkan variabel counter perulangan beserta kondisinya setelah keyword. Agar lebih mudah dipahami, coba perhatikan dan praktekan 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 $
```

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 perlu dituliskan. Konsepnya mirip seperti `while` milik bahasa pemrograman lain.

Kode berikut adalah contoh penerapannya. Output yang dihasilkan sama seperti pada penerapan for menggunakan cara pertama.

```
var i = 0

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

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 bisa dilakukan dengan menggunakan keyword `break` . Contoh penerapannya bisa dilihat di kode berikut.

```
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.

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).

Penggunaan Keyword `break` & `continue`

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

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

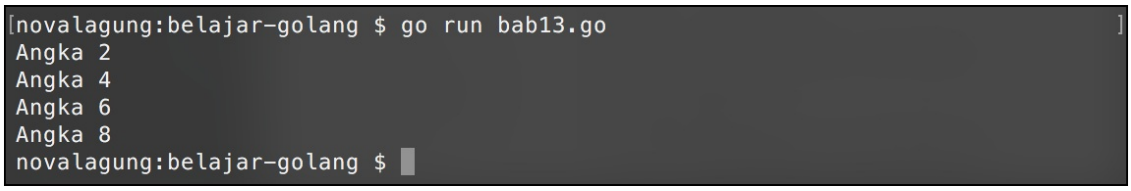
```
for i := 1; i <= 10; i++ {
    if i % 2 == 1 {
        continue
    }

    if i > 8 {
        break
    }

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

Kode di atas, saya kira 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



Perulangan Bersarang

Tak hanya seleksi kondisi yang bisa bersarang, perulangan juga bisa. Cara pengaplikasiannya kurang lebih sama, tinggal tulis perulangan didalam perulangan. Contohnya bisa dilihat di kode berikut.

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

    fmt.Println()
}
```

Di kode ini, untuk pertama kalinya fungsi `fmt.Println()` dipanggil tanpa disisipkan parameter. Cara seperti ini 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 $
```

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 penerapannya.

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

Tepat sebelum keyword `for` terluar, terdapat baris kode `outerLoop:` . Maksud dari kode itu 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 for yang label-nya adalah `outerLoop` . Perulangan yang memiliki label tersebut akan diberhentikan.

```
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 $
```

Array

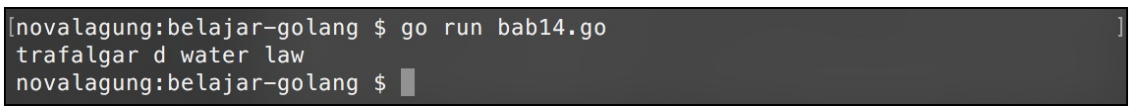
Array adalah kumpulan data bertipe sama, yang disimpan dalam sebuah variabel. Array memiliki kapasitas yang nilainya ditentukan pada saat pembuatan, menjadikan elemen/data yang disimpan di array tersebut jumlahnya tidak boleh melebihi yang sudah dialokasikan. Default nilai tiap elemen array pada awalnya 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.



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 (`,`). Berikut merupakan contohnya.

```
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()` digunakan untuk menghitung jumlah elemen sebuah array.

Inisialisasi Nilai Array Dengan Gaya Vertikal

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

```
var fruits [4]string

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

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

Perlu diperhatikan, khusus deklarasi menggunakan cara vertikal, perlu dituliskan tanda koma pada akhir elemen terakhir. Jika tidak ditulis akan muncul error.

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 diisikan.

```
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 ajaib ditentukan jumlah alokasinya yaitu `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 $
```

Array Multidimensi

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

Cara deklarasi array multidimensi secara umum sama dengan cara deklarasi array biasa. Cukup masukan data array yang merupakan 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 adalah sama nilainya.

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

Perulangan Elemen Array Menggunakan Keyword `for`

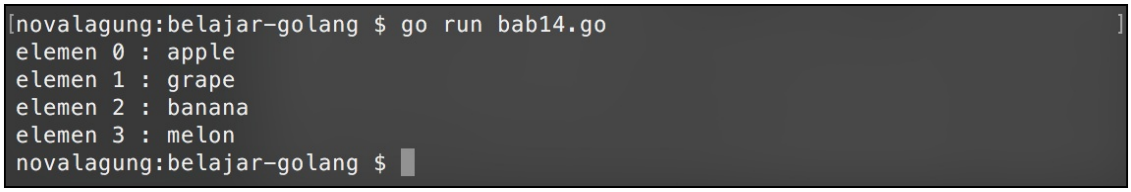
Keyword `for` dan array memiliki hubungan yang sangat erat. Dengan memanfaatkan perulangan menggunakan keyword ini, elemen-elemen dalam array bisa didapat.

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 dengan memanfaatkan variabel iterasi `i`.



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

Ada cara yang lebih mudah yang bisa dimanfaatkan untuk me-looping sebuah data array, yaitu 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.

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 seperti di 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.

```
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`. Contoh:

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

Alokasi Elemen Array Menggunakan Keyword `make`

Deklarasi sekaligus alokasi data array bisa dilakukan lewat keyword `make`. Contohnya bisa dilihat pada kode berikut.

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

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

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

Slice

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

Inisialisasi Slice

Cara membuat slice mirip seperti pada 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
```

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 sebelum indeks ke-2**. Elemen yang memenuhi kriteria tersebut kemudian dikembalikan, untuk disimpan pada variabel 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. Pengaksesan elemen lewat 2 buah indeks (seperti `data[0:2]`), mengembalikan slice atau elemen referensi.

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

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

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

Slice Merupakan Tipe Data Reference

Slice merupakan tipe reference. Artinya jika ada slice baru yang terbentuk dari slice lama, maka elemen slice baru memiliki referensi 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 "pinnaple"
baFruits[0] = "pinnaple"

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

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 "pinnaple"

fruits          [apple pinnaple banana melon]
aFruits         [apple pinnaple banana]
bFruits         [pinnaple banana melon]
aaFruits        [pinnaple]
baFruits        [pinnaple]
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 `"pinnaple"`, karena memiliki referensi yang sama.

Ada beberapa *built in function* bawaan Golang, yang bisa dimanfaatkan untuk keperluan operasi slice. Berikut adalah pembahasan mengenai fungsi-fungsi tersebut.

Fungsi `len()`

Fungsi `len()` digunakan untuk menghitung lebar slice yang ada. Sebagai contoh jika sebuah variabel adalah slice dengan data 4 buah, maka fungsi ini pada variabel tersebut akan mengembalikan angka **4**, yang angka tersebut didapat dari jumlah elemen yang ada. Contoh penerapannya bisa dilihat di kode berikut

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

Fungsi `cap()`

Fungsi `cap()` digunakan untuk menghitung lebar maksimum/kapasitas slice. Nilai kembalian fungsi ini awalnya sama dengan `len`, tapi bisa berubah tergantung dari 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. Maka fungsi `len(fruits)` dan `cap(fruits)` akan menghasilkan angka 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

Slicing yang dimulai dari indeks **0** hingga **x** akan mengembalikan elemen-elemen mulai indeks **0** hingga sebelum indeks **x**, dengan lebar kapasitas adalah sama dengan slice aslinya. Sedangkan slicing yang dimulai dari indeks **y**, yang dimana nilai **y** adalah lebih dari **0**, membuat elemen ke-**y** slice yang diambil menjadi elemen ke-0 slice baru. Hal inilah yang membuat kapasitas slice berubah.

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.

Fungsi `copy()`

Fungsi `copy()` digunakan untuk men-copy elemen slice tujuan (parameter ke-2), untuk digabungkan dengan slice target (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", "pinnaple"}

var copiedElemen = copy(fruits, aFruits)

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

Pengaksesan Elemen Slice Dengan 3 Indeks

3 index adalah teknik slicing elemen yang sekaligus menentukan kapasitasnya. Cara menggunakannnya 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
```

Map

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

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

Penggunaan Map

Cara menggunakan map adalah 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`. `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 variabel map cukup mudah, tinggal panggil variabel-nya, sisipkan `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-nya akan ditimpa dengan yang 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"`.

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 diinisialisasi dengan tanpa nilai awal, caranya cukup 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.

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 di 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 $
```

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.

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")
}
```

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*), karena tak perlu menuliskan banyak proses berkali-kali, cukup sekali saja dan tinggal panggil jika dibutuhkan.

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

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.

Berikut adalah contoh penerapan fungsi.

```
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, 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 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()`. Fungsi ini berada di dalam package `strings`.



Fungsi Dengan Return Value / Nilai Balik

Sebuah fungsi bisa didesain tidak mengembalikan apa-apa (*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 = randomWithRange(2, 10)
    fmt.Println("random number:", randomValue)
    randomValue = randomWithRange(2, 10)
    fmt.Println("random number:", randomValue)
    randomValue = randomWithRange(2, 10)
    fmt.Println("random number:", randomValue)
}

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

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

```
[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 `randomWithRange`.

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

Sedangkan cara untuk mengembalikan nilainya adalah dengan menggunakan keyword `return` diikuti data yang ingin dikembalikan. Pada contoh di atas, `return value` artinya nilai variabel `value` dijadikan nilai kembalian fungsi.

Eksekusi keyword `return` akan menjadikan proses dalam blok fungsi berhenti pada saat itu juga. Semua statement setelah keyword tersebut tidak akan dieksekusi.

Dari kode di atas mungkin ada beberapa statement yang masih terasa asing, berikut merupakan pembahasannya.

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.

Import Banyak Package

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

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

// atau

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

Pilih cara yang terasa nyaman di hati.

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

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.

```go
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.

Didalamnya 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 $
```

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.

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, yaitu dengan langsung menuliskan tipe-tipe data nilai balik dengan pemisah tanda koma, dan 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 linkgkaran     : 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)
```

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

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`.

Penggunaan Konstanta `math.Pi`

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

Fungsi Variadic

Golang mengadopsi konsep **variadic function** atau pembuatan fungsi dengan parameter sejenis yang tak terbatas. Maksud **tak terbatas** disini adalah jumlah parameter yang disisipkan ketika pemanggilan fungsi bisa berapa saja.

Parameter variadic memiliki sifat yang mirip dengan slice. Nilai parameter-parameter yang disisipkan memiliki tipe data yang sama, dan akan ditampung oleh sebuah variabel saja. Cara pengaksesan tiap datanya juga sama, dengan menggunakan indeks.

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

Penerapan Fungsi Variadic

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

Berikut merupakan contoh penerepannya.

```
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 (`...`) sebelum penulisan tipe data-nya. Menandakan bahwa `numbers` adalah sebuah parameter variadic dengan tipe data `int`.

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

Pada pemanggilan fungsi disisipkan banyak parameter sesuai kebutuhan.

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

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

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

Berikut merupakan penjelasan tambahan mengenai beberapa kode di atas.

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 hasil `fmt.Sprintf()` ditampung oleh variabel `msg`.

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

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 penerepannya 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.

Pengisian Fungsi Variadic Menggunakan Data Slice

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

```
var 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.

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 seperi pada fungsi biasa. Contohnya bisa dilihat pada kode berikut.

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

Jika parameter kedua dan seterusnya ingin diisi dengan data dari slice, maka gunakan tanda 3 titik. Contoh:

```
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 $
```

Fungsi Closure

Definisi termudah **Closure** adalah sebuah fungsi yang bisa disimpan dalam variabel. Dengan menerapkan konsep tersebut, sangat mungkin untuk membuat fungsi didalam fungsi, atau bahkan membuat fungsi yang mengembalikan fungsi.

Closure merupakan *anonymous function* atau fungsi tanpa nama. Biasa dimanfaatkan untuk membungkus suatu proses yang hanya dipakai sekali atau dipakai pada blok tertentu saja.

Closure Disimpan Sebagai Variabel

Sebuah fungsi tanpa nama bisa disimpan dalam variabel. Variabel yang menyimpan closure memiliki sifat seperti fungsi yang disimpannya. Di bawah ini adalah contoh program sederhana untuk mencari nilai terendah dan tertinggi dari suatu array. Logika pencarian dibungkus dalam closure yang ditampung oleh variabel `getMinMax`.

```
package main

import "fmt"

func main() {
    var getMinMax = func(n []int) (int, int) {
        var min, max int
        for i, e := range n {
            switch {
            case i == 0:
                max, min = e, e
            case e > max:
                max = e
            case e < min:
                min = e
            }
        }
        return min, max
    }

    var numbers = []int{2, 3, 4, 3, 4, 2, 3}
    var min, max = getMinMax(numbers)
    fmt.Printf("data : %v\nmin  : %v\nmax  : %v\n", numbers, min, max)
}
```

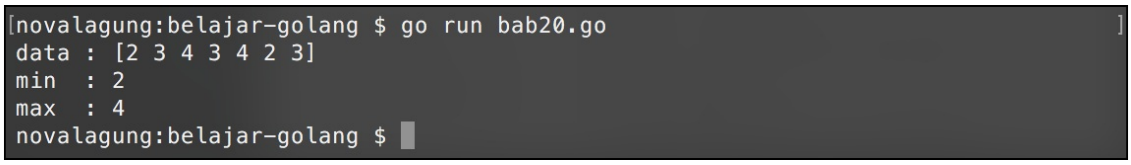
Bisa dilihat pada kode di atas bagiamana cara closure dibuat dan dipanggil. Sedikit berbeda memang dibanding pembuatan fungsi biasa. Fungsi ditulis tanpa nama, lalu ditampung dalam variabel.

```
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:



Berikut adalah penjelasan tambahan mengenai kode di atas

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.

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 kembaliannya** bukan **closure-nya**.

> 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 {`
> `    // ..`
> `}`

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 disiapkan sebuah fungsi bernama `findMax()` yang memiliki salah satu nilai kembalian 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 $
```

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.

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 digunakan untuk filtering data array (yang datanya didapat dari parameter pertama), dengan kondisi filter bisa ditentukan sendiri. Di bawah ini adalah contoh pemanfaatan fungsi tersebut.

```
func main() {
    var data = []string{"wick", "jason", "ethan"}
    var dataContainsO = filter(data, func(each string) bool {
        return strings.Contains(each, "o")
    })
    var dataLenght5 = 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 \"i\"\t:", dataContainsO)
    // filter ada huruf "i" : [wick]

    fmt.Println("filter jumlah huruf \"5\"\t:", dataLenght5)
    // 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"    : [wick]
filter jumlah huruf "5" : [jason ethan]
novalagung:belajar-golang $
```

Pada `dataContainsO`, parameter kedua fungsi `filter()` berisikan statement untuk deteksi apakah terdapat substring `"i"` 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 tau apa manfaatnya, penerapan teknik ini pada kondisi yang tepat akan sangat membantu proses pembuatan aplikasi.

Alias Skema Closure

Seperti yang sudah dipelajari di atas, bahwa closure bisa dimanfaatkan sebagai tipe parameter, contohnya seperti pada fungsi `filter()` di atas. Pada fungsi tersebut kebetulan skema tipe parameter closure-nya tidak terlalu panjang, hanya ada satu buah parameter dan satu buah nilai balik.

Pada skema fungsi yang cukup panjang, akan lebih baik jika digunakan 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()`.

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"`.

Pointer

Pointer adalah referensi atau alamat memory. Variabel pointer berarti variabel yang menampung 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-variabel yang memiliki referensi 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.

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 praktekan. 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 pointer bisa ditampilkan dengan cara variabel tersebut harus di-dereference terlebih dahulu (bisa dilihat pada kode `*numberB`).

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. Contoh:

```
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 $
```

Parameter Pointer

Parameter bisa juga didesain sebagai pointer. Cara penerapannya kurang lebih sama. Tinggal deklarasikan parameter tersebut 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`. 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 $
```

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.

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.

Penerapan Struct

Struct `student` yang sudah disiapkan di atas akan kita manfaatkan untuk mencetak sebuah variabel objek. Property variabel tersebut nantinya diisi 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 pada kode `s1.grade = 2`.

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

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`. Vartiabel 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"}
```

Variabel Objek Pointer

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

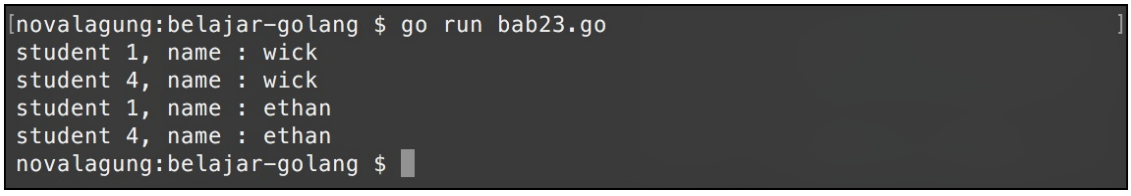
```
var s1 = student{name: "wick", grade: 2}

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`, mengakibatkan 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 keunikan variabel 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"`.



Embedded Struct

Embedded struct adalah penurunan properti dari satu struct ke struct lain, sehingga properti struct yang diturunkan bisa digunakan. 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 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. Contoh `s1.person.age`. Nilai yang dikembalikan memiliki referensi yang sama dengan `s1.age`.

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 dengan 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`.

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`.

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 sebuah property `grade`, dan property lain yang diturunkan dari struct `person`.

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 inisialisasi
var s1 = struct {
    person
    grade int
}{}

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

Inisialisasi Langsung Array Anonymous Struct

Anonymous struct bisa dijadikan sebagai tipe sebuah array. 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)
}
```

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 bagaimana struct dideklrasikan. Maknanya akan berbeda ketika keyword `type` disitu diganti `var`, seperti pada contoh di atas `var student struct`, yang artinya akan dicetak sebuah objek dari anonymous struct dan disimpan pada variabel bernama `student`.

Kelemahan metode ini, nilai tidak bisa diinisialisasi langsung pada saat deklarasi. Contohnya bisa dilihat pada kode di bawah ini.

```
// dekalrasi saja
var student struct {
    grade int
}

// dekalrasi sekaligus inisialisasi
var student = struct {
    grade int
} {
    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 struktur datanya cukup kompleks dengan proses decode hanya sekali.

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 poperty 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, dekalrasi dengan cara ini tidak bisa dilakukan, karena setiap kali save isi file program akan dirapikan. Jadi untuk mengetesnya bisa dengan menggunakan editor lain.

Tag property dalam struct

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

```
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, yaitu ketika sudah masuk bab json.

Method

Method adalah fungsi yang hanya bisa di akses lewat variabel objek. Method merupakan bagian dari `struct`.

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

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 {
```

Method Pointer

Method pointer adalah method yang variabel objeknya dideklarasikan dalam bentuk pointer. Kelebihan method jenis ini adalah manipulasi data pointer pada property milik variabel tersebut bisa dilakukan.

Pemanggilan method pointer sama seperti method biasa. Contohnya bisa dilihat di kode berikut.

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

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

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

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

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

Berikut adalah penjelasan tambahan mengenai fungsi split.

Penggunaan Fungsi `strings.Split()`

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

```
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"`.

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 fungsi.

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.

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.

Package Public & Private

Pengembangan aplikasi dalam *real development* pasti membutuhkan banyak sekali file program. Dan tidak mungkin semuanya di set sebagai package `main`. Dengan pertimbangan tersebut biasanya file-file tersebut dipisah sebagai package baru.

Folder proyek selain berisikan file-file `.go` juga bisa berisikan folder. Subfolder tersebut nantinya akan menjadi package baru. Di Golang, 1 subfolder adalah 1 package (kecuali package main yang berada langsung didalam folder proyek). Menjadikan file yang ada di dalam suatu folder memiliki nama package berbeda dengan di folder lainnya. Bahkan antara folder dan subfolder juga bisa memiliki nama package yang berbeda.

Fungsi, struct, dan variabel yang dibuat di package lain, jika diakses dari package `main` caranya tidak seperti biasanya. Perlu adanya penentuan hak akses yang tepat (apakah public atau private) agar kode tidak kacau balau. Package public, berarti bisa diakses dari package berbeda, sedangkan private berarti hanya bisa di akses dari package yang sama.

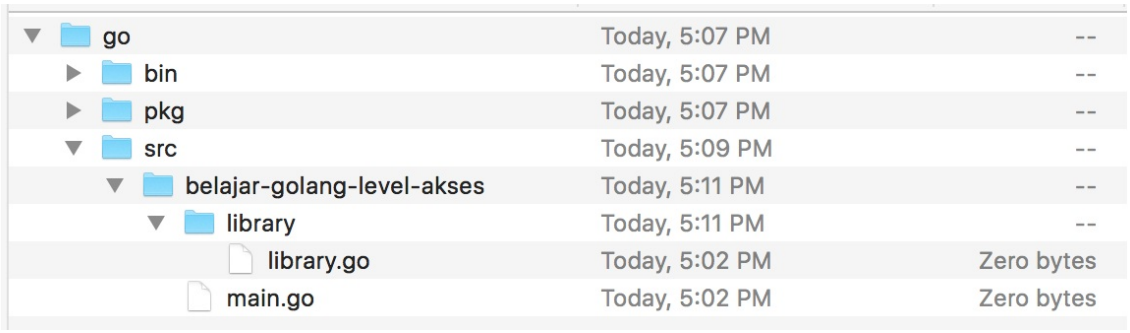
Penentuan level akses atau modifier sendiri di golang sangat mudah, ditandai dengan **character case** nama fungsi/struct/variabel yang ingin di akses. Ketika namanya diawali dengan huruf kapital menandakan bahwa modifier-nya public. Dan sebaliknya, jika diawali huruf kecil, berarti private.

Penggunaan Package, Import, Dan Modifier Public & Private

Agar lebih mudah dipahami, maka langsung saja kita praktekan.

Pertama buat folder proyek baru bernama `belajar-golang-level-akses` dalam folder `$GOPATH/src`, lalu buat file baru bernama `main.go` didalamnya. File ini kita tentukan package-nya sebagai **main**.

Selanjutnya buat folder baru didalam folder yang sudah dibuat dengan nama `library`, isinya sebuah file bernama `library.go`. File ini ditentukan package-nya adalah **library**.



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). Dalam package tersebut terdapat dua fungsi: `SayHello()` dan `introduce()`. Fungsi `SayHello()` adalah publik, bisa diakses dari package lain. Sedang fungsi `introduce()` adalah private, ditandai dengan huruf kecil di huruf pertama nama fungsi.

Selanjutnya akan di-tes apakah memang fungsi yang ber-modifier private tidak bisa diakses dari package lain. Buka file `main.go`, lalu tulis kode berikut.

```
package main

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

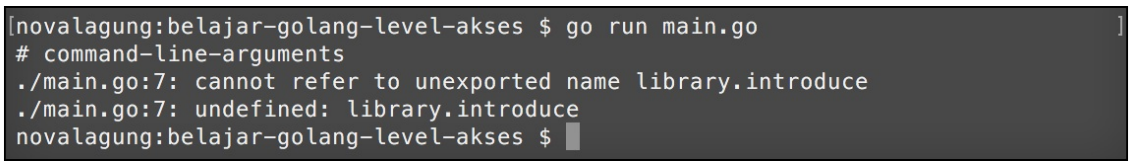
func main() {
    library.SayHello()
    library.introduce("ethan")
}
```

Package yang telah dibuat di-import ke dalam package `main`. Pada saat import, ditulis dengan `"belajar-golang-level-akses/library"` karena lokasi foldernya merupakan subfolder dari proyek `belajar-golang-level-akses`. Dengan ini fungsi-fungsi dalam package tersebut bisa digunakan.

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

Cara pemanggilan fungsi yang berada dalam package lain adalah dengan menulis nama package target diikut dengan nama fungsi menggunakan *dot notation* atau tanda titik.

OK, sekarang coba jalankan kode yang sudah disiapkan di atas. Harusnya akan ada error seperti pada gambar di bawah ini.



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

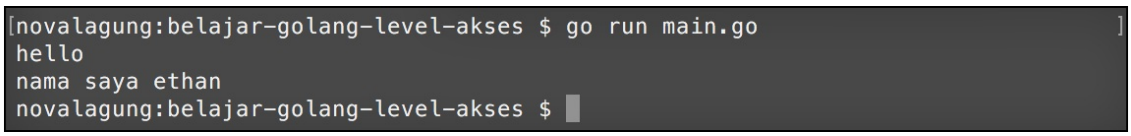
Tambahkan parameter `name` pada fungsi `SayHello()`, lalu panggil fungsi `introduce()` dengan menyisipkan parameter `name` dari dalam fungsi `SayHello()`.

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

Lalu pada main, cukup panggil fungsi `SayHello()` saja, jangan lupa menyisipkan pesan string sebagai parameter-nya.

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

Jika sudah, coba jalankan lagi, harusnya error sudah lenyap.



Penggunaan Public & Private Pada Struct Dan Propertinya

Modifier private & public bisa diterapkan di fungsi, struct, method, maupun property variabel. Dan cara penggunaannya sama seperti pada contoh di atas, yaitu dengan menentukan *case* atau huruf pertama dari nama, apakah huruf tersebut besar atau kecil.

Belajar tentang level akses di Golang akan lebih cepat jika langsung praktek. Oleh karena itu langsung saja kita praktekan. Hapus isi file `library.go`, lalu siapkan struct 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 ketika program dijalankan. 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 {

// pada main.go
var s1 = library.Student{"ethan", 21}
```

```
[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
Grade int

// pada main.go
fmt.Println("grade", s1.Grade)
```

Dari contoh program di atas, bisa disimpulkan bahwa untuk menggunakan `struct` yang berada di package lain, selain nama stuct-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 $
```

Import Dengan 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, package bisa di-import setara dengan file peng-import, caranya dengan menambahkan titik pada saat penulisan keyword `import`. Maksud dari setara 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.

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"
)

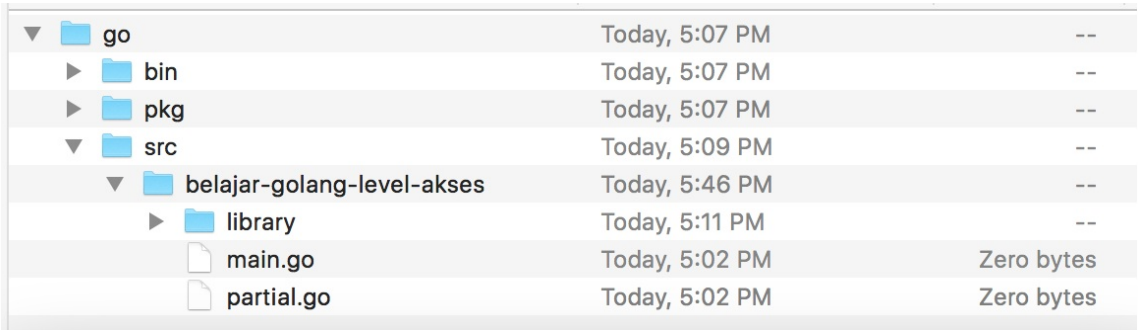
func main() {
    f.Println("Hello World!")
}
```

Disiapkan alias untuk package `fmt` dengan nama `f`. Dengan ini untuk mengakses fungsi `Println()` cukup dengan `f.Println()`.

Mengakses Properti Dalam File Yang Package-nya Sama

Jika properti yang ingin di akses masih dalam satu package tapi file nya saja yg berbeda, 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 praktekan, buat file baru dalam `belajar-golang-level-akses` dengan nama `partial.go`.



File `partial.go` ditentukan packagenya adalah **main** (sama dengan package `main.go`). Selanjutnya tulis kode berikut pada file tersebut.

```
package main

import "fmt"

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

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

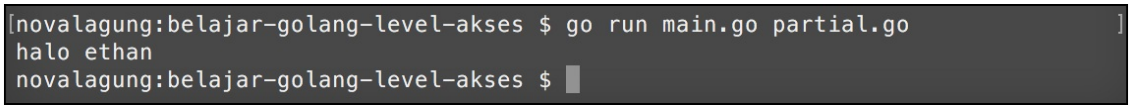
```
package main

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

Sekarang terdapat 2 file berbeda yang package-nya sama-sama `main`, yaitu `main.go` dan `partial.go`. Pada saat **go build** atau **go run**, semua file yang nama package-nya `main` tersebut harus dituliskan sebagai argumen.

```
$ 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.



Fungsi `init()`

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

Langsung saja kita praktekkan. Buka file `library.go`, 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.

Perlu diketahui bahwa 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 $
```

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.

Penerapan Interface

Yang pertama perlu dilakukan untuk menerapkan interface adalah menyiapkan interface baru beserta definisi method nya. Keyword `type` dan `interface` digunakan dalam pembuatannya. Contoh:

```
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 variabel, dimana variabel tersebut digunakan untuk menampung objek bangun datar hasil dari struct yang akan kita buat.

Dengan memanfaatkan variabel objek hasil interface `hitung`, perhitungan luas dan keliling suatu bangun datar bisa dilakukan, tanpa perlu tahu jenis bangun datarnya sendiri itu apa.

Selanjutnya disiapkan struct bangun datar `lingkaran` yang 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()`.

Setelah itu, siapkan juga 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()`.

Terakhir, buat implementasinya 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. Disiapkan variabel objek `bangunDatar` yang tipe-nya interface `hitung`. Variabel tersebut akan digunakan untuk menampung objek konkrit 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 konkritnya (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 konkrit 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.

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.

Selanjutnya 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`.

Selanjutnya, 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 $
```

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**).

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 belakang-nya (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-instansiasi, 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.

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**.

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 $
```

Reflect

Reflection adalah teknik untuk inspeksi sebuah 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

Mencari Tipe Data & Value Menggunakan Reflect

Dengan reflection, tipe data dan nilai dari suatu 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 yang bisa dimanfaatkan. Salah satunya adalah `Type()`. Method ini mengembalikan tipe data variabel yang bersangkutan dalam bentuk `string`.

Statement `reflectValue.Int()` akan menghasilkan nilai `int` dari variabel tersebut. 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()`, karena jika dipaksa, akan muncul error.

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, menggunakan `.Interface()` akan mempersingkat waktu. Leweat method tersebut segala jenis nilai akan 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)
```

Pengaksesan Informasi Property Variabel Objek

Reflect bisa digunakan untuk mengambil informasi semua property variabel objek cetakan struct, dengan catatan property-property tersebut bermodifier public. Agar lebih mudah dipahami, langsung saja kita praktekan.

Siapkan sebuah struct bernama `student`.

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

Setelah itu, 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
```

Didalam method `getPropertyInfo` terjadi beberapa hal. Pertama objek `reflect.Value` dari variabel `s` diambil. Setelah itu dilakukan pengecekkan apakah variabel objek tersebut merupakan pointer atau tidak (bisa dilihat dari `if reflectValue.Kind() == reflect.Ptr`). jika iya, maka perlu dicari 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()`.

Pengaksesan Informasi Method Variabel Objek

Method juga bias diakses lewat reflect, syaratnya masih sama seperti pada pengaksesan proprerty, yaitu harus bermodifier public.

Langsung saja kita praktekkan. 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-nya (`SetName`) juga diambil. Pengambilan refleksi method bisa lewat nama method-nya (menggunakan `MethodByName`) atau lewat indeks method-nya (menggunakan `Method(i)`).

Setelah refleksi method yang dicari sudah didapatkan, `Call()` dipanggil untuk pengeksekusian 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"),
}
```

Goroutine

Goroutine bukanlah thread. 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. Ekseksui goroutine bersifat *asynchronous*, menjadikannya tidak saling tunggu dengan goroutine lain.

Penerapan Goroutine

Untuk menerapkan goroutine, proses yang akan dieksekusi sebagai goroutine harus dibungkus kedalam fungsi. Pada saat pemanggilan fungsi tersebut, ditambahkan keyword `go` didepannya. Dengan demikian proses yang ada dalam fungsi tersebut dideteksi sebagai goroutine baru.

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, `runtime.GOMAXPROCS(n)` digunakan untuk menentukan jumlah prosesor yang aktif.

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 asnychronous. 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.

Penggunaan Fungsi `runtime.GOMAXPROCS()`

Fungsi ini digunakan untuk menentukan jumlah prosesor yang digunakan dalam eksekusi program.

Jumlah prosesor yang diinputkan secara otomatis akan disesuaikan dengan jumlah asli *logical processor* yang ada. Jika angka yg diinputkan adalah lebih, maka dianggap menggunakan semua prosesor yang ada.

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)

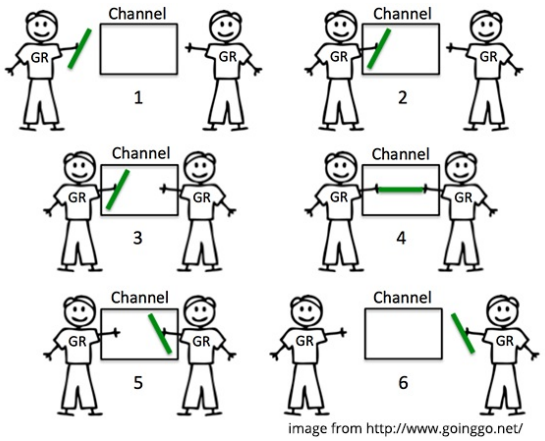
// user inputs: "trafalgar d law"

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.

Channel

Channel digunakan untuk menghubungkan gorutine satu dengan goroutine lainnya. Mekanismenya adalah dengan cara 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.

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 baru dieksekusi, yang 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 kemudian 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 paling awal mengirim data, akan diterima datanya oleh 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 dieksekusi terlebih dahulu. Goroutine yang dieksekusi 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 selesai lebih dulu.

Channel Sebagai Tipe Data Parameter

Variabel channel bisa di-passing ke fungsi lain sebagai parameter. Caranya dengan menambahkan keyword `chan` ketika deklarasinya.

Langsung saja kita praktekan. 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.

Iterasi Data Array Langsung Pada Saat Inisialisasi

Data array yang baru di-inisialisasi bisa langsung di-iterasi, caranya mudah dengan menuliskanya langsung setelah keyword `range` .

```
for _, each := range []string{"wick", "hunt", "bourne"} {
    // ...
}
```

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")

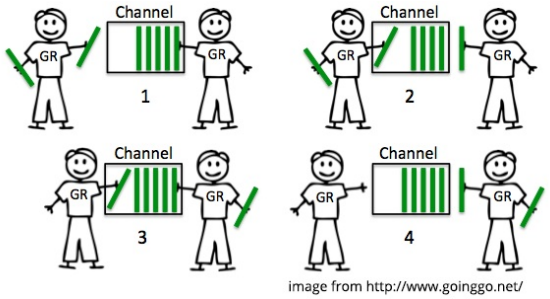
var message = <-messages
fmt.Println(message)
```

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 akan menjadi penentu kapan kita bisa mengirimkan data. 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.



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-nya. 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.

Channel - Select

Adanya channel memang sangat membantu pengontrolan goroutine, jumlah goroutine yang banyak bukan lagi masalah.

Ada kalanya dimana kita butuh tak hanya satu channel saja untuk manage goroutine yang 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`.

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 dikontrol 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 akan dieksekusi di dalam `main` sebagai goroutine baru. Di akhir masing-masing fungsi akan dikirimkan data hasil komputasi ke channel yang sudah ditentukan (`ch1` menampung data rata-rata, `ch2` untuk data nilai tertinggi).

Setelah itu, 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?

Channel - Range dan Close

Penerimaan data lewat channel yang dipakai oleh banyak goroutine, akan lebih mudah dengan memanfaatkan keyword `for` - `range`.

`for` - `range` jika diterapkan pada channel, akan melakukan perulangan tanpa henti. Perulangan tersebut tetap berjalan meski tidak ada transaksi pada channel, dan hanya akan berhenti jika status channel berubah menjadi **closed** atau sudah ditutup. Fungsi `close` digunakan utuk menutup channel.

Channel yang sudah ditutup tidak bisa digunakan lagi untuk menerima maupun mengirim data. Menjadikan penerimaan data menggunakan `for` - `range` juga ikut berhenti.

Penerapan `for` - `range` - `close` Pada Channel

Berikut adalah contoh program yang menggunakan `for` - `range` untuk pengambilan data dari channel.

Pertama siapkan fungsi `sendMessage()` untuk handle pengiriman data. Didalam fungsi ini akan dijalankan perulangan sebanyak 20 kali, ditiap 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.

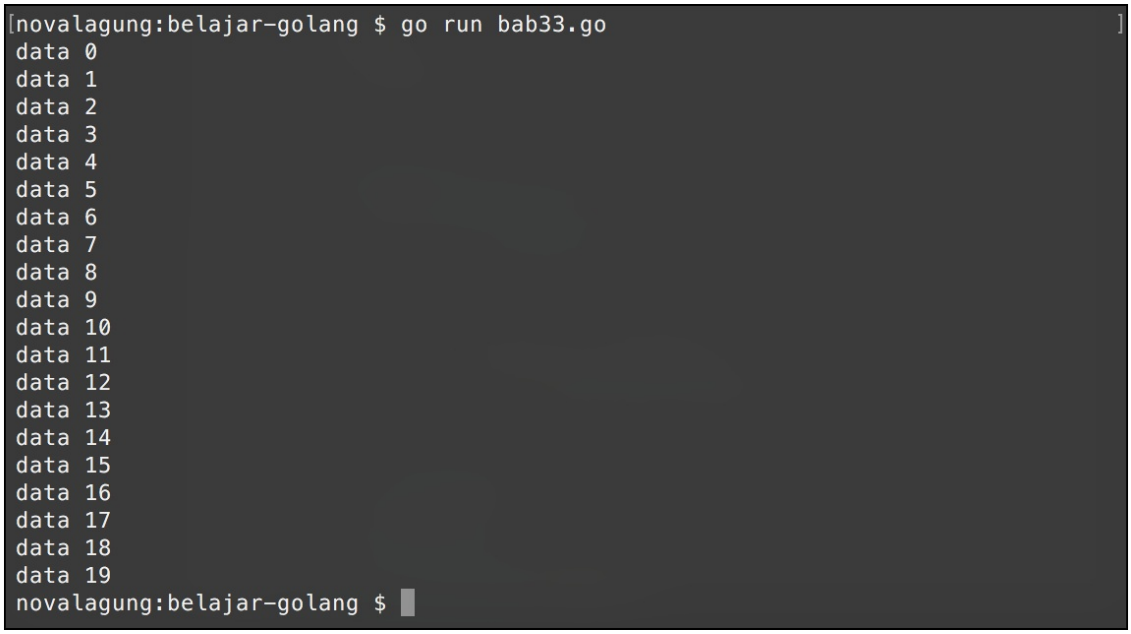
```
func printMessage(ch <-chan string) {
    for message := range ch {
        fmt.Println(message)
    }
}
```

Bat channel baru di fungsi `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` akan dimatikan (`close(ch)`). Membuat perulangan data channel dalam `printMessage()` juga akan berhenti.



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 `ch` bisa digunakan untuk **mengirim** dan **menerima** data
`ch chan<- string`	Parameter `ch` hanya bisa digunakan untuk **mengirim** data
`ch <-chan string`	Parameter `ch` 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.

Channel - Timeout

Timeout digunakan untuk mengontrol penerimaan data dari channel berdasarkan waktu diterimanya, dengan durasi timeout bisa ditentukan sendiri.

Ketika tidak ada aktivitas penerimaan data selama durasi tersebut, akan memicu callback yang isinya juga ditentukan sendiri.

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 retreiveData(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.

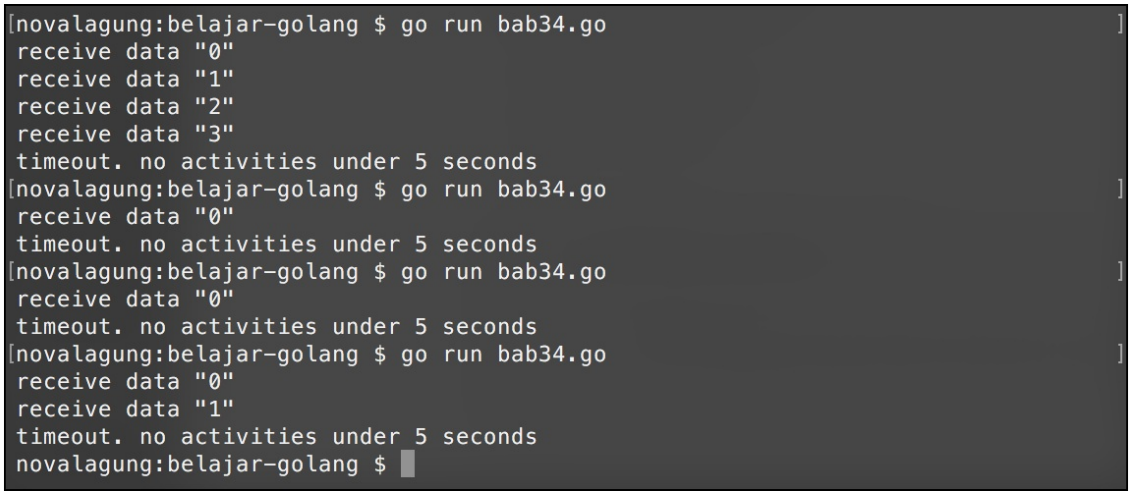
Terakhir, kedua fungsi tersebut dipanggil di `main` .

```
func main() {
    rand.Seed(time.Now().Unix())
    runtime.GOMAXPROCS(2)

    var messages = make(chan int)

    go sendData(messages)
    retreiveData(messages)
}
```

Akan muncul output setiap kali ada penerimaan data dengan delay waktu acak. Ketika tidak ada aktifitas pada channel dalam durasi 5 detik, perulangan pengecekkan channel akan dihentikan.



Defer & Exit

Defer digunakan untuk mengakhirkan eksekusi sebuah statement. Sedangkan **Exit** digunakan untuk menghentikan program. 2 topik ini sengaja digabung agar hubungan antara keduanya lebih mudah dipahami.

Penerapan keyword `defer`

Seperti yang sudah dijelaskan secara singkat di atas, bahwa defer digunakan untuk mengakhirkan eksekusi baris kode. Ketika eksekusi sudah sampai pada akhir blok fungsi, 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")
}
```

Keyword `defer` digunakan untuk mengakhirkan statement. Pada kode di atas, `fmt.Println("halo")` di-defer, hasilnya string `"halo"` akan muncul setelah `"selamat datang"`.

```
novalagung:belajar-golang $ go run bab35.go
selamat datang
halo
novalagung:belajar-golang $
```

Ketika ada banyak statement yang di-defer, maka statement tersebut akan dieksekusi di akhir secara berurutan.

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 $
```

Error & Panic

Error merupakan topik yang penting dalam pemrograman golang. Di bagian ini kita akan belajar mengenai pemanfaatan error dan cara membuat custom error sendiri.

Kita juga akan belajar tentang penggunaan **panic** untuk menampilkan pesan error.

Pemanfaatan Error

`error` merupakan sebuah tipe. Error memiliki beberapa property yang menampung informasi yang berhubungan dengan error yang bersangkutan.

Di golang, 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, kita bisa langsung tau penyebab error muncul dengan memanfaatkan nilai balik kedua.

Berikut merupakan contoh program sederhana untuk deteksi inputan dari user, apakah numerik atau bukan.

```
package main

import (
    "fmt"
    "strconv"
)

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

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

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

Ketika program dijalankan, akan muncul tulisan `"Type some number: "` . Ketik sebuah angka lalu enter.

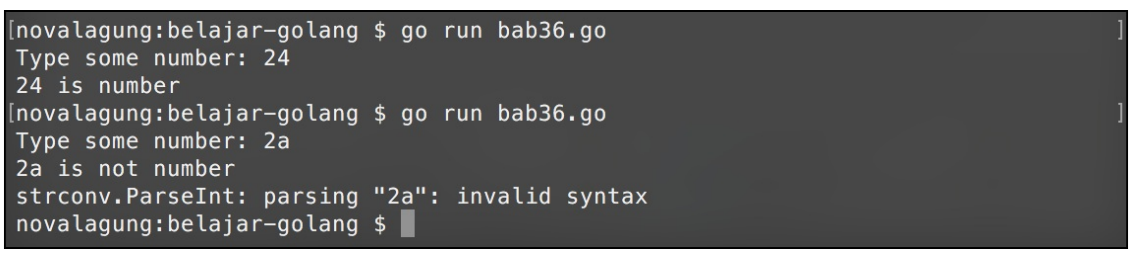
`fmt.Scanln(&input)` akan mengambil inputan yang diketik user sebelum dia menekan enter, lalu menyimpannya sebagai string ke variabel `input` .

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

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

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

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



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).

Berikut merupakan contoh pembuatan 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 program 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 $
```

Penggunaan Keyword `panic`

Panic digunakan untuk menampilkan *trace* error. Hasil keluarannya sama seperti `fmt.Println()` hanya saja informasi yang ditampilkan lebih detail.

Pada program yang telah kita buat tadi, ubah `fmt.Println()` yang berada di dalam blok kondisi `else` pada fungsi main menjadi `panic()`.

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

Ketika user menginputkan string kosong, maka error akan dimunculkan menggunakan fungsi `panic`.

```
[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 $
```

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 pada 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. Sampel data yang digunakan sebagai contoh adalah kode berikut.

```
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"},
}
```

Layout Format `%b`

Digunakan untuk memformat data numerik, menjadi bentuk string numerik berbasis 2 (biner).

```
fmt.Printf("%b\n", data.age)
// 11010
```

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
```

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
```

Layout Format `%e` atau `%E`

Digunakan untuk memformat data numerik desimal ke dalam bentuk notasi 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 x 10^2**, dan hasil operasi tersebut adalah sesuai dengan data asli = **182.5**.

Perbedaan antara `%e` dan `%E` hanya huruf besar kecil karakter `e` pada hasil.

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
```

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
```

Layout Format `%o`

Digunakan untuk memformat data numerik, menjadi bentuk string numerik berbasis 8 (oktal).

```
fmt.Printf("%o\n", data.age)
// 32
```

Layout Format `%p`

Digunakan untuk memformat data pointer, mengembalikan alamat pointer referensi variabel-nya.

Alamat pointer dituliskan dalam bentuk numerik berbasis 16 dengan prefix `0x`.

```
fmt.Printf("%p\n", &data.name)
// 0x2081be0c0
```

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 \""
```

Layout Format `%s`

Digunakan untuk memformat data string.

```
fmt.Printf("%s\n", data.name)
// wick
```

Layout Format `%t`

Digunakan untuk memformat data boolean, menampilkan nilai `bool`-nya.

```
fmt.Printf("%t\n", data.isGraduated)
// false
```

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
```

Layout Format `%v`

Digunakan untuk memformat data apa saja (termasuk data bertipe `interface{}`). Hasil kembaliannya 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]}
```

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]}
```

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.

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.

Layout Format `%%`

Cara untuk menulis karakter `%` pada string format.

```
fmt.Printf("%%\n")
// %
```

Time, Parsing Time, & Format Time

Pada bab ini kita akan belajar tentang pemanfaatan time, method-method yang disediakan, dan juga **format** & **parsing** data `string` ke `time.Time` dan sebaliknya.

Golang menyediakan package `time` yang berisikan banyak sekali komponen yang bisa digunakan untuk keperluan pemanfaatan waktu.

> Time disini maksudnya adalah gabungan **date** dan **time**, bukan hanya waktu saja.

Penggunaan `time.Time`

`time.Time` adalah tipe data untuk objek waktu. Ada 2 cara yang bisa digunakan untuk membuat data bertipe ini.

- Dengan menjadikan waktu sekarang sebagai time
- Membuat time dengan data 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 data adalah waktu sekarang. 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 $
```

Sedangkan `time.Date()` digunakan untuk membuat objek `time.Time` baru dengan informasi waktu yang bisa 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 kita.

Method Milik `time.Time`

```
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 tersedia pada tipe `time.Time`.

Method	Return Type	Penjelasan
`now.Year()`	`int`	Tahun
`now.YearDay()`	`int`	Hari ke-? di mulai awal tahun
`now.Month()`	`int`	Bulan
`now.Weekday()`	`string`	Nama hari. Bisa menggunakan `now.Weekday().String()` untuk mengambil bentuk string-nya
`now.ISOWeek()`	(`int` , `int`)	Tahun dan minggu ke-? mulai awal tahun
`now.Day()`	`int`	Tanggal
`now.Hour()`	`int`	Jam
`now.Minute()`	`int`	Menit
`now.Second()`	`int`	Detik
`now.Nanosecond()`	`int`	Nano detik
`now.Local()`	`time.Time`	Waktu dalam timezone lokal
`now.Location()`	`*time.Location`	Mengambil informasi lokasi, apakah *local* atau *utc*. Bisa menggunakan `now.Location().String()` untuk mengambil bentuk string-nya
`now.Zone()`	(`string` , `int`)	Mengembalikan informasi *timezone offset* dalam string dan numerik. Sebagai contoh `WIB, 25200`
`now.IsZero()`	`bool`	Deteksi apakah nilai object `now` adalah `01 Januari tahun 1, 00:00:00 UTC` . Jika iya maka bernilai `true`
`now.UTC()`	`time.Time`	Waktu dalam timezone `UTC`
`now.Unix()`	`int64`	Waktu dalam format *unix time*
`now.UnixNano()`	`int64`	Waktu dalam format *unix time*. Infomasi nano detik juga dimasukkan
`now.String()`	`string`	Waktu dalam string

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
```



Layout format time di golang berbeda dibanding bahasa lain. Umumnya layout format yang digunakan adalah seperti `"DD/MM/YYYY"` dll, 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`

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 bisa dimanfaatkan. Berikut adalah list nya.

Predefined Layout Format	Layout Format
`time.ANSIC`	Mon Jan _2 15:04:05 2006
`time.UnixDate`	Mon Jan _2 15:04:05 MST 2006
`time.RubyDate`	Mon Jan 02 15:04:05 -0700 2006
`time.RFC822`	02 Jan 06 15:04 MST
`time.RFC822Z`	02 Jan 06 15:04 -0700
`time.RFC850`	Monday, 02-Jan-06 15:04:05 MST
`time.RFC1123`	Mon, 02 Jan 2006 15:04:05 MST
`time.RFC1123Z`	Mon, 02 Jan 2006 15:04:05 -0700
`time.RFC3339`	2006-01-02T15:04:05Z07:00
`time.RFC3339Nano`	2006-01-02T15:04:05.999999999Z07:00
`time.Kitchen`	3:04PM
`time.Stamp`	Jan _2 15:04:05
`time.StampMilli`	Jan _2 15:04:05.000
`time.StampMicro`	Jan _2 15:04:05.000000
`time.StampNano`	Jan _2 15:04:05.000000000

Format `time.Time`

Setelah sebelumnya kita belajar tentang konversi `string` ke `time.Time`. Kali ini kita akan belajar sebaliknya, yaitu konversi `time.Time` ke `string`.

Method `Format()` digunakan untuk membentuk `string` menggunakan layout format tertentu. Layout format yang bisa digunakan adalah sama seperti pada parsing time. 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` berisikan hasil parsing menggunakan predefined layout format `time.RFC822`. Variabel tersebut kemudian ditampilkan 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 $
```

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.

Untuk mendeteksi hal ini bisa memanfaatkan 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 harusnya 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 $
```

Timer

Ada beberapa fungsi dalam package `time` yang memiliki kegunaan sebagai timer. Dengan memanfaatkan fungsi-fungsi tersebut, kita bisa menunda eksekusi sebuah proses dengan durasi waktu tertentu.

Fungsi `time.Sleep()`

Fungsi ini digunakan untuk menghentikan program sejenak. `time.Sleep()` bersifat **blocking**, sehingga statement dibawahnya tidak akan dieksekusi sampai waktu pemberhentian usai. Contoh sederhana penerapannya 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")
}
```

Tulisan `"start"` muncul, lalu 4 detik kemudian tulisan `"after 4 seconds"` muncul.

Fungsi `time.NewTimer()`

Fungsi ini sedikit berbeda dengan `time.Sleep()`. Fungsi `time.NewTimer()` mengembalikan sebuah objek `*time.Timer` yang memiliki method `C`. Method ini mengembalikan sebuah channel dan akan dieksekusi dalam waktu yang sudah ditentukan. Contoh penerapannya bisa dilihat pada kode berikut.

```
var timer = time.NewTimer(4 * time.Second)
fmt.Println("start")
<-timer.C
fmt.Println("finish")
```

Tulisan `"finish"` akan muncul setelah delay **4 detik**.

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")
```

Tulisan `"start"` akan muncul di awal. Diikuti 4 detik kemudian tulisan `"expired"`.

Didalam callback terdapat proses transfer data lewat channel, mengakibatkan 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.

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.

Kombinasi Timer & Goroutine

Berikut merupakan contoh penerapan timer dan goroutine. Program di bawah ini adalah program tanya-jawab sederhana. User harus menginputkan jawaban dalam waktu tidak lebih dari 5 detik. Jika lebih dari waktu tersebut belum ada jawabamn, maka akan muncul pesan *time out*.

Pertama, import package yang diperlukan.

```
package main

import "fmt"
import "os"
import "time"
```

Buat fungsi `timer()`, yang nantinya akan dieksekusi sebagai goroutine. Di dalam fungsi ini akan ada proses pengiriman data lewat channel `ch` ketika waktu sudah mencapai `timeout`.

```
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 ketika sebuah data diterima dari channel `ch` maka akan ditampilkan tulisan penanda waktu 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`.

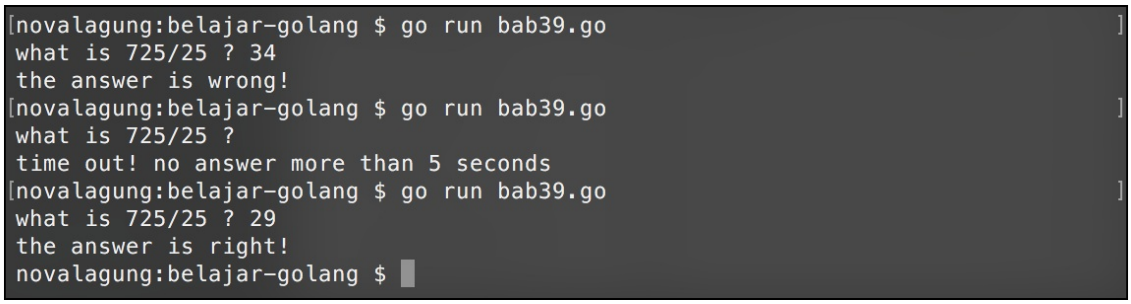
```
func main() {
    var timeout = 5
    var ch = make(chan bool)

    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.



Konversi 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.

Konversi Menggunakan `strconv`

`strconv` berisikan banyak fungsi yang sangat membantu untuk keperluan konversi data. Berikut merupakan beberapa fungsi dalam package tersebut yang bisa dimanfaatkan.

Fungsi `strconv.Atoi()`

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

```
package main

import "fmt"
import "strconv"

func main() {
    var str = "124"
    var num, err = strconv.Atoi(str)

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

Fungsi `strconv.Itoa()`

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

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

fmt.Println(str) // "124"
```

Fungsi `strconv.ParseInt()`

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

Pada contoh berikut, string `"124"` ditentukan basis numeriknya 10, akan dikonversi ke jenis tipe data `int64`.

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

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

Contoh lainnya, string `"1010"` ditentukan basis numeriknya 2 (biner), akan dikonversi ke jenis tipe data `int8`.

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

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

Fungsi `strconv.FormatInt()`

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

```
var num = int64(24)
var str = strconv.FormatInt(num, 8)

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

Fungsi `strconv.ParseFloat()`

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

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

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

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

Fungsi `strconv.FormatFloat()`

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

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

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

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

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.dddde±dd, a, eksponen desimal (basis 10)
`E`	-d.ddddE±dd, a, eksponen desimal (basis 10)
`f`	-ddd.dddd, tanpa eksponen
`g`	Akan menggunakan format eksponen `e` untuk eksponen besar dan `f` untuk selainnya
`G`	Akan menggunakan format eksponen `E` untuk eksponen besar dan `f` untuk selainnya

Fungsi `strconv.ParseBool()`

Digunakan untuk konversi `string` ke `bool`.

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

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

Fungsi `strconv.FormatBool()`

Digunakan untuk konversi `bool` ke `string`.

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

fmt.Println(str) // 124
```

Konversi Data Menggunakan Casting

Keyword tipe data bisa digunakan untuk casting. Cara penggunaannya adalah dengan memanggilnya sebagai fungsi dan menyisipkan data yang akan dikonversi sebagai parameter.

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

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

Casting `string` ↔ `byte`

String sebenarnya adalah slice/array `byte`. Di Golang sebuah karakter biasa (bukan unicode) direpresentasikan oleh sebuah elemen slice byte. Nilai slice tersebut adalah data `int` yang (default-nya) ber-basis desimal, yang merupakan kode ASCII dari karakter biasa tersebut.

Cara mendapatkan slice byte dari sebuah data string adalah dengan meng-casting-nya ke tipe `[]byte`. Tiap elemen `byte` isinya adalah data numerik dengan basis desimal.

```
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 selanjutnya dibawah ini merupakan kebalikan dari contoh di atas, sebuah `[]byte` akan dicari bentuk `string` -nya.

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

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

Beberapa kode byte string saya tuliskan sebagai dalam sebuah slice, yang ditampung oleh 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 byte dari karakter yang akan dihasilkan.

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

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

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).

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

```
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 `interface{}` ke tipe `type`. Namun casting ini hanya bisa dilakukan pada `switch`.

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

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

Fungsi String

Golang menyediakan package `strings` yang didalamnya berisikan cukup banyak fungsi untuk keperluan pengolahan data bertipe string. Bab ini merupakan pembahasan mengenai beberapa fungsi yang ada di dalam package tersebut.

Fungsi `strings.Contains()`

Dipakai untuk deteksi apakah string (parameter kedua) merupakan bagian dari string lain (parameter pertama). Nilai kembaliannya 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"`.

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
```

Fungsi `strings.HasSuffix()`

Digunakan untuk deteksi apakah sebuah string (parameter pertama) diakhiri string tertentu (parameter kedua).

```
var isSuffix1 = strings.HasSuffix("john wick", "ic")
fmt.Println(isSuffix1) // false

var isSuffix2 = strings.HasSuffix("john wick", "ck")
fmt.Println(isSuffix2) // true
```

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"`.

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`.

Fungsi `strings.Replace()`

Fungsi ini digunakan untuk replace/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"
```

Pada contoh di atas, substring `"a"` pada string `"banana"` akan di-replace dengan string `"o"`.

Pada `newText1`, hanya 1 huruf `o` saja yang tereplace karena maksimal substring yang ingin di-replace ditentukan 1.

Angka `-1` akan menjadikan proses replace berlaku pada semua substring. Contoh bisa dilihat pada `newText3`.

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"`

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 menggunakan pemisah string spasi `" "`, hasilnya kemudian ditampung oleh `string1`.

Untuk memisah string menjadi array tiap 1 string, gunakan pemisah string kosong `""`. Bisa dilihat contohnya pada variabel `string2`.

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* (`-`).

Fungsi `strings.ToLower()`

Mengubah huruf-huruf string menjadi huruf kecil.

```
var str = strings.ToLower("aLAy")
fmt.Println(str) // "alay"
```

Fungsi `strings.ToUpper()`

Mengubah huruf-huruf string menjadi huruf besar.

```
var str = strings.ToUpper("eat!")
fmt.Println(str) // "EAT!"
```

Regexp

Regexp atau **regular expression** adalah suatu teknik yang digunakan untuk pencocokan string dengan pola tertentu. Regexp 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`.

Penerapan Regexp

Fungsi `regexp.Compile()` digunakan untuk mengkompilasi ekspresi regex yang dimasukkan. 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` yang tipenya adalah `regexp.*Regexp`.

Struct `regexp.Regexp` memiliki banyak method, salah satunya adalah `FindAllString()`, yang berfungsi untuk pencarian 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.

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]+`.

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.

Method `FindStringIndex()`

Digunakan untuk mencari index string kembalian hasil dari operasi regexp.

```
var text = "banana burger soup"
var regex, _ = regexp.Compile(`[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.

Method `FindAllString()`

Digunakan untuk mencari banyak string yang memenuhi kriteria regexp yang telah ditentukan.

```
var text = "banana burger soup"
var regex, _ = regexp.Compile(`[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.

Method `ReplaceAllString()`

Berguna untuk me-replace semua string yang memenuhi kriteri regexp, dengan string lain.

```
var text = "banana burger soup"
var regex, _ = regexp.Compile(`[a-z]+`)

var str = regex.ReplaceAllString(text, "potato")
fmt.Println(str)
// "potato potato potato"
```

Method `ReplaceAllStringFunc()`

Digunakan untuk me-replace semua string yang memenuhi kriteri regexp, dengan kondisi yang bisa ditentukan untuk setiap substring yang akan di replace.

```
var text = "banana burger soup"
var regex, _ = regexp.Compile(`[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.

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 diisi `-1` maka akan me-replace semua karakter.

```
var text = "banana,burger,soup"
var regex, _ = regexp.Compile(`[a-z]+`)

var str = regex.Split(text, -1)
fmt.Printf("%#v \n", str)
// ["", ",", ",", ""]
```

Encode - Decode Base64

Golang memiliki package `encoding/base64`, yang 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.

Penerapan Fungsi `EncodeToString()` & `DecodeString()`

Fungsi `EncodeToString()` dan `DecodeString()` digunakan untuk encode dan decode data dari bentuk `string` ke `[]byte` atau sebaliknya. 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 berupa `string`.

Sedangkan pada fungsi decode `base64.StdEncoding.DecodeString()`, data `string` yang di-decode akan dikembalikan dalam bentuk `[]byte`. Ekspresi `string(decodedByte)` menjadikan data `[]byte` tadi berubah menjadi string.

```
novalagung:belajar-golang $ go run bab43.go
encoded: am9obiB3aWNr
decoded: john wick
novalagung:belajar-golang $
```

Penerapan Fungsi `Encode()` & `Decode()`

Kedua fungsi ini digunakan untuk decode dan encode data dari `[]byte` ke `[]byte`. Penggunaan cara ini cukup panjang karena variabel penyimpan hasil encode maupun decode harus disiapkan terlebih dahulu dengan ketentuan memiliki lebar elemen 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, karena tipe datanya sudah dalam bentuk `[]byte`.

Encode & Decode Data URL

Khusus untuk data string yang bentuknya URL, akan lebih efektif menggunakan `URLEncoding` dibanding `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 = "http://depeloper.com/"

var encodedString = base64.URLEncoding.EncodeToString([]byte(data))
fmt.Println(encodedString)

var decodedByte, _ = base64.URLEncoding.DecodeString(encodedString)
var decodedString = string(decodedByte)
fmt.Println(decodedString)
```

Hash Sha1

Hash merupakan sebuah algoritma enkripsi untuk mengubah text menjadi deretan karakter acak. Jumlah karakter hasil hash selalu sama. Hash termasuk *one-way encription*, 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.

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` yang memiliki 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 $ ]
```

Metode Salting Pada Hash

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, menjadikan 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 praktekan. 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. Selain hasil hash, data salt juga harus disimpan pada database, karena digunakan dalam pencocokan password setiap user melakukan login.

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.

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/sdbvcqxd0pq3jz14pwyf_rz00000gn/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 nya ( ), maka harus diapit tanda petik (`"`), agar tidak dideteksi sebagai 2 argumen.

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:

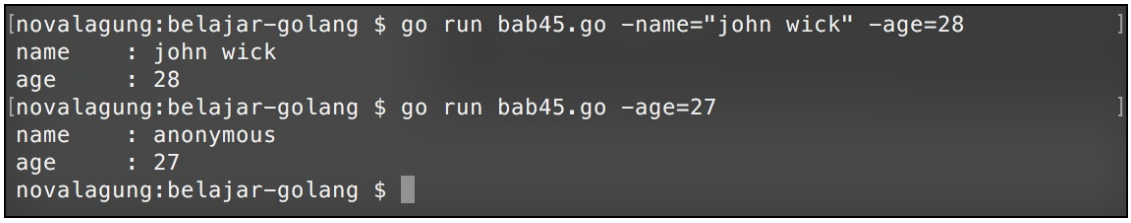
```
$ go run bab45.go -name="john wick" -age=28
```

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`).



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
`flag.Bool(name, defaultValue, usage)`	`*bool`
`flag.Duration(name, defaultValue, usage)`	`*time.Duration`
`flag.Float64(name, defaultValue, usage)`	`*float64`
`flag.Int(name, defaultValue, usage)`	`*int`
`flag.Int64(name, defaultValue, usage)`	`*int64`
`flag.String(name, defaultValue, usage)`	`*string`
`flag.Uint(name, defaultValue, usage)`	`*uint`
`flag.Uint64(name, defaultValue, usage)`	`*uint64`

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.

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 Wind*ws).

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()`. Contoh:

```
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
```

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`.

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 ditimpa. 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 checkError(err error) {
    if err != nil {
        fmt.Println(err.Error())
        os.Exit(0)
    }
}

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)
        checkError(err)
        defer file.Close()
    }
}

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

Fungsi `os.Stat()` mengembalikan 2 data, yaitu informasi tetang 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. Perlu diketahui bahwa file yang baru terbuat statusnya adalah otomatis **open**, jadi perlu untuk di-**close** menggunakan method `file.Close()` setelah file tidak digunakan lagi. Membiarkan file terbuka ketika sudah tak lagi digunakan bukan hal yang baik, menyebabkan alokasi memory menjadi sia-sia.

```
[novalagung:belajar-golang $ ls /Users/novalagung/Documents/temp                ]
[novalagung:belajar-golang $ go run bab47.go                                    ]
[novalagung:belajar-golang $ ls /Users/novalagung/Documents/temp                ]
 test.txt
 novalagung:belajar-golang $
```

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)
    checkError(err)
    defer file.Close()

    // tulis data ke file
    _, err = file.WriteString("halo\n")
    checkError(err)
    _, err = file.WriteString("mari belajar golang\n")
    checkError(err)

    // simpan perubahan
    err = file.Sync()
    checkError(err)
}

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:belajar-golang $ cat /Users/novalagung/Documents/temp/test.txt       ]
[novalagung:belajar-golang $ go run bab47.go                                     ]
[novalagung:belajar-golang $ cat /Users/novalagung/Documents/temp/test.txt       ]
 halo
 mari belajar golang
 novalagung:belajar-golang $
```

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)
    checkError(err)
    defer file.Close()

    // baca file
    var text = make([]byte, 1024)
    for {
        n, err := file.Read(text)
        if err != io.EOF {
            checkError(err)
        }
        if n == 0 {
            break
        }
    }
    fmt.Println(string(text))
    checkError(err)
}

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

`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

Kemudian disiapkan variabel `text` yang bertipe slice `[]byte` dengan alokasi elemen 1024. Variabel tersebut akan berisikan data hasil fungsi `file.Read()` . Proses pembacaan file akan dilakukan terus menerus, berurutan dari baris pertama hingga akhir.

Error yang muncul ketika eksekusi `file.Read()` akan difilter, 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 file atau **end of file**.

```
[novalagung:belajar-golang $ go run bab47.go
halo
mari belajar golang
```

Menghapus File

Cara menghapus file sangatlah mudah, cukup panggil fungsi `os.Remove()` , masukkan path file yang ingin dihapus sebagai parameter.

```
func deleteFile() {
    var err = os.Remove(path)
    checkError(err)
}

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

```
[novalagung:belajar-golang $ ls /Users/novalagung/Documents/temp
test.txt
[novalagung:belajar-golang $ go run bab47.go
[novalagung:belajar-golang $ ls /Users/novalagung/Documents/temp
novalagung:belajar-golang $
```

Web

Golang menyediakan package `net/http` yang berisikan berbagai macam fitur untuk keperluan pembuatan aplikasi berbasi web. Termasuk didalamnya routing, server, templating, dan lainnya, semua tinggal pakai.

Golang memiliki web server sendiri, dan web server tersebut berada di dalam golang, tdak seperti bahasa lain yang server nya terpisah dan perlu di-instal sendiri (seperti PHP yang memerlukan Apache, .NET yang memerlukan IIS). Tapi tak menutup kemungkinan juga sebuah aplikasi Golang dijalankan pada web server lain seperti server Nginx.

Di bab ini kita akan belajar cara pembuatan aplikasi web sederhanda, dan juga pemanfaatan template dalam mendesain view.

Membuat Aplikasi Web Sederhanda

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!")
    })

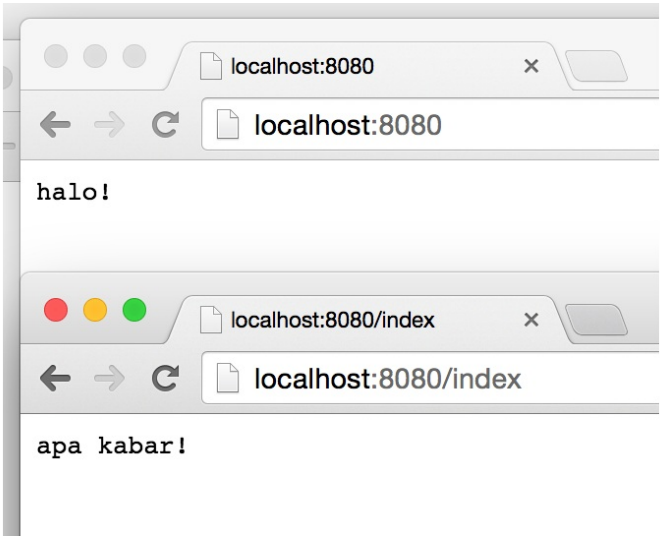
    http.HandleFunc("/index", index)

    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 run` harus dipanggil lagi.

Untuk menghentikan web server, tekan **CTRL+C** pada terminal atau CMD dimana pengeksekusian aplikasi berlangsung.

Penggunaan Template Web

Template 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}}` maksudnya adalah representasi variabel `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"

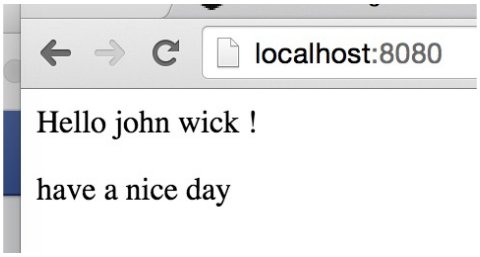
func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        var data = map[string]string{
            "Name":    "john wick",
            "Message": "have a nice day",
        }

        var t, err = template.ParseFiles("template.html")
        if err != nil {
            fmt.Println(err.Error())
        }

        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.

URL Parsing

Data string url bisa dikonversi kedalam bentuk `url.URL`. Tipe tersebut berisikan banyak informasi yang bisa diakses, 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())
    }

    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 $
```

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 oeprasi json.

Di bab ini, kita akan belajar cara untuk konverstri string yang berbentuk json menjadi objek golang, dan sebaliknya.

Decode JSON Ke Variabel Objek Cetakan Struct

Data json tipenya adalah `[]byte`, bisa didapat dari file ataupun string (dengan hasil casting). Dengan menggunakan `json.Unmarshal`, data tersebut 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, dan 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 sederhana, gunakan casting ke `[]byte` agar dideteksi sebagai data json.

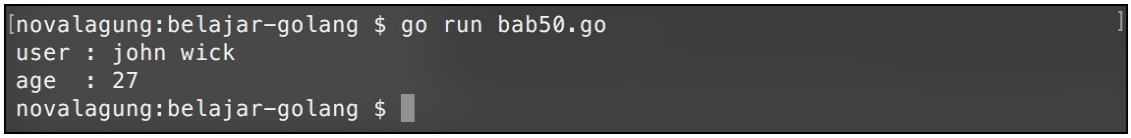
```
func main() {
    var jsonString = `{"Name": "john wick", "Age": 27}`
    var jsonData = []byte(jsonString)

    var data User

    var err = json.Unmarshal(jsonData, &data)
    if err != nil {
        fmt.Println(err.Error())
    }

    fmt.Println("user :", data.FullName)
    fmt.Println("age  :", data.Age)
}
```

Dalam penggunaan fungsi `json.Unmarshal`, variabel yang akan menampung hasil decode harus di-passing sebagai pointer (`&data`).



Pada kode di atas bisa dilihat bahwa salah satu proprty struct `User` ada yang memiliki **tag**, yaitu `FullName`. 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 struct-nya.

Perlu diketahui bahwa untuk decode data json ke variabel objek hasil struct, semua level akses property struct-nya harus publik.

Decode JSON Ke `map[string]interface{}` & `interface{}`

Selain ke variabel objek, 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"])
```

Selain itu, `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"])
```

Decode Array JSON Ke Array Objek

Jika data json adalah array berisikan objek, maka variabel penampung harus bertipe array dengan isi pointer objek. Pada pemanggilan `json.Unmarshal`, variabel tersebut juga di-passing-kan sebagai pointer. Contoh:

```
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())
}

fmt.Println("user 1:", data[0].FullName)
fmt.Println("user 2:", data[1].FullName)
```

Encode Objek Ke JSON

Setelah sebelumnya dijelaskan beberapa cara decode data dari json ke objek, sekarang kita akan belajar cara **encode** data ke bentuk json.

Fungsi `json.Marshal` digunakan untuk decoding data ke json. Data tersebut bisa berupa variabel objek cetakan struct, `map[string]interface{}` , bisa juga bertipe array.

Pada contoh berikut, data array struct akan dikonversi ke dalam bentuk json. Hasil konversi json bisa ditampilkan dengan di-casting terlebih dahulu ke string.

```
var object = []User{{"john wick", 27}, {"ethan hunt", 32}}
var jsonData, err = json.Marshal(object)
if err != nil {
    fmt.Println(err.Error())
}

var jsonString = string(jsonData)
fmt.Println(jsonString)
```

Hasil encode adalah data bertipe `[]byte` . Casting ke `string` bisa digunakan untuk menampilkan data.

```
[novalagung:belajar-golang $ go run bab50.go                                    ]
 [{"Name":"john wick","Age":27},{"Name":"ethan hunt","Age":32}]
 novalagung:belajar-golang $
```

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.

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 yang ditampung oleh 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, diambel 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 payload 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, maka web server sudah live dan bisa dikonsumsi datanya.

```
[novalagung:belajar-golang $ go run bab51.go
 starting web server at http://localhost:8080/
```

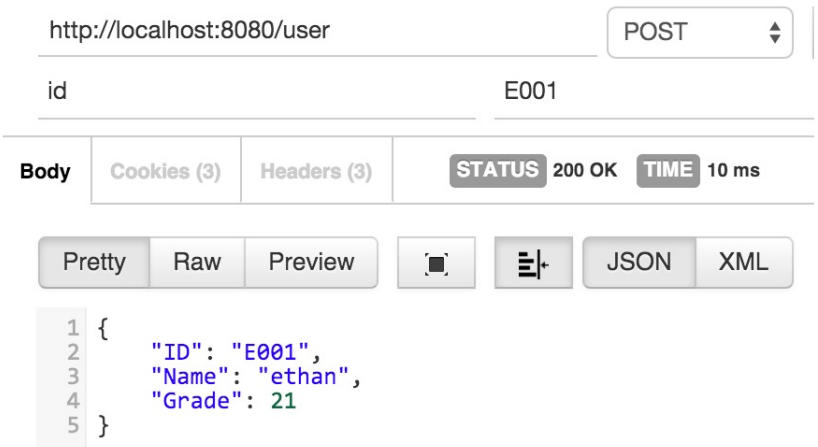
Test API

Setelah web server sudah berjalan, web API yang telah dibuat perlu untuk di tes. Di sini saya menggunakan Go*gle Chrome plugin bernama Postman untuk mengetes API yang sudah dibuat.

- Test `/users`, apakah data yang dikembalikan sudah benar.



- Test `/user` dengan data payload `id` adalah `E001`.



HTTP Request

Setelah di bab sebelumnya kita belajar tentang bagaimana membuat Web API yang menyediakan data dalam bentuk 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 dibahas pada bab tersebut digunakan juga pada bab ini.



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 {
    var err error
    var client = &http.Client{}
    var data []*student

    request, err := http.NewRequest("POST", baseURL+"/users", nil)
    if err != nil {
        fmt.Println(err.Error())
        return data
    }

    response, err := client.Do(request)
    if err != nil {
        fmt.Println(err.Error())
        return data
    }
    defer response.Body.Close()

    err = json.NewDecoder(response.Body).Decode(&data)
    if err != nil {
        fmt.Println(err.Error())
        return data
    }

    return data
}
```

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, payload 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 = fetchUsers()

    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 $
```

HTTP Request Dengan Payload

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 dengan isi adalah sebuah request ke http://localhost:8080/user dengan data yang disisipkan adalah `ID`.

```
func fetchUser(ID string) student {
    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 {
        fmt.Println(err.Error())
        return data
    }
    request.Header.Set("Content-Type", "application/x-www-form-urlencoded")

    response, err := client.Do(request)
    if err != nil {
        fmt.Println(err.Error())
        return data
    }
    defer response.Body.Close()

    err = json.NewDecoder(response.Body).Decode(&data)
    if err != nil {
        fmt.Println(err.Error())
        return data
    }

    return data
}
```

Isi fungsi di atas bisa dilihat memiliki beberapa kemiripan dengan fungsi `fetchUsers()` sebelumnya.

Statement `url.Values{}` akan menghasilkan objek yang nantinya digunakan sebagai payload 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 payload 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 = fetchUser("E001")
    fmt.Printf("ID: %s\t Name: %s\t Grade: %d\n", user1.ID, user1.Name, user1.Grade)
}
```

Pada kode di atas `ID` ditentukan nilainya adalah `"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 $
```

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

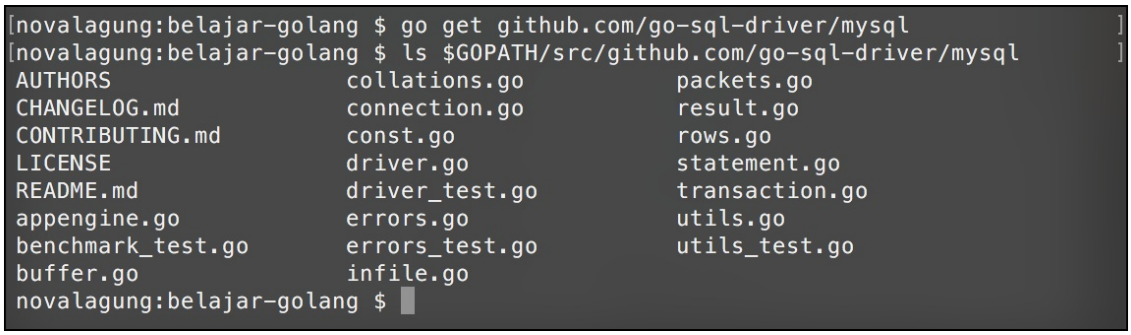
Driver-driver tersebut merupakan proyek open source yang diinisiasi oleh komunitas di Github. Artinya kita selaku developer juga bisa ikut berkontribusi didalamnya.

Pada bab ini kita akan belajar bagaiman berkomunikasi dengan database MySQL menggunakan driver Go-MySQL-Driver.

Instalasi Driver

Undah driver github.com/go-sql-driver/mysql menggunakan `go get`.

```
go get github.com/go-sql-driver/mysql
```



Setup Database

> Sebelumnya pastikan sudah ada mysql server yang terinstal di lokal anda.

Buat database baru bernama `db_belajar_golang`, dan sebuah 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`);
```

Membaca Data Dari MySQL Server

Yang pertama perlu disiapkan adalah meng-import package yang dibutuhkan. Juga, perlu disiapkan sebuah struct dengan skema yang sama seperti pada `tb_student` di database. Nantinya struct ini digunakan sebagai tipe data penampung hasil query.

```
package main

import "fmt"
import "database/sql"
import "os"
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 {
    var db, err = sql.Open("mysql", "root@tcp(127.0.0.1:3306)/db_belajar_golang")
    if err != nil {
        fmt.Println(err.Error())
        os.Exit(0)
    }

    return db
}
```

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`.

Agar tidak bingung, silakan perhatikan skema connection string di bawah ini.

```
user:password@tcp(host:port)/dbname
user@tcp(host:port)/dbname
```

Berikut 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() {
    var db = connect()
    defer db.Close()

    var age = 27
    var rows, err = db.Query("select id, name, grade from tb_student where age = ?", a
ge)
    if err != nil {
        fmt.Println(err.Error())
        os.Exit(0)
    }
    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())
            os.Exit(0)
        }

        result = append(result, each)
    }

    if err = rows.Err(); err != nil {
        fmt.Println(err.Error())
        os.Exit(0)
    }

    for _, each := range result {
        fmt.Println(each.name)
    }
}
```

Setiap kali terbuat koneksi baru, jangan lupa untuk selalu **close** instance koneksinya. 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 terjadinya 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. Variabel yang digunakan untuk menyimpan field-field record dituliskan berurutan nilai pointernya sebagai parameter variadic fungsi tersebut, sesuai dengan field yang di select pada query.

```
// query
select id, name, grade ...

// scan
rows.Scan(&each.id, &each.name, &each.grade ...
```

Setelah itu, data di-append kedalam array `result` lewat statement `result = append(result, each)`.

Setelah fungsi `sqlQuery()` siap, implementasikan pada pada fungsi `main`.

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

```
novalagung:belajar-golang $ go run bab53.go
James Bond
Ethan Hunt
novalagung:belajar-golang $
```

Membaca 1 Record Data Menggunakan Method `QueryRow()`

Untuk query yang menghasilkan 1 baris record saja, bisa gunakan method `QueryRow()` untuk eksekusi query-nya, dengan metode ini kode menjadi lebih ringkas. Chain dengan method `Scan()` untuk mendapatkan value-nya.

```
func sqlQueryRow() {
    var db = connect()
    defer db.Close()

    var result = student{}
    var id = "E001"
    var err = db.QueryRow("select name, grade from tb_student where id = ?", id).Scan(
&result.name, &result.grade)
    if err != nil {
        fmt.Println(err.Error())
        os.Exit(0)
    }

    fmt.Printf("name: %s\ngrade: %d\n", result.name, result.grade)
}

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

```
[novalagung:belajar-golang $ go run bab53.go
name: Ethan Hunt
grade: 2
novalagung:belajar-golang $
```

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() {
    var db = connect()
    defer db.Close()

    var statement, err = db.Prepare("select name, grade from tb_student where id = ?")
    if err != nil {
        fmt.Println(err.Error())
        os.Exit(0)
    }

    var result1 = student{}
    statement.QueryRow("E001").Scan(&result1.name, &result1.grade)
    fmt.Printf("name: %s\ngrade: %d\n", result1.name, result1.grade)

    var result2 = student{}
    statement.QueryRow("W001").Scan(&result2.name, &result2.grade)
    fmt.Printf("name: %s\ngrade: %d\n", result2.name, result2.grade)

    var result3 = student{}
    statement.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 $
```

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() {
    var db = connect()
    defer db.Close()

    var err error

    _, err = db.Exec("insert into tb_student values (?, ?, ?, ?)", "G001", "Galahad",
29, 2)
    if err != nil {
        fmt.Println(err.Error())
    }

    _, err = db.Exec("update tb_student set age = ? where id = ?", 28, "G001")
    if err != nil {
        fmt.Println(err.Error())
    }

    _, err = db.Exec("delete from tb_student where id = ?", "G001")
    if err != nil {
        fmt.Println(err.Error())
    }
}

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

Teknik prepared statement juga bisa digunakan pada metode ini. Berikut merupakan perbandingan eksekusi `Exec()` menggunakan `Prepare()` dan cara biasa.

```
// menggunakan metode prepared statement
var stmt, _ = db.Prepare("insert into tb_student values (?, ?, ?, ?)")
stmt.Exec("G001", "Galahad", 29, 2)

// menggunakan metode biasa
_, _ = db.Exec("insert into tb_student values (?, ?, ?, ?)", "G001", "Galahad", 29, 2)
```

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")
```

Selengkapya mengenai driver yang tersedia bisa dilihat di https://github.com/golang/go/wiki/SQLDrivers

NoSQL MongoDB

Golang tidak menyediakan interface generic untuk NoSQL, hal ini menjadikan driver tiap brand NoSQL untuk Golang bisa berbeda satu dengan lainnya.

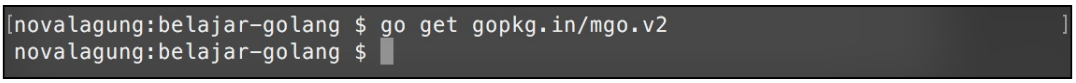
Dari sekian banyak teknologi NoSQL yang ada, yang terpilih untuk dibahas di buku ini adalah MongoDB. Dan pada bab ini kita akan belajar cara berkomunikasi dengan MongoDB menggunakan driver mgo.

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
```



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.

Insert Data

Cara insert data lewat mongo tidak terlalu sulit. Yang pertama perlu dilakukan adalah import package yang dibutuhkan, dan juga menyiapkan struct model.

```
package main

import "fmt"
import "gopkg.in/mgo.v2"
import "os"

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 {
    var session, err = mgo.Dial("localhost")
    if err != nil {
        os.Exit(0)
    }
    return session
}
```

Fungsi `mgo.Dial()` digunakan untuk membuat session baru (bertipe `*mgo.Session`). Fungsi tersebut memiliki sebuah 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 = connect()
    defer session.Close()
    var collection = session.DB("belajar_golang").C("student")

    var err = collection.Insert(&student{"Wick", 2}, &student{"Ethan", 2})
    if err != nil {
        fmt.Println(err.Error())
    }
}

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()` yang digunakan untuk memilih database yang digunakan, dan bisa langsung di chain dengan fungsi `C()` untuk memilih collection.

Setelah mendapatkan instance collection-nya, digunakan method `Insert()` untuk insert data ke database. Method ini memiliki parameter variadic 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

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 = connect()
    defer session.Close()
    var collection = session.DB("belajar_golang").C("student")

    var result = student{}
    var selector = bson.M{"name": "Wick"}
    var err = collection.Find(selector).One(&result)
    if err != nil {
        fmt.Println(err.Error())
    }

    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.

Tipe `bson.M` sebenarnya adalah alias dari `map[string]interface{}`, digunakan dalam penulisan selector.

Selector tersebut kemudian dimasukan sebagai parameter method `Find()`, yang kemudian di chain langsung dengan method `One()` untuk mengambil 1 baris datanya. Kemudian pointer variabel `result` disisipkan sebagai parameter method tersebut.

```
novalagung:belajar-golang $ go run bab54.go
Name  : Wick
Grade : 2
novalagung:belajar-golang $
```

Update Data

Method `Update()` milik struct `mgo.Collection` digunakan untuk update data. Ada 2 parameter yang harus diisi:

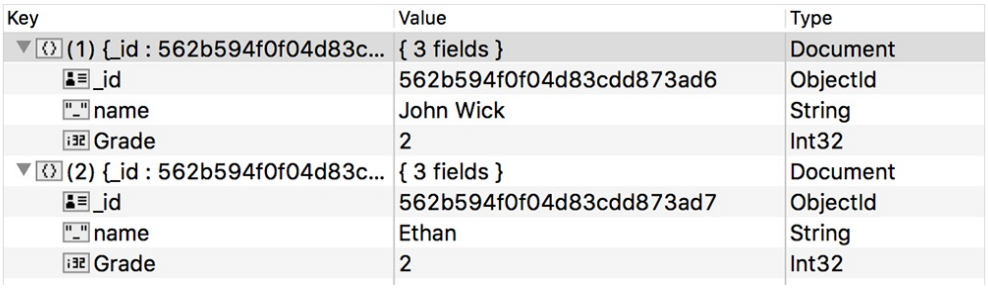
1. Parameter pertama adalah query selector data yang ingin di update
2. Parameter kedua adalah data perubahannya

```
func update() {
    var session = connect()
    defer session.Close()
    var collection = session.DB("belajar_golang").C("student")

    var selector = bson.M{"name": "Wick"}
    var changes = student{"John Wick", 2}
    var err = collection.Update(selector, changes)
    if err != nil {
        fmt.Println(err.Error())
    }
}

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

Bisa dicek lewat Mongo GUI apakah data sudah 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

Menghapus Data

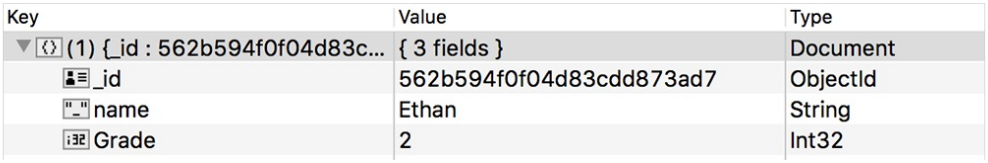
Cara menghapus document pada collection cukup mudah, tinggal gunakan method `Remove()` dengan isi parameter adalah query selector document yang ingin dihapus.

```
func remove() {
    var session = connect()
    defer session.Close()
    var collection = session.DB("belajar_golang").C("student")

    var selector = bson.M{"name": "John Wick"}
    var err = collection.Remove(selector)
    if err != nil {
        fmt.Println(err.Error())
    }
}

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

Automated Testing

Golang menyediakan package `testing`, yang berisikan banyak sekali tools untuk keperluan automation testing.

Pada bab ini kita akan belajar mengenai testing, benchmark, dan juga testing menggunakan testify.

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
}
```

Kode di atas saya simpan ke file bernama `bab55.go`.

Testing

File untuk keperluan testing terpisah dengan file utama, namanya harus berakhiran `_test.go`, dan package-nya harus sama. Pada bab ini, file utama adalah `bab55.go`, maka file testing saya beri nama `bab55_test.go`.

Testing di Golang dituliskan dalam bentuk fungsi, sebagai contoh jika ada 2 fungsi maka ada 2 test. Fungsi tersebut namanya harus diawali dengan **Test** dan memiliki parameter yang bertipe `*testing.T` (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.

Jalankan aplikasi, maka akan terlihat bahwa tidak ada 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 $
```

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.

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 $
```

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
`Skiped()`	Menampilkan laporan skip
`Parallel()`	Menge-set bahwa eksekusi testing adalah parallel

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 $
```

`30000000 51.1 ns/op` artinya adalah perulangan di atas dilakukan sebanyak **30 juta** kali, dalam waktu **51 nano detik** untuk tiap perulangannya.

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 untuk keperluan testing. Caranya cukup mudah, contohnya bisa dilihat pada kode berikut.

```
import "github.com/stretchr/testify/assert"

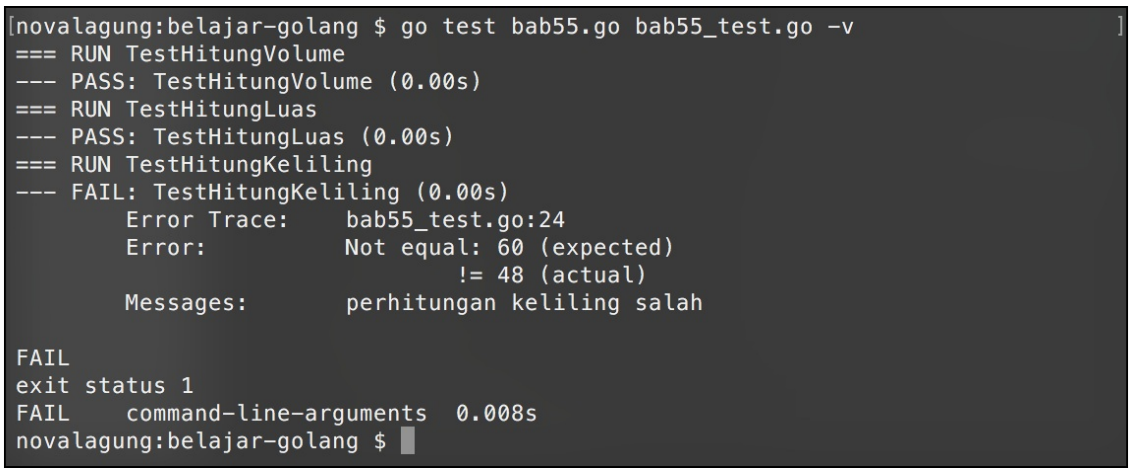
...

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.



WaitGroup

Sebelumnya kita telah belajar banyak mengenai channel, bagaimana sebuah/banyak goroutine dapat dikontrol dengan baik. Pada bab ini topik yang akan dipelajari masih relevan, yaitu tentang manajemen goroutine tetapi tidak menggunakan channel, melainkan `sync.WaitGroup`.

Golang menyediakan package `sync`, yang berisikan cukup banyak API untuk handle masalah threading (dalam kasus ini goroutine). Dari sekian banyak fitur yang ada, pada bab ini kita hanya akan mempelajari `sync.WaitGroup`.

Penerapan `sync.WaitGroup`

`sync.WaitGroup` digunakan menunggu selesainya goroutine yang sedang berjalan. 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`, nantinya digunakan untuk mengontrol proses 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.

```
[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 $
```

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 selain untuk manajemen goroutine, adalah untuk komunikasi data antar goroutine; sedangkan WaitGroup hanya digunakan untuk pengontrolan goroutine saja
- Performa `sync.WaitGroup` lebih baik dibanding channel, sumber: https://groups.google.com/forum/#!topic/golang-nuts/whpCEk9yLhc

Jika muncul pertanyaan manakah yang lebih baik, apakah WaitGroup atau channel, maka kembali ke kebutuhan, karena kedua API tersebut memiliki kelebihan masing-masing.

Mutex

Sebelum kita membahas mengenai apa itu **mutex**? ada baiknya untuk mempelajari terlebih dahulu apa itu **race condition**, karena kedua konsep ini sangat erat hubungannya satu sama lain.

Race condition adalah kondisi dimana lebih dari 1 thread (dalam konteks ini, 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 adalah 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 cara untuk menanggulanginya menggunakan teknik mutex.

Persiapan

Pertama siapkan struct baru bernama `counter`, dengan isi sebuah property `val` bertipe `int`. Property ini nantinya akan dikonsumsi dan diolah oleh banyak goroutine.

Selain itu struct ini juga memiliki beberapa method:

1. Method `Add()` untuk increment nilai
2. Method `Value()` untuk mengembalikan nilai

Jangan lupa juga untuk meng-import package yang dibutuhkan.

```
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 akan kita gunakan sebagai template contoh source code yang ada pada bab ini.

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()
        }()
    }

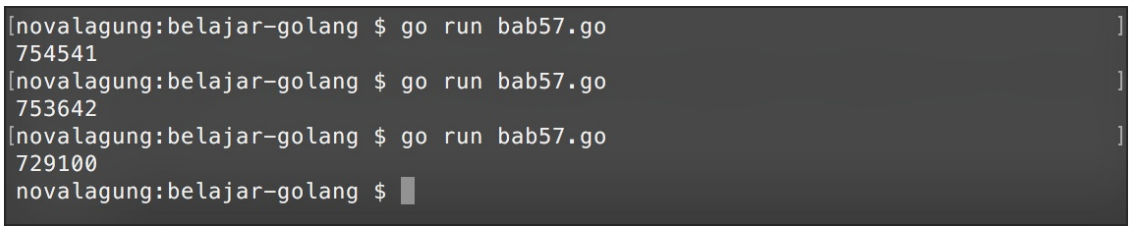
    wg.Wait()
    fmt.Println(meter.Value())
}
```

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 perulanganya 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. Hasilnya bisa dilihat pada gambar berikut.

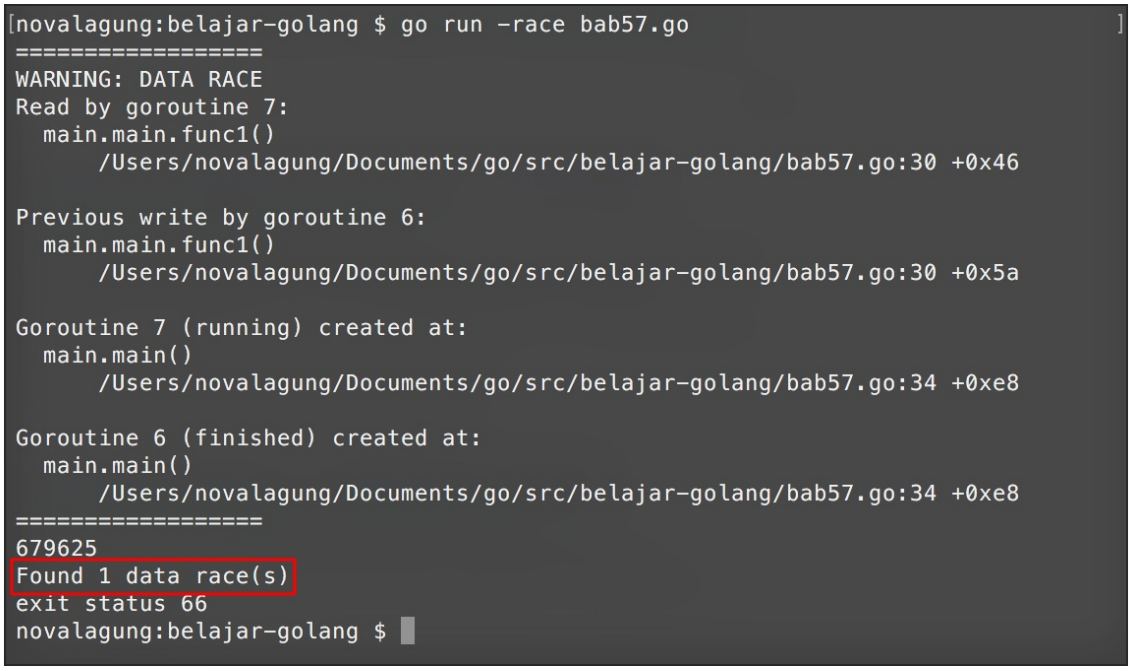


Nilai `meter.val` tidak genap 1000000? kenapa bisa begitu? Padahal seharusnya tidak ada masalah dalam kode yang kita tulis di atas.

Inilah yang disebut dengan race condition, kasus seperti ini memang hanya terjadi dalam **concurrency programming**.

Deteksi Race Condition Menggunakan Golang Race Detector

Golang menyediakan fitur untuk deteksi sebuah race condition. Cara penggunaannya adalah dengan menambahkan flag `-race` pada saat eksekusi aplikasi.



Terlihat pada gambar diatas ada pesan bahwa terdapat sebuah data yang dijadikan bahan balapan oleh goroutine (`Found 1 data race(s)`).

Penerapan `sync.Mutex`

Sekarang kita tahu bahwa program di atas menghasilkan bug race condition. Untuk mengatasi masalah tersebut ada beberapa cara yang bisa digunakan, dan disini kita akan menggunakan `sync.Mutex` .

Ubah kode di atas, tambahkan property baru pada struct `counter` , yaitu `mu` dengan tipe `sync.Mutex` . Variabel ini akan kita gunakan untuk lock dan unlock data `c.val` pada saat pengaksesan lewat method `Add()` dan `Value()` .

```
type counter struct {
    mu  sync.Mutex
    val int
}

func (c *counter) Add(x int) {
    c.mu.Lock()
    c.val++
    c.mu.Unlock()
}

func (c *counter) Value() (x int) {
    c.mu.Lock()
    var val = c.val
    c.mu.Unlock()

    return 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. Bisa dibilang, proses mutual exclusion terjadi diantara kedua method tersebut, `Lock()` dan `Unlock()`.

Tak hanya ketika pengubahan nilai, pada saat pengaksesan juga perlu ditambahkan kedua fungsi ini, agar data yang diambil benar-benar data pada waktu itu.

```
[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 $
```