



## **Pemrograman Go**

**Dr. Bambang Purnomosidi D. P. <bambangpdp@gmail.com>**

**Version 0.0.1-rev-2021-03-28 23:10:40 +0700**

# Daftar Isi

Preface .....	1
Atribusi .....	1
Bagian I: Go dan Peranti Pengembangan Go .....	3
1. Pengenalan Go .....	4
1.1. Apa itu Go? .....	4
1.2. Lisensi Go .....	4
1.3. Instalasi Go .....	4
1.4. Memahami Lingkungan Peranti Pengembangan Go .....	14
2. IDE untuk Go .....	20
2.1. Menggunakan Vim .....	20
2.2. Instalasi Plugin .....	21
2.3. Menggunakan Neovim dan SpaceVim .....	25
2.4. Menggunakan LiteIDE .....	27
2.5. Software IDE Lain .....	27
3. Struktur Program Go .....	29
3.1. Persiapan .....	30
3.2. Tanpa <b>Modules</b> .....	34
3.3. Menggunakan <b>Modules</b> .....	38
Part II: Sintaksis Go .....	43
4. Sintaksis Dasar Go .....	44
4.1. Komentar .....	44
4.2. Variabel .....	44
4.3. Tipe Data Dasar .....	45
4.4. Nilai Default Variabel .....	50
4.5. Operator .....	52
4.6. Konstanta .....	54
4.7. Pointer .....	55
4.8. Struktur Kendali .....	56
5. Fungsi .....	64
5.1. Deklarasi Fungsi .....	64
5.2. Fungsi dengan Banyak Nilai Kembalian .....	65
5.3. <b>Variadic</b> Function .....	66
5.4. Anonymous Functiona / Lambda Abstraction .....	67
5.5. <b>Closures</b> .....	67
5.6. Fungsi Rekursif .....	68

5.7. Call by Value .....	69
5.8. Call by Pointer .....	70
6. Penanganan Kesalahan .....	71
6.1. Penggunaan <b>error</b> .....	71
6.2. Panic dan Recover .....	71
7. Struktur Data .....	75
7.1. Struct .....	75
7.2. Method .....	76
7.3. Interfaces .....	77
7.4. Array .....	78
7.5. Slices .....	82
7.6. Map .....	83
8. Konkurensi dan Paralelisme .....	86
Part III: Go untuk Kasus Spesifik .....	87
9. Testing .....	88
9.1. Sub Section 1 .....	88
10. I/O dan Filesystems .....	89
10.1. Sub Section 1 .....	89
11. Akses Basis Data .....	90
11.1. Sub Section 1 .....	90
12. Sistem Terdistribusi .....	91
12.1. Sub Section 1 .....	91
13. Aplikasi Web .....	92
13.1. Sub Section 1 .....	92

# Preface



Buku ini merupakan buku bebas tentang bahasa pemrograman Go. Go merupakan bahasa pemrograman yang dirancang dan pertama kali diimplementasikan oleh Robert Griesemer, Rob Pike, dan Ken Thompson di Google. Go banyak digunakan pada sistem terdistribusi dan sistem berbasis Cloud. Buku ini dibuat dengan sponsor dari Zimera Systems dan mempunyai lisensi **Creative Commons Attribution-ShareAlike 4.0 International**.



- Lisensi dalam Bahasa Indonesia - <https://creativecommons.org/licenses/by-sa/4.0/deed.id>.
- Lisensi dalam Bahasa Inggris - <https://creativecommons.org/licenses/by-sa/4.0/deed.en>.

Secara umum, penggunaan lisensi ini mempunyai implikasi bahwa pengguna materi:

1. Harus memberikan atribusi ke penulis dan sponsor untuk penulisan materi ini.
2. Boleh menggunakan produk yang ada disini untuk keperluan apapun jika point 1 di atas terpenuhi.
3. Boleh membuat produk derivatif dari produk yang ada disini sepanjang perubahan-perubahan yang dilakukan diberitahukan ke kami dan di-share dengan menggunakan lisensi yang sama.

## Atribusi

**Dr. Bambang Purnomosidi D. P.**

*Zimera Systems*

Dusun Medelan, Umbulmartani, Ngemplak

Sleman, DIY

[https://www.google.com/maps/place/Zimera+Systems/@-](https://www.google.com/maps/place/Zimera+Systems/@-7.6975303,110.43921,17z/data=!3m1!4b1!4m5!3m4!1s0x2e7a5d7cc40e8871:0x2d44da15f0b37)

[7.6975303,110.43921,17z/data=!3m1!4b1!4m5!3m4!1s0x2e7a5d7cc40e8871:0x2d44da15f0b37](https://www.google.com/maps/place/Zimera+Systems/@-7.6975303,110.43921,17z/data=!3m1!4b1!4m5!3m4!1s0x2e7a5d7cc40e8871:0x2d44da15f0b37)

81e!8m2!3d-7.6975303!4d110.4413987

E-mail: [zimera-systems@gmail.com](mailto:zimera-systems@gmail.com)

Pembuatan buku ini merupakan hasil pekerjaan kolektif baik secara langsung maupun tidak langsung. Penulis serta kontributor mengucapkan terima kasih untuk Zimera Systems yang telah memberikan **sponsorship** selama penulisan buku ini.

# **Bagian I: Go dan Peranti Pengembangan Go**

Bagian ini membahas tentang pengenalan Go secara umum dan bagaimana menyiapkan peranti pengembangan jika ingin membuat program menggunakan Go.

# Bab 1. Pengenalan Go

## 1.1. Apa itu Go?

Go adalah nama bahasa pemrograman sekaligus nama implementasi dalam bentuk kompilator (**compiler**). Untuk pembahasan berikutnya, istilah **Go** akan mengacu juga pada spesifikasi bahasa pemrograman serta peranti pengembangannya. Nama yang benar adalah **Go**, bukan **Golang** atau **golang**. Istilah **Golang** atau **golang** muncul karena nama domain **go.org** tidak tersedia saat itu dan mencari sesuatu melalui Google atau mesin pencari lainnya menggunakan kata kunci **Go** tidak menghasilkan hasil yang baik. Dengan demikian, untuk penyebutan di **hashtag** biasanya digunakan **#golang** sehingga mesin pencari bisa mengindeks dan memberikan hasil yang baik. Lihat FAQ tentang Go di [https://golang.org/doc/faq#go\\_or\\_golang](https://golang.org/doc/faq#go_or_golang) untuk informasi lebih lanjut.

## 1.2. Lisensi Go

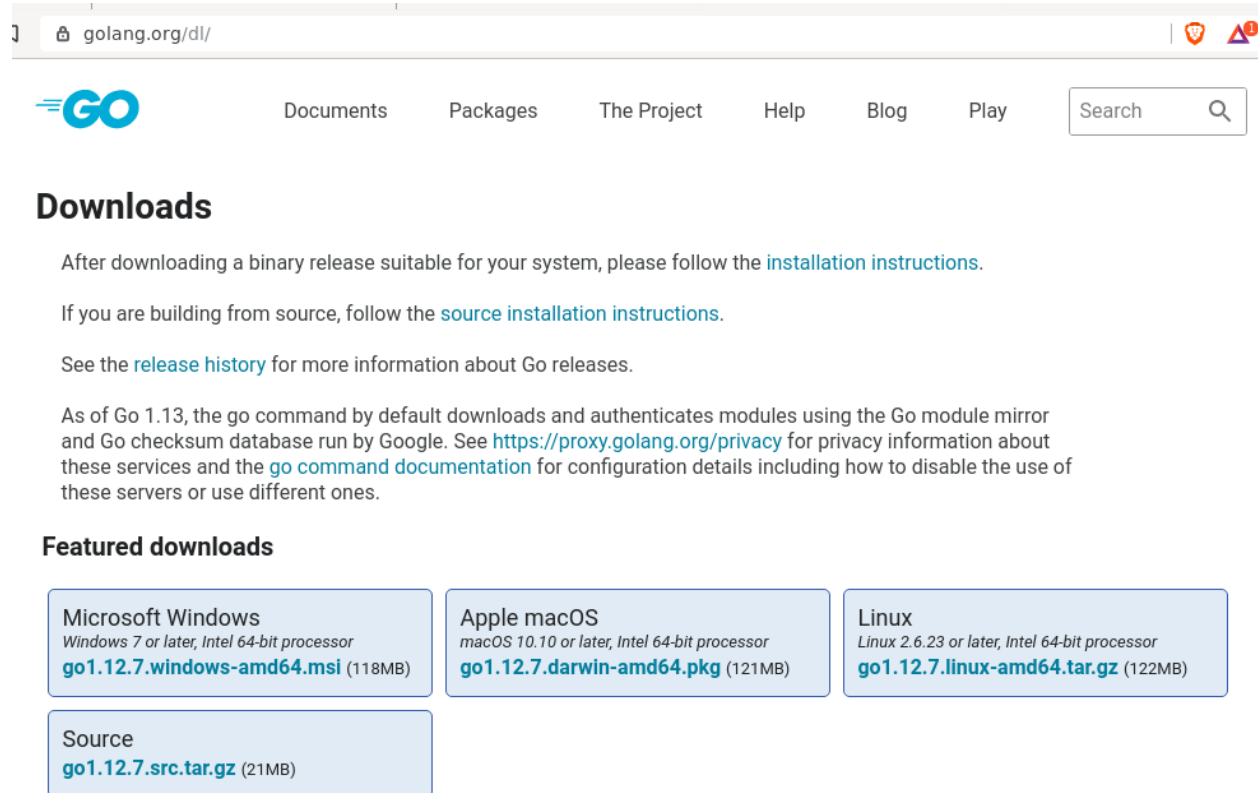
Go didistribusikan dengan menggunakan lisensi modifikasi dari BSD. Lisensi lengkap dari Go bisa diakses di [Lisensi Go](#). Secara umum, penggunaan lisensi ini mempunyai implikasi sebagai berikut:

- boleh digunakan untuk keperluan komersial maupun non-komersial tanpa batasan
- boleh memodifikasi sesuai keperluan
- boleh mendistribusikan
- boleh memberikan sublisensi ke pihak lain
- boleh memberikan garansi
- tidak boleh menggunakan merk dagang Go
- tanpa jaminan dan jika terjadi kerusakan terkait penggunaan software ini maka pemberi lisensi tidak bisa dituntut
- jika mendistribusikan harus mengikutsertakan pemberitahuan hak cipta.

## 1.3. Instalasi Go

Go tersedia pada berbagai platform. Proyek Go sendiri secara resmi mendukung platform Linux, FreeBSD, MacOSX, dan Windows. Dukungan tersebut merupakan dukungan resmi dan

distribusi **binary executable** dari berbagai platform tersebut tersedia di [repository download Go](#) seperti pada gambar berikut:



Dengan dukungan tersebut, Proyek Go akan menerima laporan **bugs** terkait dengan distribusi pada berbagai platform tersebut. Meski demikian, bukan berarti platform-platform lain tidak bisa menggunakan Go karena distribusi dalam bentuk kode sumber tersedia dan telah berhasil dikompilasi ke berbagai platform: NetBSD, OpenBSD, DragonFlyBSD, dan lain-lain. Informasi mengenai platform-platform yang mungkin bisa digunakan oleh Go bisa diperoleh di [wiki](#).

### 1.3.1. Download dan Install Go

Download dan instalasi Go pada tulisan ini adalah download dan instalasi untuk lebih dari satu versi Go dan masing-masing menggunakan workspace sendiri-sendiri. Hal ini disebabkan karena seringkali software yang dibangun ditargetkan untuk lebih dari satu versi, misalnya Go 1.11 ke atas (Go 1.11.x, 1.12.x, dan 1.13.x). Kondisi ini menjadi tidak sederhana karena penulis tidak ingin mencampuraduk kode sumber yang dibuat menggunakan masing-masing versi. Go sendiri menyarankan untuk menggunakan satu workspace untuk semua proyek Go yang kita buat. Satu workspace saja tidak masalah jika hanya menargetkan satu versi. Di bagian ini penulis akan menjelaskan konfigurasi yang penulis gunakan untuk menangani masalah tersebut.



### *Lokasi instalasi Go*

Go akan diinstall di direktori `$HOME/software/go-dev-tools/goVERSI`



VERSI = versi dari Go yang akan diinstall, misalnya `go1.12.7`

Lokasi instalasi tersebut digunakan penulis karena penulis mempunyai lebih dari 1 versi Go, jika nanti ada versi lainnya, versi lain tersebut akan di-install (misal versi 1.13.0) di `$HOME/software/go-dev-tools/go1.13.0`

Meski mendukung banyak platform, di buku ini hanya akan dibahas penggunaan Go di platform Linux. Pada dasarnya peranti pengembang yang disediakan sama. Silahkan menyesuaikan dengan platform yang anda gunakan. Untuk instalasi berikut ini, ambil distribusi yang sesuai dengan platform di komputer anda. Untuk pembahasan ini, digunakan `go1.12.7.linux-amd64.tar.gz`. Setelah itu, ikuti langkah-langkah berikut:

```
$ ls -la
total 475520
drwxr-xr-x  4 bdp bdp      4096 Jul 21 08:16 ./
drwxr-xr-x 88 bdp bdp      4096 Jul 12 14:17 ../
-rw-r--r--  1 bdp bdp 127959471 Jul 19 04:41 go1.12.7.linux-
amd64.tar.gz
...
...
$ mkdir -p $HOME/software/go-dev-tools
$ cd $HOME/software/go-dev-tools
$ tar -xvf ~/master/go/go1.12.7.linux-amd64.tar.gz
$ mv go go1.12.7
```

Setelah menjalankan langkah-langkah di atas, Go sudah terinstall di direktori `$HOME/software/go-dev-tools/go1.12.7`

### **1.3.2. Konfigurasi Variabel Lingkungan Sistem Operasi, Compiler Go, dan Workspace**

Untuk konfigurasi kompiler, ada tiga langkah yang perlu dilakukan: download, ekstrak pada lokasi tertentu, dan terakhir setting environment variables. Pada konfigurasi ini, compiler dan workspace berada pada `$HOME/software/go-dev-tools/`. Lokasi ini selanjutnya akan kita sebut dengan `GODEVTOOLS_HOME`. Setelah download dan install compiler Go seperti langkah di atas, buat struktur direktori sebagai berikut (untuk `go1.12.7` sudah dibuat dengan cara di atas):

```
~/s/go-dev-tools tree . -L 1
.
├── go1.11 -> go1.11.12
├── go1.11.12
├── go1.12 -> go1.12.7
├── go1.12.7
├── go1.13 -> go1.13beta1
├── go1.13beta1
├── go-tools
├── liteide -> liteidex36.0
├── liteidex36.0
└── workspace

10 directories, 0 files
~/s/go-dev-tools
```

Semua versi Go ada di \$GODEVTOOLS\_HOME. Direktori workspace digunakan untuk menyimpan kode sumber yang kita buat sesuai dengan versi Go yang kita targetkan. Untuk setiap direktori di workspace, buat struktur dan 1 file env.sh sebagai berikut:

```
~/s/g/workspace tree . -L 2
.
├── go1.11
│   ├── bin
│   ├── env.fish
│   ├── pkg
│   └── src
├── go1.12
│   ├── bin
│   ├── env.fish
│   ├── pkg
│   └── src
└── go1.13
    ├── bin
    ├── env.fish
    ├── pkg
    └── src

12 directories, 3 files
~/s/g/workspace
```

Isi dari file env.sh menyesuaikan shell yang digunakan:

*Bash*

```
export GOPATH=`pwd`  
export PATH=$PATH:$GOPATH/bin
```

*Fish*

```
set -x GOPATH (pwd)  
set -x PATH $PATH $GOPATH/bin
```

Go menggunakan beberapa variabel lingkungan sistem operasi. Supaya berfungsi dengan baik, tetapkan nilai-nilai variabel lingkungan tersebut di file inisialisasi shell. Jika menggunakan **Fish**, maka inisialisasi tersebut ada di `$HOME/.config/fish/config.fish`. Jika menggunakan **Bash**, maka inisialisasi tersebut diletakkan di `$HOME/.bashrc`). Meski bisa diletakkan pada file tersebut, penulis menyarankan untuk meletakkan pada suatu file text biasa dan kemudian di - **source**. Pada bagian ini, penulis akan meletakkan di file `%HOME/env/fish/go/go1.12`.

```
set GODEVTOOLS_HOME /home/bpdp/software/go-dev-tools  
  
set GO_HOME $GODEVTOOLS_HOME/go1.12  
set LITEIDE_HOME $GODEVTOOLS_HOME/liteide  
set GOTTOOLS $GODEVTOOLS_HOME/go-tools  
  
set -x GOROOT $GO_HOME  
set -x GOOS linux  
set -x GOARCH amd64  
set -x GOHOSTOS linux  
set -x GOHOSTARCH amd64  
  
alias godev='cd $GODEVTOOLS_HOME/workspace/go1.12'  
alias godevtools='cd $GOTTOOLS'  
  
set -x PATH $PATH $GO_HOME/bin $LITEIDE_HOME/bin $GOTTOOLS/bin
```

Jika menggunakan **Bash**:

```
GODEVTOOLS_HOME=/home/bdpd/software/go-dev-tools

GO_HOME=$GODEVTOOLS_HOME/go/go1.12.7
LITEIDE_HOME=$GODEVTOOLS_HOME/liteide
GOTOOLS=$GODEVTOOLS_HOME/go-tools

export GOROOT=$GO_HOME
export GOOS=linux
export GOARCH=amd64
export GOHOSTOS=linux
export GOHOSTARCH=amd64

export PATH=$PATH:$GO_HOME/bin:$LITEIDE_HOME/bin:$GOTOOLS:
$G03RDPARTYTOOLS/bin

alias godev='cd $GODEVTOOLS_HOME/workspace/go1.12'
alias godevtools='cd $GOTOOLS'
```

Dengan memasukkan beberapa variabel lingkungan tersebut ke file, saat kita ingin menggunakan Go, tinggal di - **source** sebagai berikut:

```
$ source ~/env/fish/go/go1.12.7
```

Setelah itu, Go bisa digunakan. Untuk melihat hasil, eksekusi perintah **go env**, hasilnya seharusnya adalah sebagai berikut:

```

$ go env
GOARCH="amd64"
GOBIN=""
GOCACHE="/home/bpdp/.cache/go-build"
GOEXE=""
GOFLAGS=""
GOHOSTARCH="amd64"
GOHOSTOS="linux"
GOOS="linux"
GOPATH="/home/bpdp/go"
GOPROXY=""
GORACE=""
GOROOT="/home/bpdp/software/go-dev-tools/go1.12"
GOTMPDIR=""
GOTOOLDIR="/home/bpdp/software/go-dev-tools/go1.12/pkg/tool/linux_amd64"
GCCGO="gccgo"
CC="gcc"
CXX="g++"
CGO_ENABLED="1"
GOMOD=""
CGO_CFLAGS="-g -O2"
CGO_CPPFLAGS=""
CGO_CXXFLAGS="-g -O2"
CGO_FFLAGS="-g -O2"
CGO_LDFLAGS="-g -O2"
PKG_CONFIG="pkg-config"
GOGCCFLAGS="-fPIC -m64 -pthread -fmessage-length=0 -fdebug-prefix
-map=/tmp/go-build584380045=/tmp/go-build -gno-record-gcc-switches"
$

```

Variabel `$GOPATH` seharusnya menunjuk ke workspace, baru akan berisi nilai yang benar (bukan `$HOME/go`) jika sudah men-**source** file `env.sh` di workspace.

Saat bekerja menggunakan Go, pada dasarnya kita akan menemukan berbagai macam proyek yang bisa dikategorikan menjadi 2 berdasarkan output dari proyek tersebut:

1. **Ready-to-use application**: aplikasi yang siap dipakai, biasanya didistribusikan dalam bentuk **binary executable(s)** atau kode sumber seperti nsq, Hugo, dan lain-lain.
2. Pustaka / **library** maupun aplikasi yang kita kembangkan sendiri.

Untuk dua kategori ini, ada dua perlakuan.

### Ready-to-use application

Untuk kategori ini, siapkan lokasi khusus di media penyimpan untuk menyimpan hasil binary executable, setelah itu set `PATH`, `GOPATH` dan `go get -u -v <repo-url>`. Berikut adalah setting

pada komputer penulis:

```
~/s/g/go-tools tree . -L 1
.
├── bin
├── env.fish
├── go-pkg-needed.sh
├── pkg
└── src

3 directories, 2 files
~/s/g/go-tools cat go-pkg-needed.sh
#!/usr/bin/fish
#go get -u -v github.com/nsf/gocode
# diganti:
go get -u -v github.com/stamblerre/gocode
go get -u -v github.com/rogppe/godef
go get -u -v golang.org/x/lint/golint
go get -u -v github.com/lukehoban/go-outline
go get -u -v github.com/sqs/goreturns
go get -u -v golang.org/x/tools/...
go get -u -v github.com/uudashr/gopkgs
go get -u -v github.com/newhook/go-symbols
go get -u -v github.com/go-delve/delve/cmd/dlv
go get -u -v github.com/pointlander/peg
go get -u -v github.com/songgao/colargo
go get -u -v github.com/motemen/gore
go get -u -v github.com/onsi/ginkgo/ginkgo
go get -u -v github.com/onsi/gomega/...
go get -u -v github.com/smartystrs/goconvey
go get -u -v github.com/blynn/nex
go get -u -v github.com/zmb3/gogetdoc
~/s/g/go-tools
```

Isi dari file `go-pkg-needed.sh` adalah sebagai berikut, anda bisa menambah atau mengurangi sesuai kebutuhan:

```
#!/usr/bin/fish
```

```
# ganti di atas dengan #!/usr/bin/bash jika anda menggunakan Bash
```

```
go get -u -v github.com/stamblerre/gocode
go get -u -v github.com/rogppe/godef
go get -u -v golang.org/x/lint/golint
go get -u -v github.com/lukehoban/go-outline
go get -u -v github.com/sqs/goreturns
go get -u -v golang.org/x/tools/...
go get -u -v github.com/uudashr/gopkgs
go get -u -v github.com/newhook/go-symbols
go get -u -v github.com/go-delve/delve/cmd/dlv
go get -u -v github.com/pointlander/peg
go get -u -v github.com/songgao/colorgo
go get -u -v github.com/motemen/gore
go get -u -v github.com/onsi/ginkgo/ginkgo
go get -u -v github.com/onsi/gomega/...
go get -u -v github.com/smartystricks/goconvey
go get -u -v github.com/blynn/nex
go get -u -v github.com/zmb3/gogetdoc
go get -u -v golang.org/x/tools/gopls
```

Dengan konfigurasi seperti itu, kerjakan berikut ini untuk install:

```
$ source env/fish/go/go1.12.7
$ godevtools
$ source env.sh
$ ./go-pkg-needed.sh
```

Perintah `source env.sh` di atas berguna antara lain untuk menetapkan `$GOPATH` ke `$GOTOOLS`. Setelah proses sebentar, hasil **binary executables** akan diletakkan pada `$GOTOOLS/bin` dan bisa kita jalankan langsung.



jangan meletakkan paket-paket **executables** ini jika `$GOPATH` belum menunjukkan nilai yang benar karena nanti akan tercampur dengan **binary executables** dari distribusi Go.

## Pustaka / library maupun aplikasi yang kita kembangkan sendiri

Untuk keperluan ini biasanya kita menggunakan **modules** yang mulai ada pada versi Go 1.11 dan akan stabil pada versi 1.13. Modules ini akan kita bahas tersendiri.

### 1.3.3. Menguji Instalasi Go

Kode sumber Go yang kita buat bisa dijalankan / dieksekusi tanpa harus dikompilasi (jadi seperti script Python atau Ruby) atau bisa juga dikompilasi lebih dulu untuk menghasilkan **binary executable**. Selain menghasilkan **binary executable**, sebenarnya ada paket pustaka yang dimaksudkan untuk digunakan dalam program (disebut sebagai **package**). Package akan dibahas lebih lanjut pada bab-bab berikutnya.

Untuk menguji, buat program sederhana seperti listing **hello.go**. Setelah itu, gunakan **go run namafile.go** untuk menjalankan secara langsung atau dikompilasi lebih dulu dengan **go build namafile.go**.

```
// hello.go
package main

import "fmt"

func main() {
    fmt.Printf("hello, world\n")
}
```

Berikut ini adalah langkah-langkah untuk mengeksekusi **hello.go**:



```

$ go run hello.go
hello, world
$ go build hello.go
$ ls -la
total 1980
drwxr-xr-x 2 bdp bdp 4096 Jul 21 10:41 ./
drwxr-xr-x 3 bdp bdp 4096 Jul 21 10:40 ../
-rwxr-xr-x 1 bdp bdp 2014135 Jul 21 10:41 hello*
-rw-r--r-- 1 bdp bdp 86 Jul 21 10:40 hello.go
$ file hello
hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically
linked, Go
BuildID=-WQRW-exSunj5kUQwAX9/zYf98wtRiMVNHHsFRqn-/1wN0h--
29c_4cVsSKleo/4LgNCTrEswXoVuMCJgkH, not
stripped
$ strip hello
$ ls -la
total 1400
drwxr-xr-x 2 bdp bdp 4096 Jul 21 10:42 ./
drwxr-xr-x 3 bdp bdp 4096 Jul 21 10:40 ../
-rwxr-xr-x 1 bdp bdp 1420104 Jul 21 10:42 hello*
-rw-r--r-- 1 bdp bdp 86 Jul 21 10:40 hello.go
$ ./hello
hello, world
$ file hello
hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically
linked, Go
BuildID=-WQRW-exSunj5kUQwAX9/zYf98wtRiMVNHHsFRqn-/1wN0h--
29c_4cVsSKleo/4LgNCTrEswXoVuMCJgkH,
stripped
$

```

## 1.4. Memahami Lingkungan Peranti Pengembangan Go

Saat menginstall Go, kita akan memperoleh 3 buah file **binary executable**:

```

$ pwd
/home/bdp/software/go-dev-tools/go1.12.7/bin
$ ls -la
total 34744
drwxr-xr-x 2 bdp bdp 4096 Jul 9 04:32 ./
drwxr-xr-x 10 bdp bdp 4096 Jul 9 04:29 ../
-rwxr-xr-x 1 bdp bdp 14617729 Jul 9 04:31 go*
-rwxr-xr-x 1 bdp bdp 17422226 Jul 9 04:32 godoc*
-rwxr-xr-x 1 bdp bdp 3525802 Jul 9 04:31 gofmt*
$

```

Penjelasan untuk masing-masing akan diuraikan di sub-sub bab berikut.

#### **1.4.1. go**

**go** merupakan peranti untuk mengelola kode sumber Go yang kita buat. Beberapa argumen dari **go** adalah:

```
$ go version
go version go1.12.7 linux/amd64
$ go
Go is a tool for managing Go source code.
```

Usage:

```
go <command> [arguments]
```

The commands are:

bug	start a bug report
build	compile packages and dependencies
clean	remove object files and cached files
doc	show documentation for package or symbol
env	print Go environment information
fix	update packages to use new APIs
fmt	gofmt (reformat) package sources
generate	generate Go files by processing source
get	download and install packages and dependencies
install	compile and install packages and dependencies
list	list packages or modules
mod	module maintenance
run	compile and run Go program
test	test packages
tool	run specified go tool
version	print Go version
vet	report likely mistakes in packages

Use "go help <command>" for more information about a command.

Additional help topics:

buildmode	build modes
c	calling between Go and C
cache	build and test caching
environment	environment variables
filetype	file types
go.mod	the go.mod file
gopath	GOPATH environment variable
gopath-get	legacy GOPATH go get
goproxy	module proxy protocol
importpath	import path syntax
modules	modules, module versions, and more
module-get	module-aware go get
packages	package lists and patterns
testflag	testing flags
testfunc	testing functions

Use "go help <topic>" for more information about that topic.

```
$
```

## 1.4.2. godoc

**godoc** merupakan peranti untuk menampilkan dokumentasi paket pustaka standar Go atau menampilkan server untuk dokumentasi Go (mirip seperti yang terdapat pada [website dokumentasi Go](#)).

```
$ godoc --help
usage: godoc -http=localhost:6060
  -analysis string
      comma-separated list of analyses to perform (supported: type,
pointer). See http://golang.org/lib/godoc/analysis/help.html
  -goroot string
      Go root directory (default "/home/bpdp/software/go-dev-
tools/go1.12")
  -http string
      HTTP service address (default "localhost:6060")
  -index
      enable search index
  -index_files string
      glob pattern specifying index files; if not empty, the index is
read from these files in sorted order
  -index_interval duration
      interval of indexing; 0 for default (5m), negative to only index
once at startup
  -index_throttle float
      index throttle value; 0.0 = no time allocated, 1.0 = full throttle
(default 0.75)
  -links
      link identifiers to their declarations (default true)
  -maxresults int
      maximum number of full text search results shown (default 10000)
  -notes string
      regular expression matching note markers to show (default "BUG")
  -play
      enable playground
  -templates string
      load templates/JS/CSS from disk in this directory
  -timestamps
      show timestamps with directory listings
  -url string
      print HTML for named URL
  -v
      verbose mode
  -write_index
      write index to a file; the file name must be specified with
-index_files
  -zip string
      zip file providing the file system to serve; disabled if empty
$
```

### 1.4.3. gofmt

**gofmt** merupakan peranti untuk mem-format kode sumber dalam bahasa pemrograman Go.

```
$ gofmt --help
usage: gofmt [flags] [path ...]
  -cpuprofile string
    write cpu profile to this file
  -d
    display diffs instead of rewriting files
  -e
    report all errors (not just the first 10 on different lines)
  -l
    list files whose formatting differs from gofmt's
  -r string
    rewrite rule (e.g., 'a[b:len(a)] -> a[b:]')
  -s
    simplify code
  -w
    write result to (source) file instead of stdout
$
```

Untuk melihat bagaimana **gofmt** bisa digunakan untuk membantu memformat kode sumber, buat kode sumber sederhana berikut ini:

```
// hello-unformatted.go
package main
import "fmt"
func main() {
    fmt.Printf("halo\n") // menampilkan tulisan
    fmt.Printf("dunia")  // ini tulisan baris kedua
}
```

Format file kode sumber di atas sebagai berikut:

```
$ gofmt hello-unformatted.go > hello-formatted.go
```

Hasilnya adalah sebagai berikut:

```
// hello-formatted.go
package main

import "fmt"

func main() {
    fmt.Printf("halo\n") // menampilkan tulisan
    fmt.Printf("dunia")  // ini tulisan baris kedua
}
```

## Bab 2. IDE untuk Go

IDE (**I**ntegrated **D**evelopment **E**nvironment) merupakan software yang digunakan oleh pemrogram dan pengembang software untuk membangun software. IDE berisi berbagai fasilitas komprehensif yang diperlukan para pemrogram untuk membantu mereka dalam membangun software aplikasi. Secara minimal, biasanya IDE terdiri atas editor teks (untuk mengetikkan kode sumber), debugger (pencari bugs), **syntax highlighting**, **code completion**, serta dokumentasi / **help**. Bab ini akan membahas beberapa software yang bisa digunakan. Sebenarnya menggunakan editor teks yang menghasilkan file text / ASCII murni sudah cukup untuk bisa menuliskan dan kemudian mengkompilasi kode sumber. Pada bab ini akan dibahas [Vim](#) sebagai editor teks dan [LiteIDE](#) sebagai software IDE yang lebih lengkap untuk Go, tidak sekedar hanya untuk menuliskan kode sumber.

### 2.1. Menggunakan Vim

Untuk menggunakan Vim, ada plugin utama serta berbagai plugin pendukung yang bisa digunakan. Untuk instalasi berbagai plugin tersebut, ada 2 kemungkinan:

1. Jika Vim anda menggunakan Vim sebelum versi 8, gunakan [Pathogen](#).
2. Jika Vim anda versi 8 atau lebih tinggi, gunakan **packages** dari Vim untuk **native package management**.

#### 2.1.1. Menggunakan Pathogen

Pathogen adalah plugin dari Tim Pope yang digunakan untuk mempermudah pengelolaan plugin. Kode sumber dari Pathogen bisa diperoleh di <https://github.com/tpope/vim-pathogen>. Untuk instalasi, ikuti langkah berikut:

```

$ cd
$ mkdir .vim/autoload
$ mkdir .vim/bundle
$ cd .vim/autoload
$ wget -c https://raw.githubusercontent.com/tpope/vim-
pathogen/master/autoload/pathogen.vim
--2019-07-23 13:18:11-- https://raw.githubusercontent.com/tpope/vim-
pathogen/master/autoload/pathogen.vim
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
151.101.8.133
Menghubungi raw.githubusercontent.com (raw.githubusercontent.com
)|151.101.8.133|:443... terhubung.
Permintaan HTTP dikirimkan, menunggu balasan... 200 OK
Besar: 8848 (8,6K) [text/plain]
Simpan ke: `pathogen.vim'

pathogen.vim
100%[=====>]
8,64K  --.-KB/s    in 0s

2019-07-23 13:18:12 (41,2 MB/s) - `pathogen.vim' disimpan [8848/8848]

$

```

Setelah itu, untuk menggunakan Pathogen, letakkan aktivasinya di `$HOME/.vimrc` atau di-copy satu direktori ke direktori `$HOME/.vim/bundle`.

### 2.1.2. Native Package Management

Dengan menggunakan cara ini, kita hanya perlu menyediakan direktori `$HOME/.vim/pack/default/start`, setelah itu, copy semua repo plugin ke lokasi direktori tersebut masing-masing menempati satu direktori.

## 2.2. Instalasi Plugin

Setelah selesai melakukan persiapan di atas, berbagai plugin yang diperlukan bisa diambil langsung dari Internet. Berikut ini adalah daftar yang digunakan penulis:

- `nerdtree`: untuk menampilkan file-file dalam struktur pohon di sebelah kiri sehingga memudahkan navigasi.
- `nerdtree-git-plugin`: untuk menampilkan status Git.
- `vim-go`: plugin utama agar Vim mengenali kode sumber Go.

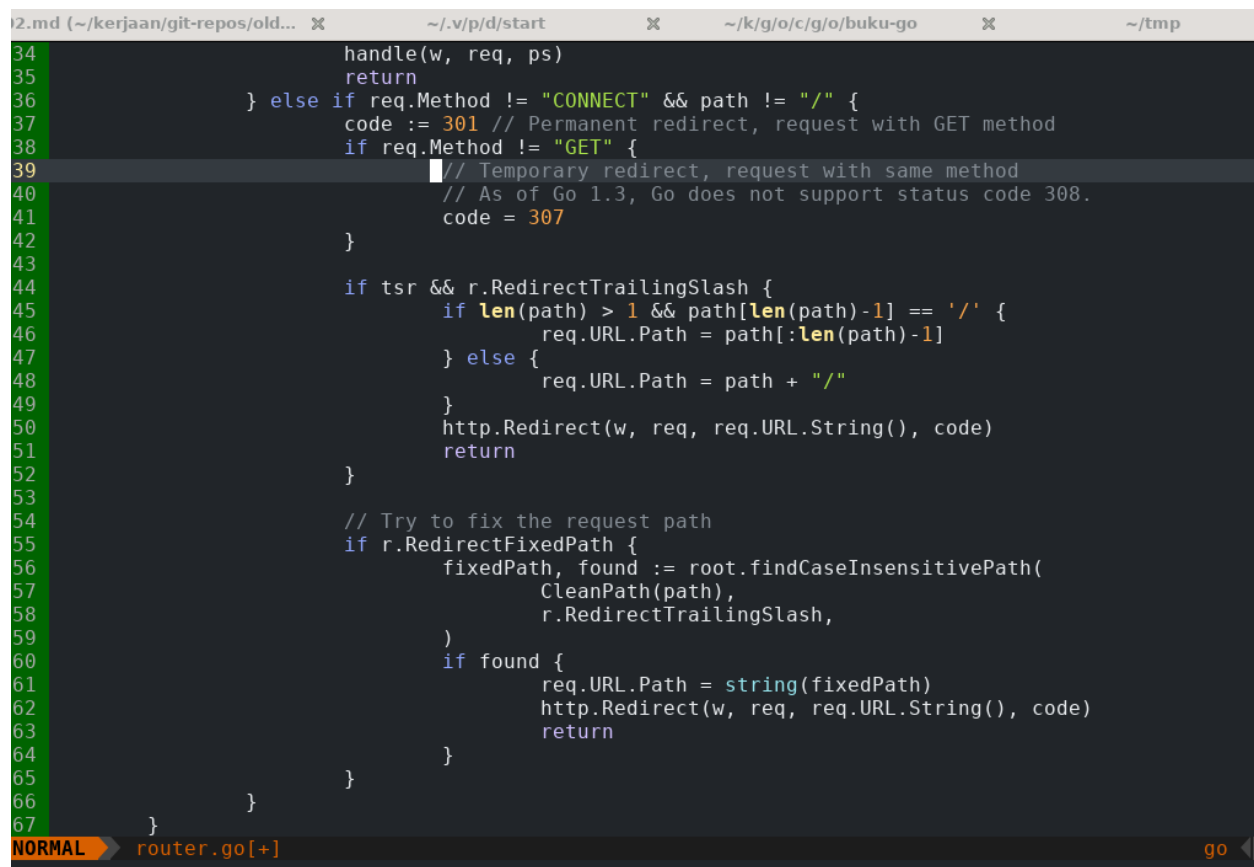


- [vim-airline](#): untuk menampilkan status/tabline dengan format yang lebih bagus.
- [vim-airline-themes](#): themes dari vim-airline.
- [vim-open-color](#): skema warna Vim menggunakan **open color**.

Cara instalasi:

```
$ cd
$ cd .vim/bundle
atau
$ cd .vim/pack/default/start
$ git clone <masing-masing lokasi plugin>
```

Hasil dari menjalankan **vim** melalui shell untuk menulis kode sumber Go bisa dilihat pada gambar berikut ini:



```
34     handle(w, req, ps)
35     return
36 } else if req.Method != "CONNECT" && path != "/" {
37     code := 301 // Permanent redirect, request with GET method
38     if req.Method != "GET" {
39         // Temporary redirect, request with same method
40         // As of Go 1.3, Go does not support status code 308.
41         code = 307
42     }
43
44     if tsr && r.RedirectTrailingSlash {
45         if len(path) > 1 && path[len(path)-1] == '/' {
46             req.URL.Path = path[:len(path)-1]
47         } else {
48             req.URL.Path = path + "/"
49         }
50         http.Redirect(w, req, req.URL.String(), code)
51         return
52     }
53
54     // Try to fix the request path
55     if r.RedirectFixedPath {
56         fixedPath, found := root.findCaseInsensitivePath(
57             CleanPath(path),
58             r.RedirectTrailingSlash,
59         )
60         if found {
61             req.URL.Path = string(fixedPath)
62             http.Redirect(w, req, req.URL.String(), code)
63             return
64         }
65     }
66 }
67 }
```

NORMAL router.go[+] go

## 2.2.1. Autocompletion

Vim menyediakan fasilitas **autocompletion** melalui **Omniautocompletion**. Fasilitas ini sudah terinstall saat kita menginstall Vim. Untuk mengaktifkan fasilitas ini untuk keperluan Go, kita

harus menginstall dan mengaktifkan `gopls`. Gopls sudah terinstall setelah selesai mengerjakan instalasi di bab 1. Hasil dari instalasi Gopls adalah file **binary executable** `$GOPATH/bin/gopls` (sesuai letak GOPATH di `env.sh`). Untuk konfigurasi, tambahkan satu baris di `$HOME/.vim/vimrc`: `set ofu=syntaxcomplete#Complete` di bawah baris `filetype plugin indent on`.

Kode sumber lengkap dari `$HOME/.vim/vimrc` yang penulis gunakan bisa dilihat pada listing berikut ini:

```
set number
set linebreak
set showbreak=+++
set textwidth=100
set showmatch
set nocompatible

set hlsearch
set smartcase
set ignorecase
set incsearch

set autoindent
set expandtab
set shiftwidth=2
set smartindent
set smarttab
set softtabstop=2

set ruler

set undolevels=1000
set backspace=indent,eol,start

filetype plugin indent on
set ofu=syntaxcomplete#Complete

" Use 24-bit (true-color) mode in Vim/Neovim when outside tmux or screen.
" If you're using tmux version 2.2 or later, you can remove the outermost
$TMUX
" check and use tmux's 24-bit color support
" (http://sunaku.github.io/tmux-24bit-color.html#usage for more
information.)
if empty($TMUX) && empty($STY)
    " See https://gist.github.com/XVilka/8346728.
    if $COLORTERM =~# 'truecolor' || $COLORTERM =~# '24bit'
        if has('termguicolors')
            " See :help xterm-true-color
            if $TERM =~# '^screen'
```

```

        let &t_8f = "\<Esc>[38;2;%lu;%lu;%lum"
        let &t_8b = "\<Esc>[48;2;%lu;%lu;%lum"
    endif
    set termguicolors
endif
endif
endif

set background=dark
colorscheme open-color
syntax on
highlight LineNr term=bold cterm=NONE ctermfg=DarkGrey ctermbg=NONE gui
=NONE guifg=DarkGrey guibg=darkgreen
set cursorline
" set cursorcolumn

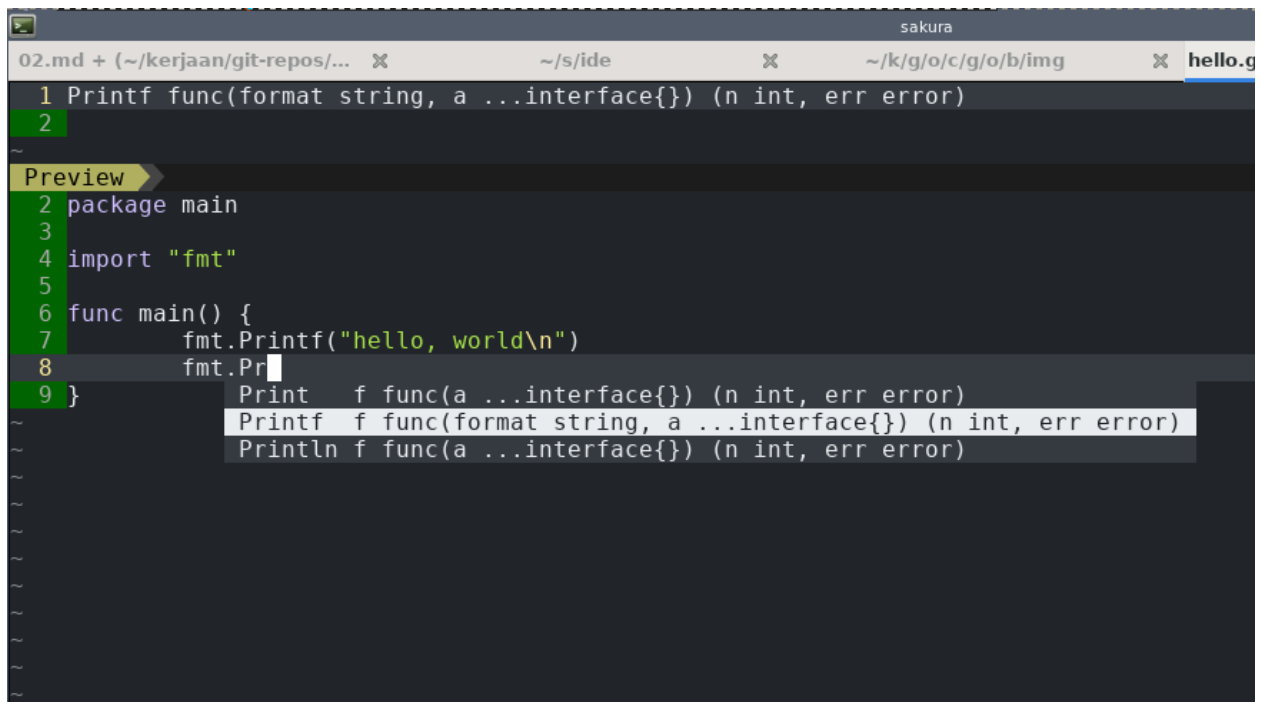
set guifont=Monospace\ 14

" nerdtree
let g:NERDTreeWinPos = "right"
autocmd bufenter * if (winnr("$") == 1 && exists("b:NERDTree") &&
b:NERDTree.isTabTree()) | q | endif
let g:NERDTreeNodeDelimiter = "\u00a0"
nnoremap <F4> :NERDTreeToggle<CR>
let g:NERDTreeFileExtensionHighlightFullName = 1
let g:NERDTreeExactMatchHighlightFullName = 1
let g:NERDTreePatternMatchHighlightFullName = 1
let g:NERDTreeHighlightFolders = 1 " enables folder icon highlighting
using exact match
let g:NERDTreeHighlightFoldersFullName = 1 " highlights the folder name
let NERDTreeShowHidden=1

" airline
let g:airline_powerline_fonts = 1
let g:airline_theme='distinguished'

```

Untuk mengaktifkan completion, kita harus masuk ke mode **Insert** dari Vim, setelah itu tekan **Ctrl-X**, **Ctrl-O** secara cepat. Hasil **autocompletion** bisa dilihat di gambar berikut ini:



## 2.3. Menggunakan Neovim dan SpaceVim

Untuk keperluan ini, install [Neovim](#) kemudian pastikan bahwa [gopls](#) juga sudah terinstall (lihat bab 1). Setelah itu, gunakan [SpaceVim](#) sebagai berikut:

- ## 1. Clone SpaceVim:

```
$ cd
$ curl -sLf https://spacevim.org/install.sh | bash -s -- --install neovim
```

Hasil dari langkah di atas adalah direktori `$HOME/.SpaceVim`. Untuk update SpaceVim, lakukan `git pull` pada direktori tersebut. Untuk konfigurasi Neovim + SpaceVim sebagai IDE untuk Go, gunakan konfigurasi di `$HOME/.SpaceVim.d/init.toml` sebagai berikut:

```

#=====
====
# dark_powered.toml --- dark powered configuration example for SpaceVim
# Copyright (c) 2016-2017 Wang Shidong & Contributors
# Author: Wang Shidong <wsdjeg at 163.com >
# URL: https://spacevim.org
# License: GPLv3
#=====
====

# All SpaceVim option below [option] section
[options]
# set spacevim theme. by default colorscheme layer is not loaded,
# if you want to use more colorscheme, please load the colorscheme
# layer
colorscheme = "gruvbox"
background = "dark"
# Disable guicolors in basic mode, many terminal do not support 24bit
# true colors
enable_guicolors = true
# Disable statusline separator, if you want to use other value, please
# install nerd fonts
statusline_separator = "arrow"
statusline_inactive_separator = "arrow"
buffer_index_type = 4
enable_tabline_filetype_icon = true
enable_statusline_display_mode = false

# Enable autocomplete layer
[[layers]]
name = 'autocomplete'
auto-completion-return-key-behavior = "complete"
auto-completion-tab-key-behavior = "smart"

[[layers]]
name = 'shell'
default_position = 'top'
default_height = 30

[[layers]]
name = "lang#go"

[[layers]]
name = "format"

```

Hasil dari konfigurasi di atas untuk proses edit kode sumber Go adalah sebagai berikut:

```
02.md + (~/kerjaan/git-r... x ~/s/ide x ~/k/g/o/c/g/o/b/img x ~/k/s/go x ~/m
1 hello.go
7 // hello.go
6 package main
5
4 import "fmt"
3
2 func main() {
1   fmt.Printf("hello, world\n")
8   fmt.|
1 }
Errorf      f func(format string, a ...interface{}) error
Formatter   t interface{...}
Fprint      f func(w io.Writer, a ...interface{}) (n int, err error)
Fprintf     f func(w io.Writer, format string, a ...interface{}) (n int, err error)
Fprintln    f func(w io.Writer, a ...interface{}) (n int, err error)
Fscan       f func(r io.Reader, a ...interface{}) (n int, err error)
Fscanf      f func(r io.Reader, format string, a ...interface{}) (n int, err error)
Fscanln     f func(r io.Reader, a ...interface{}) (n int, err error)
GoStringer  t interface{...}
Print       f func(a ...interface{}) (n int, err error)
Printf      f func(format string, a ...interface{}) (n int, err error)
Println     f func(a ...interface{}) (n int, err error)
Scan        f func(a ...interface{}) (n int, err error)
ScanState   t interface{...}
Scanf       f func(format string, a ...interface{}) (n int, err error)
1 * 86 bytes hello.go go vim-go: [completion] SUCCESS
```

## 2.4. Menggunakan LiteIDE



LiteIDE dibuat oleh visualfc dan tersedia dalam bentuk kode sumber maupun binary. Kode sumber bisa diperoleh di <https://github.com/visualfc/liteide>. Installer executable bisa diperoleh di <http://sourceforge.net/projects/liteide/files>.

Instalasi di Linux sangat mudah, hanya tinggal mengekstrak file yang kita download pada suatu in menjalankan cukup dengan mengeksekusi file `$LITEIDE_HOME/bin/liteide` (`cd $LITEIDE_HOME/bin; ./liteide &`)

## 2.5. Software IDE Lain

Vim dan LiteIDE hanyalah beberapa peranti yang bisa digunakan oleh pengembang. Distribusi

Go juga menyediakan dukungan untuk berbagai peranti lunak lain:

- Emacs. Dukungan untuk Go diwujudkan dalam fasilitas **add-on**. Untuk Emacs 24 ke atas, bisa diinstall melalui manajer paket (M-x package-list-packages), cari dan install **go-mode**. Emacs juga mendukung **gopls** untuk **completion**.
- Eclipse. Dukungan untuk Go diwujudkan melalui plugin **goclipse**, bisa diperoleh di <https://code.google.com/p/goclipse/>.
- Selain software-software yang telah disebutkan, rata-rata IDE / Editor sudah mempunyai dukungan terhadap bahasa pemrograman Go (JEdit, Sublime-text, Notepad++, dan lain-lain).
- **Visual Studio Code** mempunyai dukungan yang kuat untuk Go dengan menggunakan ekstensi **Go for Visual Studio Code** - <https://marketplace.visualstudio.com/items?itemName=ms-vscode.Go>.

## Bab 3. Struktur Program Go

Secara umum, teknik penulisan kode sumber pada Go ini harus dipahami terlebih dahulu sebelum memulai **coding**. Go telah melalui berbagai teknik penulisan kode sumber. Sampai saat ini, kita bisa memisahkan menjadi 2 bagian besar:

1. Tanpa **modules** (Go sebelum 1.11)
2. Menggunakan **modules** (Go 1.11 ke atas dengan transisi pada Go 1.11 dan Go 1.12, mulai menggunakan **modules** secara penuh pada versi 1.13).

Pada awalnya, Go menggunakan variabel lingkungan `$GOPATH` untuk mengelola proyek. Biasanya - seperti terlihat pada bab awal buku ini - `$GOPATH` berisi direktori workspace tempat **developer** menuliskan kode sumber. Jika kode sumber sudah cukup kompleks dan melibatkan banyak pustaka internal maupun eksternal, maka `GOPATH` ini perlu diatur, jika sederhana (hanya 1 **binary executable** tanpa pustaka internal maupun eksternal), maka `$GOPATH` tidak perlu diatur. Ketentuan untuk `$GOPATH` ini adalah sebagai berikut:

1. Lokasi pembuatan kode sumber disebut **workspace**.
2. Setiap **workspace** berisi direktori **bin**, **pkg**, dan **src**.
3. Jika `$GOPATH` tidak ditetapkan, maka workspace default akan berada di `$HOME/go`
4. Jika `$GOPATH` ditetapkan, **workspce** akan berada pada nilai dari `$GOPATH`.

Isi dari **workspace** adalah sebagai berikut:

```
$ ls -la
total 24
drwxr-xr-x 5 bdp bdp 4096 Jan 7 2019 ./
drwxr-xr-x 5 bdp bdp 4096 Jul 19 04:48 ../
drwxr-xr-x 2 bdp bdp 4096 Jan 7 2019 bin/
-rwxr-xr-x 1 bdp bdp 50 Jan 7 2019 env.sh
drwxr-xr-x 2 bdp bdp 4096 Jan 7 2019 pkg/
drwxr-xr-x 2 bdp bdp 4096 Jan 7 2019 src/
$
```

Isi dari **env.sh** adalah:

```
export GOPATH=`pwd`
export PATH=$PATH:$GOPATH/bin
```



Penjelasan masing-masing direktori tersebut:

- bin: berisi hasil kompilasi aplikasi
- pkg: berisi hasil kompilasi pustaka
- src: kode sumber untuk pustaka serta aplikasi

### 3.1. Persiapan

Untuk mempelajari bab ini, ada beberapa kode sumber yang harus disiapkan. Kode sumber ini akan ditulis disini lebih dahulu supaya memudahkan untuk diacu pada pembahasan berikutnya.

Nama file: `showuserenv.go`

```

/*
    showuserenv.go

    Contoh program sederhana untuk menjelaskan
    struktur program Go untuk aplikasi executable

*/

// Program Go diawali dengan nama paket.
// Paket untuk aplikasi executable selalu berada
// pada paket main.
package main

// pustaka standar yang diperlukan
// Jika hanya satu:
// import "fmt"
// Jika lebih dari satu:
import (
    "fmt"
    "os"
)

// "Fungsi" merupakan satuan terintegrasi dari
// program Go, selalu diberi nama "main" untuk
// aplikasi executable.
func main() {

    // ini adalah kode sumber / program Go
    // akan dijelaskan lebih lanjut, abaikan
    // jika belum paham
    var (
        user    string
        homeDir string
    )

    user = os.Getenv("USER")
    homeDir = os.Getenv("HOME")

    fmt.Printf("Halo %s", user)
    fmt.Printf("\nHome anda di %s", homeDir)
    fmt.Printf("\n")
}

```

Nama file: `showgoenv.go`

```

/*
    showgoenv.go

    Contoh program sederhana untuk menjelaskan
    struktur program Go untuk lebih dari satu
    binary executable

*/

package main

import (
    "fmt"
    "os"
)

func main() {

    var (
        user      string
        goHome    string
        goPath    string
    )

    user = os.Getenv("USER")
    goHome = os.Getenv("GOROOT")
    goPath = os.Getenv("GOPATH")

    fmt.Printf("Halo %s", user)
    fmt.Printf("\nAnda menggunakan Go di %s", goHome)
    fmt.Printf("\nGOPATH anda di %s", goPath)
    fmt.Printf("\n")

}

```

Nama file: `reverse.go`

```

/*
    reverse.go
    Contoh pustaka sederhana untuk membalik kata.
    diambil dari https://golang.org/doc/code.html

*/
package stringutil

// Reverse returns its argument string reversed rune-wise left to right.
func Reverse(s string) string {
    r := []rune(s)
    for i, j := 0, len(r)-1; i < len(r)/2; i, j = i+1, j-1 {
        r[i], r[j] = r[j], r[i]
    }
    return string(r)
}

```

Nama file: **hellostringutil.go**

```

/*
    hellostringutil.go
    Contoh sederhana untuk menggambarkan cara menggunakan lib
    Diambil dari https://golang.org/doc/code.html

*/
package main

import (
    "fmt"
    "github.com/bpdp/stringutil"
)

func main() {
    fmt.Printf(stringutil.Reverse("Hello, World!"))
}

```

Nama file: **hellorsc.go**

```

package main

import (
    "fmt"
    "rsc.io/quote"
)

func main() {
    fmt.Println(quote.Hello())
}

```

Nama file: **gomtk.go**

```

package gomtk

func Add(x int, y int) int {
    return x + y
}

```

Nama file: **gomtktest.go**

```

package main

import (
    "fmt"
    adder "github.com/oldstager/gomtk"
)

func main() {
    fmt.Println(adder.Add(2, 4))
}

```

## 3.2. Tanpa Modules

### 3.2.1. Program Aplikasi Sederhana - 1 File **binary executable** Utama

Suatu aplikasi **executable** (artinya bisa dijalankan secara langsung oleh sistem operasi) mempunyai struktur seperti yang terlihat pada listing **showuserenv.go**. Untuk kasus sederhana dan tanpa ketergantungan kepada pustaka eksternal seperti ini, file bisa diletakkan dimana saja. Untuk menjalankan kode sumber tersebut, ikuti langkah-langkah berikut:

## Tanpa Proses Kompilasi

```
$ go run showuserenv.go
Halo bdpd
Home anda di /home/bdpd
```

## Mengkompilasi Menjadi Binary Executable

```
$ go build showuserenv.go
$ ls -la
total 1992
drwxr-xr-x 2 bdpd bdpd    4096 Jul 29 06:41 ./
drwxr-xr-x 3 bdpd bdpd    4096 Jul 29 06:36 ../
-rwxr-xr-x 1 bdpd bdpd 2027001 Jul 29 06:41 showuserenv*
-rw-r--r-- 1 bdpd bdpd    813 Jul 29 06:39 showuserenv.go
$ ./showuserenv
Halo bdpd
Home anda di /home/bdpd
$ file showuserenv
showuserenv: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
statically linked, Go BuildID
=9rxlkJ0BRey3wgq8FCzN/hTjP17z9sr3yZTVx1JEZ/KJmV7CpJpm_WaUyomyhS/rvGW2o0cNy
_kmNe0caJe, not stripped
$ strip showuserenv
$ ls -la
total 1408
drwxr-xr-x 2 bdpd bdpd    4096 Jul 29 06:42 ./
drwxr-xr-x 3 bdpd bdpd    4096 Jul 29 06:36 ../
-rwxr-xr-x 1 bdpd bdpd 1428296 Jul 29 06:42 showuserenv*
-rw-r--r-- 1 bdpd bdpd    813 Jul 29 06:39 showuserenv.go
$ ./showuserenv
Halo bdpd
Home anda di /home/bdpd
$
```

### 3.2.2. Program Aplikasi: Lebih dari 1 File **binary executable** tanpa ketergantungan pustaka eksternal

Jika tanpa pustaka internal maupun eksternal, maka membangun lebih dari satu **binary executable** dilakukan cukup dengan meletakkan pada sembarang direktori dan mem-**build** satu persatu.

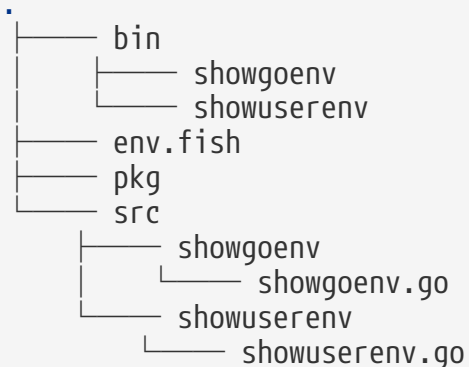
```

$ ls -la
total 16
drwxr-xr-x 2 bdp bdp 4096 Jul 29 06:52 ./
drwxr-xr-x 4 bdp bdp 4096 Jul 29 06:51 ../
-rw-r--r-- 1 bdp bdp 503 Jul 29 06:52 showgoenv.go
-rw-r--r-- 1 bdp bdp 813 Jul 29 06:51 showuserenv.go
$ go build showgoenv.go
$ go build showuserenv.go
$ ls -la
total 3976
drwxr-xr-x 2 bdp bdp 4096 Jul 29 06:52 ./
drwxr-xr-x 4 bdp bdp 4096 Jul 29 06:51 ../
-rwxr-xr-x 1 bdp bdp 2027001 Jul 29 06:52 showgoenv*
-rw-r--r-- 1 bdp bdp 503 Jul 29 06:52 showgoenv.go
-rwxr-xr-x 1 bdp bdp 2027001 Jul 29 06:52 showuserenv*
-rw-r--r-- 1 bdp bdp 813 Jul 29 06:51 showuserenv.go
$

```

### 3.2.3. Program Aplikasi: Lebih dari 1 File **binary executable**

Untuk keperluan ini, buat **workspace** seperti petunjuk di awal bab, setelah itu, letakkan file **showuserenv.go** dan **showgoenv.go** masing-masing dalam **sub package** tersendiri. Perhatikan struktur direktori berikut:



Untuk mengkompilasi, **env.sh** sudah harus di-**source** terlebih dahulu. Setelah itu kompilasi sekaligus install:

```
$ go install ...
$ ls -la bin/
total 3968
drwxr-xr-x 2 bdp bdp 4096 Jul 29 20:43 ./
drwxr-xr-x 5 bdp bdp 4096 Jul 29 20:59 ../
-rwxr-xr-x 1 bdp bdp 2027001 Jul 29 21:01 showgoenv*
-rwxr-xr-x 1 bdp bdp 2027001 Jul 29 21:01 showuserenv*
$
```

### 3.2.4. Pustaka / Library / Package dan Penggunaannya

Ada kalanya, para software developer membangun pustaka yang berisi berbagai fungsionalitas yang bisa digunakan kembali suatu saat nanti. Untuk keperluan ini, Go menyediakan fasilitas untuk membangun library dalam bentuk kumpulan fungsi. Kumpulan fungsi ini nantinya akan diletakkan pada suatu repo tertentu sehingga bisa langsung di `go get <lokasi repo pustaka>`. Pada penjelasan berikut ini, kita akan membangun suatu aplikasi kecil (`hellostringutil`) yang menggunakan suatu pustaka yang sebelumnya sudah kita bangun (`stringutil/Reverse` - untuk membalik kata). Kode sumber diambil dari [How to write Go code](#). Semua kode sumber, baik untuk pustaka ataupun aplikasi akan diletakkan pada pola direktori tertentu. Go menggunakan pola repo untuk penamaan / pengelompokan aplikasi atau pustaka meskipun belum dimasukkan ke repo di Internet. Sebaiknya membiasakan diri sejak awal menggunakan pola tersebut meskipun belum akan dimasukkan ke repositori di Internet. Untuk mengerjakan bagian ini, buat **workspace** terlebih dahulu.

#### Membuat Pustaka

Kode sumber untuk pustaka (`reverse.go`) ini akan diletakkan di `src/github.com/oldstager/stringutil`. Paket yang dibuat dengan penamaan ini, nantinya akan diacu dalam `import` sebagai `github.com/oldstager/stringutil`. Untuk mengkompilasi:

```
$ go build github.com/oldstager/stringutil
$
```

Jika tidak ada kesalahan, maka akan langsung kembali ke prompt shell.

#### Membuat Aplikasi yang Memanfaatkan Pustaka

Sama halnya dengan pustaka, aplikasi juga menggunakan pola penamaan yang sama. Letakkan `hellostringutil.go` di `src/github.com/oldstager/hellostringutil`.



Untuk mengkompilasi dan menjalankan:

```
$ go install ...
$ hellostringutil
!dlroW ,olleH
$ ls -la bin/
total 1976
drwxr-xr-x 2 bdp bdp 4096 Jul 29 21:46 ./
drwxr-xr-x 5 bdp bdp 4096 Jul 29 21:26 ../
-rwxr-xr-x 1 bdp bdp 2014199 Jul 29 21:46 hellostringutil*
$
```

### 3.3. Menggunakan Modules

Penggunaan **modules** lebih disarankan untuk proses pengembangan berikutnya. Saat menggunakan Go 1.11 dan Go 1.12, kita masih berada pada masa transisi. Meskipun demikian, saat **modules** telah diimplementasikan penuh di Go 1.13, tidak akan mengacaukan kode sumber dengan **modules** yang dibuat menggunakan Go 1.11 dan Go 1.12.

#### 3.3.1. Program Aplikasi

Untuk keperluan ini, buat direktori (lokasi bebas - di luar \$GOPATH). Pada direktori tersebut, inisialisasi **modules** terlebih dahulu menggunakan:

```
$ go mod init github.com/oldstager/go-to-hell-o
$ cat go.mod
module github.com/oldstager/go-to-hell-o

go 1.12
$
```

Setelah itu baru buat kode sumber **hellorsc.go** pada direktori tersebut. Untuk mengkompilasi:

```
$ go build
go: finding rsc.io/quote v1.5.2
go: downloading rsc.io/quote v1.5.2
go: extracting rsc.io/quote v1.5.2
go: finding rsc.io/sampler v1.3.0
go: finding golang.org/x/text v0.0.0-20170915032832-14c0d48ead0c
go: downloading rsc.io/sampler v1.3.0
go: extracting rsc.io/sampler v1.3.0
go: downloading golang.org/x/text v0.0.0-20170915032832-14c0d48ead0c
go: extracting golang.org/x/text v0.0.0-20170915032832-14c0d48ead0c
$
```

Hasil:

```
$ ls -la
total 2296
drwxr-xr-x 2 bdp bdp 4096 Jul 28 06:55 ./
drwxr-xr-x 4 bdp bdp 4096 Jul 28 06:45 ../
-rw----- 1 bdp bdp 79 Jul 28 06:55 go.mod
-rw----- 1 bdp bdp 499 Jul 28 06:55 go.sum
-rwxr-xr-x 1 bdp bdp 2327743 Jul 28 06:55 go-to-hell-o*
-rw-r--r-- 1 bdp bdp 93 Jul 28 06:54 hello.go
$
```

Module yang sudah di-**get** dan di-**build** berada di:

```

$ tree -L 3 ~/go/pkg/mod/
/home/bpdp/go/pkg/mod/
├── cache
│   ├── download
│   │   ├── golang.org
│   │   └── rsc.io
│   ├── lock
│   └── vcs
├── 0c8659d2f971b567bc9bd6644073413a1534735b75ea8a6f1d4ee4121f78fa5b
│   ├── 0c8659d2f971b567bc9bd6644073413a1534735b75ea8a6f1d4ee4121f78fa5b.info
│   └── 0c8659d2f971b567bc9bd6644073413a1534735b75ea8a6f1d4ee4121f78fa5b.lock
├── 4db0c9594744360b0eaa452d2ccfbd45b05dfffb9810882957d10d69e61e66382
│   ├── 4db0c9594744360b0eaa452d2ccfbd45b05dfffb9810882957d10d69e61e66382.info
│   └── 4db0c9594744360b0eaa452d2ccfbd45b05dfffb9810882957d10d69e61e66382.lock
├── 5b03666c2d7b526129bad48c5cea095aad8b83badc1daa202e7b0279e3a5d861
│   ├── 5b03666c2d7b526129bad48c5cea095aad8b83badc1daa202e7b0279e3a5d861.info
│   └── 5b03666c2d7b526129bad48c5cea095aad8b83badc1daa202e7b0279e3a5d861.lock
├── golang.org
│   └── x
│       └── text@v0.0.0-20170915032832-14c0d48ead0c
└── rsc.io
    ├── quote@v1.5.2
    │   ├── buggy
    │   ├── go.mod
    │   ├── LICENSE
    │   ├── quote.go
    │   ├── quote_test.go
    │   └── README.md
    └── sampler@v1.3.0
        ├── glass.go
        ├── glass_test.go
        ├── go.mod
        ├── hello.go
        ├── hello_test.go
        ├── LICENSE
        └── sampler.go

```

15 directories, 19 files

\$

### 3.3.2. Pustaka dan Penggunaannya

Penggunaan **modules** menyebabkan proses pengembangan menjadi lebih sederhana. Untuk contoh ini, kita akan membuat 2 proyek:

1. **gomtk**: berisi pustaka matematika - saat ini hanya berisi 1 function, yaitu **Add**.
2. **gomtktest**: berisi program aplikasi yang akan memanggil pustaka **gomtk**.

Kedua direktori tersebut bisa berada di mana saja.

#### Pustaka

Direktori untuk pustaka ini diinisialisasi sebagai **module** dengan cara:

```
$ go mod init github.com/oldstager/gomtk
$ cat go.mod
module github.com/oldstager/gomtk

go 1.12
$
```

Untuk mengkompilasi:

```
$ go build
$
```

#### Penggunaan Pustaka

Direktori untuk penggunaan pustaka ini diinisialisasi sebagai **module** dengan cara:

```
$ go mod init github.com/oldstager/gomtktest
$
```

Aplikasi ini menggunakan pustaka, sehingga **go.mod** harus diedit:

```
module github.com/oldstager/gomtktest  
  
go 1.12  
  
replace github.com/oldstager/gomtk => ../gomtk  
$
```

Untuk mengkompilasi menjadi **binary executable**:

```
$ go build  
$ ls -la  
total 1976  
drwxr-xr-x 2 bdp bdp 4096 Jul 29 22:43 ./  
drwxr-xr-x 4 bdp bdp 4096 Jul 28 23:43 ../  
-rw----- 1 bdp bdp 166 Jul 28 23:45 go.mod  
-rwxr-xr-x 1 bdp bdp 2005767 Jul 29 22:43 gomtktest*  
-rw-r--r-- 1 bdp bdp 115 Jul 28 23:45 gomtktest.go  
$
```

## Part II: Sintaksis Go

Bagian ini membahas tentang sintaks dari bahasa pemrograman Go.

## Bab 4. Sintaksis Dasar Go

### 4.1. Komentar

Bagian komentar dimaksudkan untuk dokumentasi dari **source code**. Ada beberapa cara untuk memberikan komentar:

- Menggunakan `/* ... */` untuk komentar yang meliputi lebih dari satu baris
- Menggunakan `//` di awal baris untuk komentar yang meliputi satu baris saja
- Menggunakan `//` di suatu baris untuk komentar mulai dari sebelah kana `//` sampai ke akhir baris.

Komentar ini sejak awal sebaiknya sudah dibiasakan harus ada karena Go menyediakan fasilitas `godoc` untuk menghasilkan dokumentasi dari **source code**. Bagian yang sebaiknya diberikan komentar / dokumentasi adalah bagian diatas `package` dan di atas setiap definisi fungsi (lihat contoh dari `stringutil` di atas.

### 4.2. Variabel

Variabel (bahasa inggris: **variable**) merupakan nama yang digunakan untuk menyimpan data. Data yang disimpan mempunyai tipe. Kata kunci yang digunakan untuk mendeklarasikan variabel adalah **var**:

```
...
...
    var namaVariabel tipe = isiData
// bisa juga:
    var namaVariabel tipe
// atau bisa juga menggunakan type inferencing:
    namaVariabel := isiData
    var namaVariabel = isiData
...
...
```

Pada dasarnya Go tidak peduli nama variabel (`x` atau `namaSiswa` mempunyai efek yang sama), tetapi praktik yang baik untuk Go adalah dengan menggunakan nama yang bermakna dengan pola yang disebut dengan **lower camel case** (atau **mixed case**, **bumpy caps**, **camel back**, **hump back**) berikut ini:

```

...
...
    var namaPerusahaan string
...
...

```

## 4.3. Tipe Data Dasar

Data di Go mempunyai tipe. Go termasuk dalam kategori **static typing** (tipe data akan diperiksa saat proses kompilasi dan tidak boleh ada perubahan tipe) dan **strongly-typed** (tipe data tidak bisa berubah secara implisit pada konteks tertentu, perubahan hanya bisa dilakukan jika eksplisit dilakukan **casting**).

### 4.3.1. Tipe Data Angka / Numerik

Untuk tipe numerik, pada dasarnya kita bisa menggunakan bilangan bulat (**integer**) dan bilangan pecahan (**floating-point**). Bilangan bulat terdiri atas bilangan bertanda (**signed** - int) dan bilangan tak-bertanda (**unsigned** - uint). Berikut ini adalah daftar lengkap dari tipe data numerik tersebut:

Tipe	Arti	Jangkauan
uint8	unsigned 8-bit integer	0 sampai 255
uint16	unsigned 16-bit integer	0 sampai 65535
uint32	unsigned 32-bit integer	0 sampai 4294967295
uint64	unsigned 64-bit integer	0 sampai 18446744073709551615
int8	signed 8-bit integer	-128 sampai 127
int16	signed 16-bit integer	-32768 sampai 32767
int32	signed 32-bit integer	-2147483648 sampai 2147483647
int64	signed 64-bit integer	-9223372036854775808 sampai 9223372036854775807
float32	IEEE-754 32-bit floating-point	



float64	IEEE-754 64-bit floating-point	
complex64	bilangan kompleks dengan float32 riil dan imajiner	~
complex128	bilangan kompleks dengan float64 riil dan imajiner	~
byte	alias dari uint8	
rune	alias dari int32	

Go tidak mempunyai tipe data karakter, sebagai gantinya, **byte** digunakan untuk merepresentasikan karakter ASCII, sedangkan **rune** digunakan untuk karakter **Unicode** UTF-8:

```
/*
diambil dari: https://www.callicoder.com/golang-basic-types-operators-
type-conversion/
*/
package main

import "fmt"

func main() {
    var myByte byte = 'a'
    var myRune rune = 'a'

    fmt.Printf("%c = %d and %c = %U\n", myByte, myByte, myRune, myRune)
}
```

Hasil:

```
$ go run char.go
a = 97 and a = U+2665
$
```

Selain definisi di atas, Go juga mempunyai alias penyebutan yang implementasinya tergantung pada arsitektur komputer yang digunakan:

Tipe	Arti
uint	arsitektur 32 atau 64 bit

int	mempunyai ukuran yang sama dengan uint
uintptr	bilangan bulat tak bertanda untuk menyimpan nilai pointer

Go sebagai bahasa pemrograman **static-typing** dan **strongly-typed** bisa dilihat pada contoh berikut:

```
package main

import (
    "fmt"
)

var (
    angka1    uint8  = 21
    angka2    uint8  = 17
    angkaFloat float64 = 7.1
)

func main() {
    // ./typecast.go:14:11: cannot use "abc" (type string) as type uint8
    in assignment
    //angka1 = "abc"
    fmt.Println(angka1 + angka2)
    // ./typecast.go:15:21: invalid operation: angka1 + angkaFloat
    (mismatched types uint8 and float64)
    //fmt.Println(angka1 + angkaFloat)
    fmt.Println(float64(angka1) + angkaFloat)
}
```

Hasil:

```
$ go run typecast.go
38
28.1
$
```

### 4.3.2. String

String digunakan untuk tipe data berupa sekumpulan huruf / karakter.

```

package main

import (
    "fmt"
    "reflect"
    s "strings"
)

// Definisi string
var str1 string = "UGM"
var str2 = "Yogyakarta"
var str3 = "universitas\ngadjah mada"

var str3backtick = `universitas\ngadjah mada`

// error: illegal rune literal
//var str3singlequoted = 'universitas gadjah mada'

func main() {

    // Lihat https://golang.org/pkg/strings/
    fmt.Println(str1)
    fmt.Println(len(str1))
    fmt.Println(s.Contains(str1, "GM"))
    fmt.Println(s.Title(str3))
    fmt.Println(str1[0])
    fmt.Println(s.Join([]string{str1, str2}, " "))
    fmt.Println(s.Join([]string{str3, str2}, "\n"))
    fmt.Println(s.Join([]string{str3backtick, str2}, "\n"))
    fmt.Println(reflect.TypeOf(str1))
    fmt.Println(reflect.TypeOf(str2))
    fmt.Println()

}

```

Hasil:

```
$ go run varstring.go
UGM
3
true
Universitas
Gadjah Mada
85
UGM Yogyakarta
universitas
gadjah mada
Yogyakarta
universitas\ngadjah mada
Yogyakarta
string
string
$
```

### 4.3.3. Boolean

Tipe data Boolean berisi nilai benar (**true**) atau salah (**false**).

```
package main

import (
    "fmt"
    "reflect"
)

var (
    hasilPerbandingan bool
    angka1             uint8 = 21
    angka2             uint8 = 17
)

func main() {
    hasilPerbandingan = angka1 < angka2
    fmt.Printf("angka1 = %d\n", angka1)
    fmt.Printf("angka2 = %d\n", angka2)
    fmt.Println(reflect.TypeOf(hasilPerbandingan))
    fmt.Println(hasilPerbandingan)
}
```

Hasil:

```
$ go run varboolean.go
angka1 = 21
angka2 = 17
bool
false
$
```

## 4.4. Nilai Default Variabel

Setiap variabel yang dideklarasikan dan tidak di-**assign** isi data tertentu akan mempunyai nilai default.

```
// nilai-default-variabel.go
package main

import "fmt"

func main() {

    // unsigned-integer
    var defUint8 uint8
    var defUint16 uint16
    var defUint32 uint32
    var defUint64 uint64
    var defUint uint

    // signed-integer
    var defInt8 int8
    var defInt16 int16
    var defInt32 int32
    var defInt64 int64
    var defInt int

    // string
    var defString string

    // floating-point
    var defFloat32 float32
    var defFloat64 float64

    // complex
    var defComplex64 complex64
    var defComplex128 complex128

    // alias
    var defByte byte
    var defRune rune
}
```

```

fmt.Println("\nNilai default untuk uint8 = ", defUint8)
fmt.Println("Nilai default untuk uint16 = ", defUint16)
fmt.Println("Nilai default untuk uint32 = ", defUint32)
fmt.Println("Nilai default untuk uint64 = ", defUint64)
fmt.Println("Nilai default untuk uint = ", defUint)

fmt.Println("\nNilai default untuk int8 = ", defInt8)
fmt.Println("Nilai default untuk int16 = ", defInt16)
fmt.Println("Nilai default untuk int32 = ", defInt32)
fmt.Println("Nilai default untuk int63 = ", defInt64)
fmt.Println("Nilai default untuk int = ", defInt)

fmt.Println("\nNilai default untuk string = ", defString)

fmt.Println("\nNilai default untuk float32 = ", defFloat32)
fmt.Println("Nilai default untuk float64 = ", defFloat64)

fmt.Println("\nNilai default untuk complex64 = ", defComplex64)
fmt.Println("Nilai default untuk complex128 = ", defComplex128)

fmt.Println("\nNilai default untuk byte = ", defByte)
fmt.Println("Nilai default untuk rune = ", defRune)

}

```

Hasil eksekusi:

```
$ go run nilai-default-variabel.go

Nilai default untuk uint8 = 0
Nilai default untuk uint16 = 0
Nilai default untuk uint32 = 0
Nilai default untuk uint64 = 0
Nilai default untuk uint = 0

Nilai default untuk int8 = 0
Nilai default untuk int16 = 0
Nilai default untuk int32 = 0
Nilai default untuk int64 = 0
Nilai default untuk int = 0

Nilai default untuk string =

Nilai default untuk float32 = 0
Nilai default untuk float64 = 0

Nilai default untuk complex64 = (0+0i)
Nilai default untuk complex128 = (0+0i)

Nilai default untuk byte = 0
Nilai default untuk rune = 0
$
```

## 4.5. Operator

Operator merupakan simbol yang digunakan sebagai penunjuk bagi **compiler** untuk melaksanakan operasi tertentu terhadap data. Operator di Go secara umum adalah sebagai berikut:

### 4.5.1. Aritmatika

Operator	Deskripsi
+	Penambahan
-	Pengurangan
*	Perkalian
/	Pembagian
%	Sisa hasil bagi

<code>&amp;</code>	bitwise AND
<code> </code>	bitwise OR
<code>^</code>	bitwise XOR
<code>&amp;^</code>	bitclear (AND NOT)
<code>&lt;&lt;</code>	left shift
<code>&gt;&gt;</code>	right shift

#### 4.5.2. Perbandingan

Operator	Deskripsi
<code>==</code>	sama dengan
<code>!=</code>	tidak sama dengan
<code>&lt;</code>	lebih kecil daripada
<code>&lt;=</code>	lebih kecil atau sama dengan
<code>&gt;</code>	lebih besar daripada
<code>&gt;=</code>	lebih besar atau sama dengan

#### 4.5.3. Logika

Operator	Deskripsi
<code>&amp;&amp;</code>	AND
<code>  </code>	OR
<code>!</code>	NOT

#### 4.5.4. Lain-lain (pointer dan channel)

Operator	Deskripsi
<code>&amp;</code>	alamat pointer
<code>*</code>	de-referensi pointer
<code>←</code>	send / receive untuk channel



Bagian yang biasanya dirasakan cukup rumit adalah operasi bit. Untuk mengetahui lebih lanjut tentang operasi bit, <https://medium.com/learning-the-go-programming-language/bit-hacking-with-go-e0acee258827> menyediakan informasi serta contoh kode yang cukup lengkap.

## 4.6. Konstanta

Konstanta dimaksudkan untuk menampung data yang tidak akan berubah-ubah. Konstanta dideklarasikan menggunakan kata kunci **const**. Konstant bisa bertipe **character**, string, boolean, atau numerik.

```
package main

import (
    "fmt"
)

func main() {

    const mainCodingLang = "Go"
    const kiamatMakinDekat = true

    const angka1, angka2 = 25, 8

    const (
        nomorPegawai = "P001"
        gaji          = 50000000
    )

    const negaraKu string = "Indonesia"

    const gajiBersihSetelahSetorIstri = gaji - 49000000

    fmt.Println(mainCodingLang)
    fmt.Println(kiamatMakinDekat)
    fmt.Println(angka1)
    fmt.Println(angka2)
    fmt.Println(nomorPegawai)
    fmt.Println(gaji)
    fmt.Println(negaraKu)
    fmt.Println(gajiBersihSetelahSetorIstri)

    // ./konstanta.go:28:7: cannot assign to gaji
    //gaji = 10000000

}
```

Hasil:

```
$ go run konstanta.go
Go
true
25
8
P001
50000000
Indonesia
1000000
$
```

## 4.7. Pointer

Konsep **pointer** sebenarnya sudah ada pada bahasa pemrograman lain, khususnya C/C++ (dengan kompleksitas yang lebih tinggi). Suatu **pointer** menyimpan **memory address** dari suatu nilai. Di Go, `&` menunjukkan **memory address** suatu variabel, sementara `*` menunjukkan isi dari memory yang ditunjukkan oleh pointer tersebut (disebut juga dereferensi). Pointer ini sangat bermanfaat terutama jika berkaitan dengan manipulasi `*function*`. Untuk saat ini, batasi pemahaman pada operator dasar pointer, pembahasan lebih lanjut ada pada pembahasan tentang **function**.

```

package main

import "fmt"

func main() {
    i, j := 42, 2701

    // p berisi memory address dari i
    p := &i
    // tampilkan isi dari memory address p
    fmt.Println(*p)
    // isi dari memory address yang ditunjuk p diubah
    *p = 21
    // implikasinya pada variabel i:
    fmt.Println(i)

    // p berisi memory address dari j
    p = &j
    // isi dari memory address yang ditunjuk p, diubah
    // menjadi isi memoery address yang lama, dibagi 37
    *p = *p / 37
    // implikasinya pada variabel j
    fmt.Println(j) // see the new value of j

    var pa *int

    fmt.Printf("pointer pa dengan tipe %T dan nilai %v\n", pa, pa)
}

```

Hasil:

```

$ go run pointer.go
42
21
73
pointer pa dengan tipe *int dan nilai <nil>
$

```

## 4.8. Struktur Kendali

Saat membuat kode sumber, seringkali ada beberapa bagian dari program yang harus kita kendalikan (dilakukan perulangan, mengambil keputusan, dan sejenisnya).

### 4.8.1. Seleksi Kondisi

Bagian ini digunakan dalam hal terdapat kondisi tertentu dan akan dilakukan suatu tindakan berdasarkan kondisi tertentu tersebut.

#### Pernyataan **if** dan macam-macam penggunaannya

```
if boolean_expression {  
    // ...  
    // dieksekusi jika boolean_expression bernilai true  
}
```

Contoh penggunaan:

```
package main  
  
import "fmt"  
  
func main() {  
    var a int = 10  
  
    if a < 20 {  
        fmt.Println("a < 20")  
    }  
}
```

Hasil:

```
$ go run if1.go  
a < 20  
$
```

Pernyataan **if** juga bisa meliputi kondisi yang lebih kompleks:

```

package main

import "fmt"

func main() {
    var a int = 200

    if a == 10 {
        fmt.Printf("Nilai a = 10\n")
    } else if a == 20 {
        fmt.Printf("Nilai a = 20\n")
    } else if a == 30 {
        fmt.Printf("Nilai a = 30\n")
    } else {
        fmt.Printf("Semua nilai salah\n")
    }
    fmt.Printf("Nilai dari a adalah: %d\n", a)
}

```

Hasil:

```

$ go run if2.go
Semua nilai salah
Nilai dari a adalah: 200
$

```

**Pernyataan switch**

```

package main

import "fmt"

func main() {

    var nilaiAngka int = 20
    var nilaiHuruf string

    switch nilaiAngka {
    case 90:
        nilaiHuruf = "A"
    case 80:
        nilaiHuruf = "B"
    case 50, 60, 70:
        nilaiHuruf = "C"
    default:
        nilaiHuruf = "D"
    }

    switch {
    case nilaiHuruf == "A":
        fmt.Println("Apik tenan!")
    case nilaiHuruf == "B":
        fmt.Println("Lumayan lah")
    case nilaiHuruf == "C", nilaiHuruf == "D":
        fmt.Println("Lulus sih ... tapi ..")
    case nilaiHuruf == "E":
        fmt.Println("Nangis bombay")
    default:
        fmt.Println("Nilai gak jelas, seperti wajah dosennya!")
    }
    fmt.Printf("Nilai anda = %s\n", nilaiHuruf)
}

```

Hasil:

```

$ go run switch.go
Lulus sih ... tapi ..
Nilai anda = D
$

```

#### 4.8.2. Perulangan dengan **for**

Perulangan atau **looping** menggunakan **for** adalah perulangan yang bisa kita definisikan ketentuan jumlah perulangannya. Sintaksis dari **for** adalah sebagai berikut:

```
for [condition | ( init; condition; increment ) | Range] {  
    statement(s);  
}
```

Sintaksis dari **for** ini juga memungkinkan dilakukan secara **nested** atau bertingkat.

```
package main  
  
import "fmt"  
  
func main() {  
    var i, j int  
  
    for i = 1; i < 10; i++ {  
        fmt.Println(i)  
        for j = 1; j <= i; j++ {  
            fmt.Println(j)  
        }  
    }  
}
```

Hasil:

```
go run loopfor.go  
1  
1  
2  
1  
2  
3  
1  
2  
3  
4  
1  
2  
3  
4  
5  
1  
2  
3  
4  
5  
6  
1  
2
```

```
3
4
5
6
7
1
2
3
4
5
6
7
8
1
2
3
4
5
6
7
8
9
1
2
3
4
5
6
7
8
9
```

Pada kondisi tertentu, dimungkinkan untuk menghentikan perulangan menggunakan **break** atau meneruskan ke perulangan berikutnya menggunakan **continue**.



```

package main

import "fmt"

func main() {
    var a int = 10

    for a < 20 {
        if a == 12 {
            a += 1
            continue
        }
        a++
        if a > 15 {
            break
        }
        fmt.Printf("Nilai a: %d\n", a)
    }
}

```

Hasil:

```

$ go run ifcontinuebreak.go
Nilai a: 11
Nilai a: 12
Nilai a: 14
Nilai a: 15
$

```

### 4.8.3. Defer

Defer digunakan untuk mengeksekusi suatu perintah sebelum suatu fungsi berakhir. Jika berada pada suatu fungsi, baris kode sumber yang di-defer akan dikerjakan sebelum menemui akhir (**return**). Kegunaan utama dari **defer** ini adalah untuk keperluan pembersihan (**cleanup**). Saat kita membuat kode sumber Go, sering kali dalam setiap operasi terdapat beberapa hal yang harus kita akhiri dengan kondisi tertentu, misalnya jika kita membuka file maka kita harus menutup file jika kita sudah selesai melakukan operasi dengan file tersebut. **Defer** mempermudah kita untuk memastikan bahwa pekerjaan-pekerjaan pembersihan tersebut selalu bisa dilakukan.

```
package main

import "fmt"

func main() {
    defer fmt.Println("world")

    fmt.Println("hello")
}
```

Hasil:

```
$ go run defer.go
hello
world
$
```

## Bab 5. Fungsi

Fungsi merupakan bagian dari kode sumber yang dimaksudkan untuk mengerjakan sesuatu hal. Hal yang dikerjakan tersebut biasanya merupakan suatu hal yang sifatnya cukup umum sehingga terdapat kemungkinan dalam kode sumber bisa digunakan berkali-kali. Fungsi dibuat supaya tidak perlu mengkode ulang pekerjaan tersebut. Jika diperlukan pada suatu kode, bagian tersebut tinggal memanggil fungsi. Untuk mengerjakan pekerjaan tersebut, fungsi biasanya memerlukan data masukan (sering disebut dengan **argumen** atau **parameter**). Setelah mengerjakan fungsi tersebut, fungsi biasanya menghasilkan suatu nilai (sering disebut dengan istilah **return value** / nilai kembalian). Kode sumber Go yang dimaksudkan untuk menghasilkan **binary executable** mempunyai satu fungsi yang akan dikerjakan saat kode tersebut dikompilasi dan dieksekusi, yaitu fungsi **main()** (lihat bab 2).

### 5.1. Deklarasi Fungsi

Fungsi dibuat dengan menggunakan kata kunci **func**, diikuti nama, argumen, serta tipe nilai kembalian. Berikut ini adalah contoh dari fungsi serta pemakaiannya.

```
package main

import "fmt"

func findSumOfChars(strCounted string, theChar rune) int {
    counter := 0
    for _, c := range strCounted {
        if c == theChar {
            counter++
        }
    }
    return counter
}

func main() {
    theStr := "STMIK Akakom"
    fmt.Printf("Jumlah karakter 'k' dari string %s adalah %d",
        theStr, findSumOfChars(theStr, 'k'))
}
```

Hasil:

```
$ go run function-01.go
Jumlah karakter 'k' dari string STMIK Akakom adalah 2
$
```

## 5.2. Fungsi dengan Banyak Nilai Kembalian

Berbeda dengan kebanyakan bahasa pemrograman lain yang mengembalikan hanya 1 nilai kembalian dari fungsi, Go menyediakan fasilitas untuk memberikan banyak nilai kembalian. Berikut ini adalah contohnya.

```
package main

import "fmt"

func doMath(a, b float64) (float64, float64, float64, float64) {
    return a * b, a / b, a + b, a - b
}

func main() {
    a, b, c, d := doMath(4, 10)

    fmt.Println(a)
    fmt.Println(b)
    fmt.Println(c)
    fmt.Println(d)

    _, nil1, _, _ := doMath(12, 21)

    fmt.Println(nil1)
}
```

Hasil:

```
$ go run f-multi-retval.go
40
0.4
14
-6
0.5714285714285714
```

## 5.3. Variadic Function

**Variadic function** adalah fungsi yang jumlah argumennya sembarang dan belum pasti. Untuk menangani kondisi seperti ini, kita bisa menggunakan **pack type** (titik 3: ...).

```
package main

import "fmt"
import "strings"

func combineStr(theStr ...string) string {
    return strings.Join(theStr, " ")
}

func doMath(theOperator rune, theNums ...float64) float64 {
    counter := 0.0
    if theOperator == '+' {
        for _, theVal := range theNums {
            counter += theVal
        }
    }
    return counter
}

func main() {
    str1 := "String1"
    str2 := "String2"

    fmt.Println(combineStr(str1, str2))

    fmt.Println(doMath('+', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
}
```

Hasilnya:

```
$ go run variadic-f.go
String1 String2
55
$
```

## 5.4. Anonymous Functiona / Lambda Abstraction

Fungsi juga bisa tanpa diberi nama. Biasanya hal seperti ini diperlukan jika ada satu block kode yang perlu kita "evaluasi dan lupakan". Setelah evaluasi dan mengerjakan fungsi, GC (**garbage collector**) akan membersihkan memory yang digunakan.

```
package main

import "fmt"

func main() {

    func(angka1, angka2 float64) {
        fmt.Println(angka1 + angka2)
    }(10.0, 20.5)

}
```

Hasilnya:

```
$ go run lambda.go
30.5
$
```

## 5.5. Closures

Closures adalah bentuk khusus dari abstraksi lambda yang mengambil referensi variabel dari luar definisi fungsi. Contoh:

```
// Diambil dari: https://www.calhoun.io/what-is-a-closure/
package main

import "fmt"

func main() {
    n := 0
    counter := func() int {
        n += 1
        return n
    }
    fmt.Println(counter())
    fmt.Println(counter())
}
```

Hasilnya:

```
$ go run closures.go
1
2
$
```

## 5.6. Fungsi Rekursif

Fungsi rekursif adalah fungsi yang memanggil dirinya sendiri.

```
package main

import "fmt"

func factorial(i int) int {
    if i <= 1 {
        return 1
    }
    return i * factorial(i-1)
}

func main() {
    var i int = 15
    fmt.Printf("Factorial dari %d is %d", i, factorial(i))
}
```

Hasilnya:

```
$ go run rekursif-factorial.go
Factorial of 15 is 1307674368000
$
```

## 5.7. Call by Value

Saat memberikan argumen dari suatu fungsi, Go membuat salinan baru dari variabel tersebut. Dengan demikian, pemanggilan fungsi tidak akan menyebabkan variabel terpengaruh.

```
package main

import "fmt"

func changeMe(theVal int) int {
    theVal++
    return theVal
}

func main() {
    theVal := 45
    fmt.Println(theVal)
    fmt.Println(changeMe(theVal))
    fmt.Println(theVal)
}
```

Hasilnya:

```
$ go run func-pass-by-val.go
45
46
45
$
```



## 5.8. Call by Pointer

Jika suatu variabel dimaksudkan untuk dimanipulasi di dalam badan fungsi, maka pemanggilan fungsi tersebut harus menggunakan pemanggilan pointer.

```
package main

import "fmt"

var theVal int = 0

func changeMe(theVal *int) int {
    *theVal = 150
    return *theVal
}

func main() {
    theVal := 45
    fmt.Println(theVal)
    fmt.Println(changeMe(&theVal))
    fmt.Println(theVal)
}
```

Hasilnya:

```
$ go run func-pass-by-pointer.go
45
150
150
$
```

## Bab 6. Penanganan Kesalahan

### 6.1. Penggunaan error

Saat terjadi kesalahan, Go menyediakan fasilitas pada pustaka standar `errors`. Berikut adalah contohnya:

```
package main

import "fmt"
import "errors"

func bagi(angka1, angka2 float64) (float64, error) {
    if angka2 == 0 {
        return -1, errors.New("Pembagian dengan angka 0 dilarang")
    }
    return angka1 / angka2, nil
}

func main() {

    var hasilBagi, err = bagi(23, 3)
    fmt.Println(hasilBagi, err)

    var hasilBagi2, err2 = bagi(23, 0)
    fmt.Println(hasilBagi2, err2)

}
```

Hasilnya:

```
$ go run error.go
7.666666666666667 <nil>
-1 Pembagian dengan angka 0 dilarang
$
```

### 6.2. Panic dan Recover

Go menyediakan konstruksi **panic** dan **recover** untuk menangani kesalahan yang tidak bisa "ditolerir". Sebagai contoh, jika aplikasi kita mutlak memerlukan suatu file konfigurasi dan file konfigurasi tersebut tidak ada, maka kita bisa menggunakan **panic** untuk menghentikan eksekusi aplikasi dan kemudian memberikan pesan kesalahan. Jika kita hanya menggunakan **panic**, maka

program akan berakhir dengan pesan error serta dump dari instruksi biner yang menyebabkan error tersebut. Hal ini seringkali tidak dikehendaki sehingga diperlukan **recover** untuk mengakhiri program dengan baik.

```

// slightly modified from:
// https://blog.golang.org/defer-panic-and-recover
package main

import "fmt"

func main() {
    f()
    f2()
    fmt.Println("Returned normally from f.")
}

func f() {
    // jika ada panic, maka defer akan dipanggil
    defer func() {
        // mengambil nilai revocer, jika tidak nil (karena panic)
        // maka pada bagian ini akan dilakukan berbagai hal untuk
        // menangani panic tersebut
        if r := recover(); r != nil {
            fmt.Println("Recovered in f", r)
        }
    }()
    fmt.Println("Calling g.")
    g(0)
    fmt.Println("Returned normally from g.")
}

func f2() {
    fmt.Println("Calling g.")
    g(0)
    fmt.Println("Will not be executed because of panic")
}

func g(i int) {
    if i > 3 {
        fmt.Println("Panicking!")
        panic(fmt.Sprintf("%v", i))
    }
    defer fmt.Println("Defer in g", i)
    fmt.Println("Printing in g", i)
    // rekursif
    g(i + 1)
}

```

Hasilnya:

```

$ go run panic-recover.go
Calling g.
Printing in g 0
Printing in g 1
Printing in g 2
Printing in g 3
Panicking!
Defer in g 3
Defer in g 2
Defer in g 1
Defer in g 0
Recovered in f 4
Calling g.
Printing in g 0
Printing in g 1
Printing in g 2
Printing in g 3
Panicking!
Defer in g 3
Defer in g 2
Defer in g 1
Defer in g 0
panic: 4

goroutine 1 [running]:
main.g(0x4)
    /home/bpdp/kerjaan/git-repos/oldstager/current/github/oldstager/buku-
go/src/bab-07/panic-recover.go:41 +0x28a
main.g(0x3)
    /home/bpdp/kerjaan/git-repos/oldstager/current/github/oldstager/buku-
go/src/bab-07/panic-recover.go:46 +0x16f
main.g(0x2)
    /home/bpdp/kerjaan/git-repos/oldstager/current/github/oldstager/buku-
go/src/bab-07/panic-recover.go:46 +0x16f
main.g(0x1)
    /home/bpdp/kerjaan/git-repos/oldstager/current/github/oldstager/buku-
go/src/bab-07/panic-recover.go:46 +0x16f
main.g(0x0)
    /home/bpdp/kerjaan/git-repos/oldstager/current/github/oldstager/buku-
go/src/bab-07/panic-recover.go:46 +0x16f
main.f2()
    /home/bpdp/kerjaan/git-repos/oldstager/current/github/oldstager/buku-
go/src/bab-07/panic-recover.go:33 +0x87
main.main()
    /home/bpdp/kerjaan/git-repos/oldstager/current/github/oldstager/buku-
go/src/bab-07/panic-recover.go:9 +0x27
exit status 2
$

```

## Bab 7. Struktur Data

### 7.1. Struct

**Struct** merupakan struktur data di Go yang digunakan untuk menampung koleksi **fields**. Mirip dengan DBMS, struct juga mengelola data dalam bentuk tabel: ada baris dan ada kolom. Kolom disebut dengan **field** sementara baris disebut dengan **record** dengan isi hanya 1. Berikut adalah contoh **struct**:

```
package main

import "fmt"

type Pegawai struct {
    Nama string
    Usia int
}

func main() {

    var pegawaiKantor1 Pegawai
    fmt.Println(pegawaiKantor1)

    var pegawaiKantor2 = Pegawai{"Peg kantor 2 - 1", 24}
    fmt.Println(pegawaiKantor2)

    var pegawaiKantor3 = Pegawai{Nama: "Peg kantor 3 - 1", Usia: 54}
    fmt.Println(pegawaiKantor3)

    pegawaiKantor4 := Pegawai{
        Nama: "Peg kantor 4 - 1",
        Usia: 54,
    }
    fmt.Println(pegawaiKantor4)
    fmt.Println(pegawaiKantor4.Nama)
    fmt.Println(pegawaiKantor4.Usia)
    pegawaiKantor4.Nama = "Peg kantor 4 - baru"
    fmt.Println(pegawaiKantor4)

    var pPegawaiKantor4 = &pegawaiKantor4
    var p2PegawaiKantor4 = &pegawaiKantor4

    fmt.Println(*pPegawaiKantor4)
    fmt.Println(*p2PegawaiKantor4)

    pegawaiKantor4.Nama = "Pengganti pegawai kantor 4"
```

```

    fmt.Println(*pPegawaiKantor4)
    fmt.Println(*p2PegawaiKantor4)
    fmt.Println((*p2PegawaiKantor4).Nama)

    *pPegawaiKantor4 = Pegawai{Nama: "Pengganti lagi utk kantor 4", Usia:
25}

    fmt.Println(*pPegawaiKantor4)
    fmt.Println(*p2PegawaiKantor4)
    fmt.Println((*p2PegawaiKantor4).Nama)

    // pointer ke struct dengan new:
    pPeg := new(Pegawai)

    pPeg.Nama = "pPeg 1"
    pPeg.Usia = 21

    fmt.Println(pPeg)
    fmt.Println(*pPeg)

}

```

Hasil:

```

$ go run struct.go
{ 0}
{Peg kantor 2 - 1 24}
{Peg kantor 3 - 1 54}
{Peg kantor 4 - 1 54}
Peg kantor 4 - 1
54
{Peg kantor 4 - baru 54}
{Peg kantor 4 - baru 54}
{Peg kantor 4 - baru 54}
{Pengganti pegawai kantor 4 54}
{Pengganti pegawai kantor 4 54}
Pengganti pegawai kantor 4
{Pengganti lagi utk kantor 4 25}
{Pengganti lagi utk kantor 4 25}
Pengganti lagi utk kantor 4
&{pPeg 1 21}
{pPeg 1 21}
$

```

## 7.2. Method

**Method** sering disebut juga **receiver function**. Pada dasarnya, **method** juga merupakan fungsi,

tetapi khusus untuk struktur data **struct**.

```
package main

import "fmt"

type Pegawai struct {
    Nama string
    Usia int
}

func (p Pegawai) print() {
    fmt.Println("Nama: ", p.Nama, "Usia: ", p.Usia)
}

func (p *Pegawai) printPointer() {
    fmt.Println("Nama: ", p.Nama, "Usia: ", p.Usia)
}

func main() {
    var pegawaiKantor2 = Pegawai{"Peg kantor 2 - 1", 24}
    pegawaiKantor2.print()
    pPeg := &pegawaiKantor2
    pPeg.printPointer()
}
```

Hasilnya:

```
$ go run method.go
Nama: Peg kantor 2 - 1 Usia: 24
Nama: Peg kantor 2 - 1 Usia: 24
$
```

## 7.3. Interfaces

Pada bahasa pemrograman Go, **interfaces** digunakan untuk mengelompokkan koleksi method.



```
// https://jordanorelli.com/post/32665860244/how-to-use-interfaces-in-go
package main

import (
    "fmt"
)

type Animal interface {
    Speak() string
}

type Dog struct {
}

func (d Dog) Speak() string {
    return "Woof!"
}

type Cat struct {
}

func (c Cat) Speak() string {
    return "Meow!"
}

func main() {
    animals := []Animal{Dog{}, Cat{}}
    for _, animal := range animals {
        fmt.Println(animal.Speak())
    }
}
```

Hasilnya:

```
$ go run interfaces.go
Woof!
Meow!
$
```

## 7.4. Array

Array (sering juga disebut larik) adalah struktur data yang digunakan untuk menampung banyak data dengan tipe dan ketentuan yang homogen. Misal, dalam universitas terdapat NIM (Nomor Induk Mahasiswa). Untuk keperluan itu, cukup mempunyai satu nama variabel, setelah itu variabel tersebut mempunyai index, misal **mahasiswa[04]** untuk menunjukkan **mahasiswa pada**

**urutan ke 5** (karena urutan indeks dimulai dari 0).

```
// taken from: https://www.thegeekstuff.com/2019/03/go-array-examples/  
package main  
  
import "fmt"  
  
func main() {  
    // String Array  
    fmt.Println("1. String Array : ")  
  
    var distros [5]string  
    distros[0] = "Ubuntu"  
    distros[1] = "CentOS"  
    distros[2] = "RedHat"  
    distros[3] = "Debian"  
    distros[4] = "OpenBSD"  
  
    mydistro := distros[1]  
    fmt.Println("mydistro = ", mydistro)  
    fmt.Println("distros[2] = ", distros[2])  
    fmt.Println("distros = ", distros)  
    fmt.Println("Number of distros = ", len(distros))  
  
    // Integer Array (Numbers)  
    fmt.Println()  
    fmt.Println("2. Integer Array : ")  
  
    var ids [5]int  
    ids[0] = 1  
    ids[1] = 2  
    ids[2] = 3  
    ids[3] = 4  
    ids[4] = 5  
  
    myid := ids[3]  
    fmt.Println("myid = ", myid)  
    fmt.Println("ids[2] = ", ids[2])  
    fmt.Println("ids = ", ids)  
    fmt.Println("Number of ids = ", len(ids))  
  
    // Declare and Initialize Array at the same time  
    fmt.Println()  
    fmt.Println("3. Declare and Initialize Array at the same time : ")  
  
    os := [3]string{"Linux", "Mac", "Windows"}  
    fmt.Println("os = ", os)  
    fmt.Println("Number of os = ", len(os))  
  
    fibonacci := [6]int{1, 1, 2, 3, 5, 8}
```

```

fmt.Println("fibonacci = ", fibonacci)

// Multi-line Array Initialization Syntax
fmt.Println()
fmt.Println("4. Multi-line Array Initialization Syntax : ")

temperature := [3]float64{
    98.5,
    65.5,
    83.2,
}
fmt.Println("temperature = ", temperature)

names := [3]string{
    "John",
    "Jason",
    "Alica",
    // "Rita",
}
fmt.Println("names = ", names)

// Default Values in an Array
fmt.Println()
fmt.Println("5. Default Values in an Array : ")

empIds := [5]int{101, 102, 103}
fmt.Println("empIds = ", empIds)

empNames := [5]string{"John", "Jason"}
fmt.Println("empNames = ", empNames)

// Loop through Array using For and Range
fmt.Println()
fmt.Println("6. Loop through Array using For and Range : ")

for index, value := range distros {
    fmt.Println(index, " = ", value)
}

// Loop through Array using For and Range (Ignore Index)
fmt.Println()
fmt.Println("7. Loop through Array using For and Range (Ignore Index) : ")

total := 0
for _, value := range ids {
    total = total + value
}
fmt.Println("total of all ids = ", total)

// Initialize an integer array with sequence
fmt.Println()

```

```

fmt.Println("8. Initialize an integer array with sequence : ")
var sequence [10]int
counter := 10
for index, _ := range sequence {
    sequence[index] = counter
    counter = counter + 5
}
fmt.Println("sequence = ", sequence)

// Multi dimensional array
fmt.Println()
fmt.Println("9. Multi dimensional array : ")

count := 1
var multi [4][2]int
for i := 0; i < 4; i++ {
    for j := 0; j < 2; j++ {
        multi[i][j] = count
        count++
    }
}
fmt.Println("Array 4 x 2 : ", multi)
}

```

Hasilnya:

```

$ go run array.go
1. String Array :
mydistro = CentOS
distros[2] = RedHat
distros = [Ubuntu CentOS RedHat Debian OpenBSD]
Number of distros = 5

2. Integer Array :
myid = 4
ids[2] = 3
ids = [1 2 3 4 5]
Number of ids = 5

3. Declare and Initialize Array at the same time :
os = [Linux Mac Windows]
Number of os = 3
fibonacci = [1 1 2 3 5 8]

4. Multi-line Array Initialization Syntax :
temperature = [98.5 65.5 83.2]
names = [John Jason Alica]

5. Default Values in an Array :
empIds = [101 102 103 0 0]
empNames = [John Jason ]

6. Loop through Array using For and Range :
0 = Ubuntu
1 = CentOS
2 = RedHat
3 = Debian
4 = OpenBSD

7. Loop through Array using For and Range (Ignore Index) :
total of all ids = 15

8. Initialize an integer array with sequence :
sequence = [10 15 20 25 30 35 40 45 50 55]

9. Multi dimensional array :
Array 4 x 2 : [[1 2] [3 4] [5 6] [7 8]]

```

## 7.5. Slices

Slices mirip dengan array, hanya saja slices tidak menentukan jumlah elemennya. Jadi, ukuran dari slices merupakan ukuran yang dinamis.

```

package main

import "fmt"

func main() {
    distros := [6]string{"arch", "gentoo", "opensuse", "devuan", "debian",
"fedora"}

    var myDistros []string = distros[3:5]

    fmt.Println(myDistros)
    fmt.Println(myDistros[0])

    distros[4] = "Ubuntu"

    // slices merupakan referensi ke array. Jika array diubah, slices juga
    berubah
    // karena menunjuk ke lokasi memory yang sama
    fmt.Println(myDistros)

    // membuat slices dengan cara ini juga bisa:
    b := make([]int, 1, 5)
    fmt.Println(b)

    b[0] = 2
    fmt.Println(len(b))
    fmt.Println(b[0])
}

```

```

$ go run slices.go
[devuan debian]
devuan
[devuan Ubuntu]
[0]
1
2
$

```

## 7.6. Map

**Map** merupakan implementasi dari **hash table** di Go. Go menyediakan fasilitas untuk **lookup**, menambah, mengedit, dan menghapus suatu **key** dan **value** dari **map**.

```

package main

import "fmt"

type Pegawai struct {
    Nama string
    Usia int
}

func main() {

    var mPeg = make(map[string]Pegawai)

    mPeg["p0001"] = Pegawai{Nama: "Peg 1", Usia: 23}
    fmt.Println(mPeg)

    mPeg["p0001"] = Pegawai{Nama: "Peg 1 - edit", Usia: 24}
    fmt.Println(mPeg)

    mPeg["p0002"] = Pegawai{Nama: "Peg 2", Usia: 29}
    fmt.Println(mPeg)

    // iterasi
    for key, value := range mPeg {
        fmt.Println("NIP:", key, "Nama:", value.Nama, "Usia:", value.Usia)
    }

    // memeriksa ada atau tidak
    _, ok := mPeg["p0002"]
    if ok {
        fmt.Println("Ada pegawai dengan NIP p0002")
    }

    // contoh inisialisasi lagi
    mPeg2 := map[string]Pegawai{
        "p0003": {Nama: "Peg 3", Usia: 34},
        "p0004": {Nama: "Peg 4", Usia: 35},
        "p0005": {Nama: "Peg 5", Usia: 36},
        "p0006": {Nama: "Peg 6", Usia: 37},
        "p0007": {Nama: "Peg 7", Usia: 38},
        "p0008": {Nama: "Peg 8", Usia: 39},
    }
    fmt.Println(mPeg2)
    fmt.Println(mPeg2["p0008"])

    fmt.Println("Jumlah mPeg: ", len(mPeg))
    fmt.Println("Jumlah mPeg2: ", len(mPeg2))

    delete(mPeg2, "p0008")

    fmt.Println("Jumlah mPeg: ", len(mPeg))
    fmt.Println("Jumlah mPeg2: ", len(mPeg2))
}

```

```
fmt.Println(mPeg2)
fmt.Println(mPeg2["p0008"])
```

```
}
```

Hasilnya:

```
$ go run map.go
map[p0001:{Peg 1 23}]
map[p0001:{Peg 1 - edit 24}]
map[p0001:{Peg 1 - edit 24} p0002:{Peg 2 29}]
NIP: p0001 Nama: Peg 1 - edit Usia: 24
NIP: p0002 Nama: Peg 2 Usia: 29
Ada pegawai dengan NIP p0002
map[p0003:{Peg 3 34} p0004:{Peg 4 35} p0005:{Peg 5 36} p0006:{Peg 6 37}
p0007:{Peg 7 38} p0008:{Peg 8 39}]
{Peg 8 39}
Jumlah mPeg: 2
Jumlah mPeg2: 6
Jumlah mPeg: 2
Jumlah mPeg2: 5
map[p0003:{Peg 3 34} p0004:{Peg 4 35} p0005:{Peg 5 36} p0006:{Peg 6 37}
p0007:{Peg 7 38}]
{ 0}
$
```



## Bab 8. Konkurensi dan Paralelisme

Konkurensi berbeda dengan paralelisme. Konkurensi berkaitan dengan "menangani banyak hal pada satu kurun waktu tertentu", sedangkan paralelisme terkait dengan "mengerjakan banyak hal pada satu saat tertentu". Perbedaan dari kedua hal ini adalah:

- konkurensi **tidak** mengerjakan lebih dari satu pekerjaan pada saat yang sama, tetapi pada kurun waktu tertentu akan saling bergantian secara cepat - jadi akan mempunyai kesan "dikerjakan berbarengan".
- paralelisme **benar-benar** mengerjakan lebih dari satu pekerjaan pada saat yang sama dengan memanfaatkan lebih dari satu **core** CPU.

Go mendukung konkurensi dengan menggunakan **goroutine**, yaitu dengan menambahkan **go** pada bagian depan fungsi yang akan dijalankan secara konkuren. Untuk mengatur berbagai **goroutine**, bisa digunakan **channel**.

## Part III: Go untuk Kasus Spesifik

Bagian ini membahas tentang penggunaan Go untuk menyelesaikan masalah tertentu dalam pembuatan software. Penyelesaian masalah bisa dilakukan dengan menggunakan pustaka standar maupun pustaka pihak ketiga. Pada bagian ini, pembahasan tidak hanya ditujukan pada pustaka standar tetapi juga pustaka pihak ketiga selama bisa digunakan untuk menyelesaikan masalah dalam domain masalah tertentu.

## **Bab 9. Testing**

### **9.1. Sub Section 1**

## **Bab 10. I/O dan Filesystems**

### **10.1. Sub Section 1**

## **Bab 11. Akses Basis Data**

### **11.1. Sub Section 1**

## **Bab 12. Sistem Terdistribusi**

### **12.1. Sub Section 1**

## **Bab 13. Aplikasi Web**

### **13.1. Sub Section 1**