Pemrograman Go

Dr. Bambang Purnomosidi D. P.

Daftar Isi

1 Tentang Buku Ini


```
Dr. Bambang Purnomosidi D. P.
Magister Teknologi Informasi
STMIK AKAKOM
Jl. Raya Janti no 143 Yogyakarta
bpdp@akakom.ac.id
Phone: 0274 486664
```

2 Pengenalan Go

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 untuk informasi lebih lanjut.

Lisensi Go

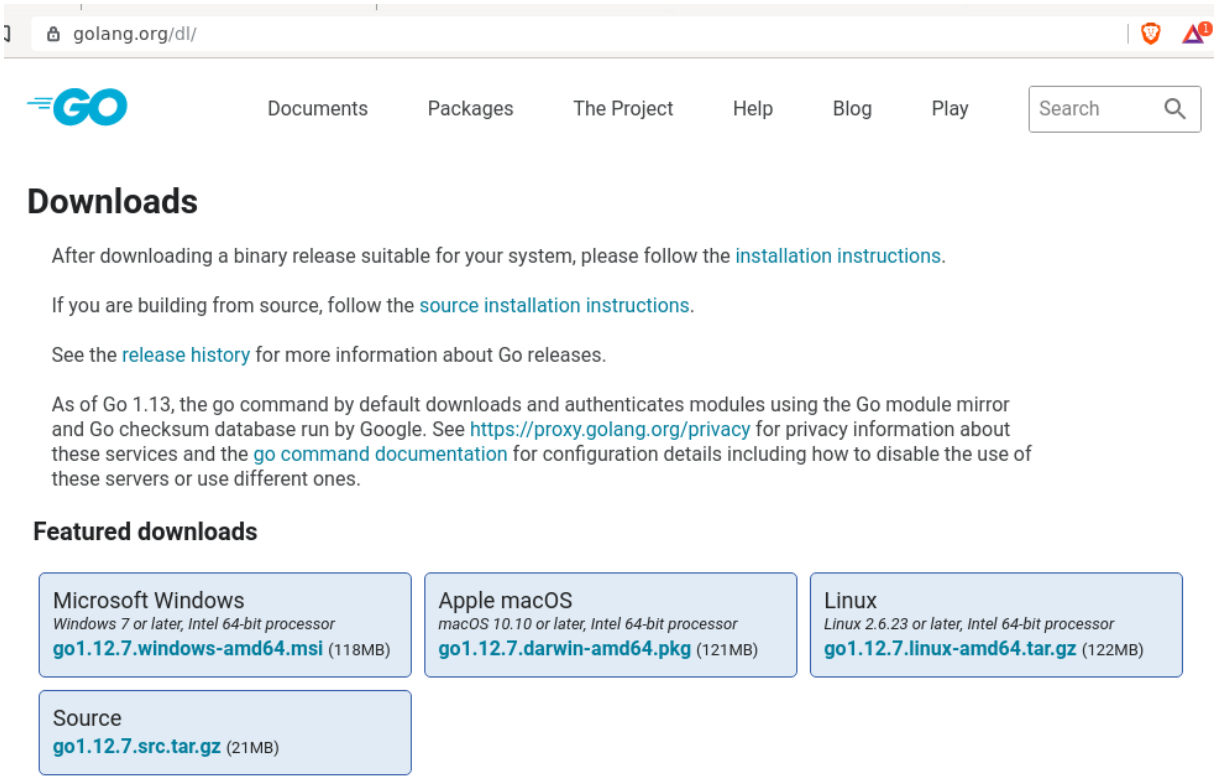
Go didistribusikan dengan menggunakan lisensi modifikasi dari BSD. Lisensi lengkap dari Go bisa di-akses di URL Lisensi Go. Secara umum, penggunaaan lisensi ini mempunyai implikasi sebagai berikut:

- 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.

Instalasi Go

Distribusi 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 downloads Go seperti bisa dilihat di gambar berikut:



Gambar 2.1: Repository downloads Go

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.

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 mencampur-aduk 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 mentargetkan satu versi. Di bagian ini penulis akan menjelaskan konfigurasi yang penulis gunakan untuk menangani masalah tersebut.

```
Catatan:

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:

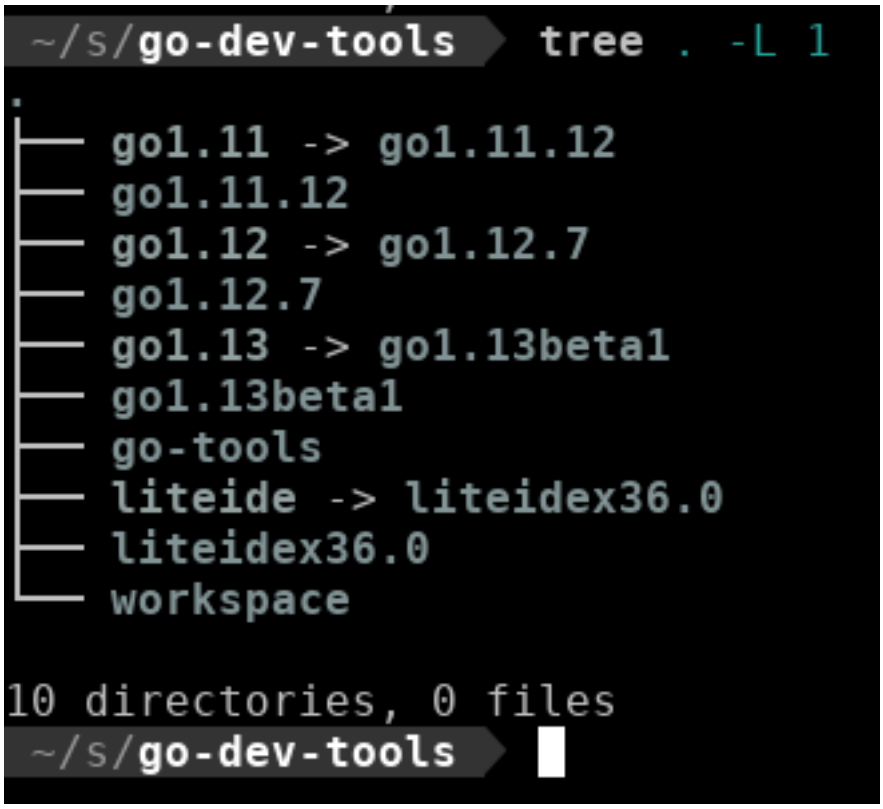
```
$ ls -la
total 475520
drwxr-xr-x   4 bpdp bpdp      4096 Jul 21 08:16 ./
drwxr-xr-x  88 bpdp bpdp      4096 Jul 12 14:17 ../
-rw-r--r--   1 bpdp bpdp 127959471 Jul 19 04:41 go1.12.7.linux-amd64.
    tar.gz
...
...
$ mkdir -p $HOME/software/go-dev-tools
$ cd $HOME/software/go-dev-tools
$ tar -xvf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

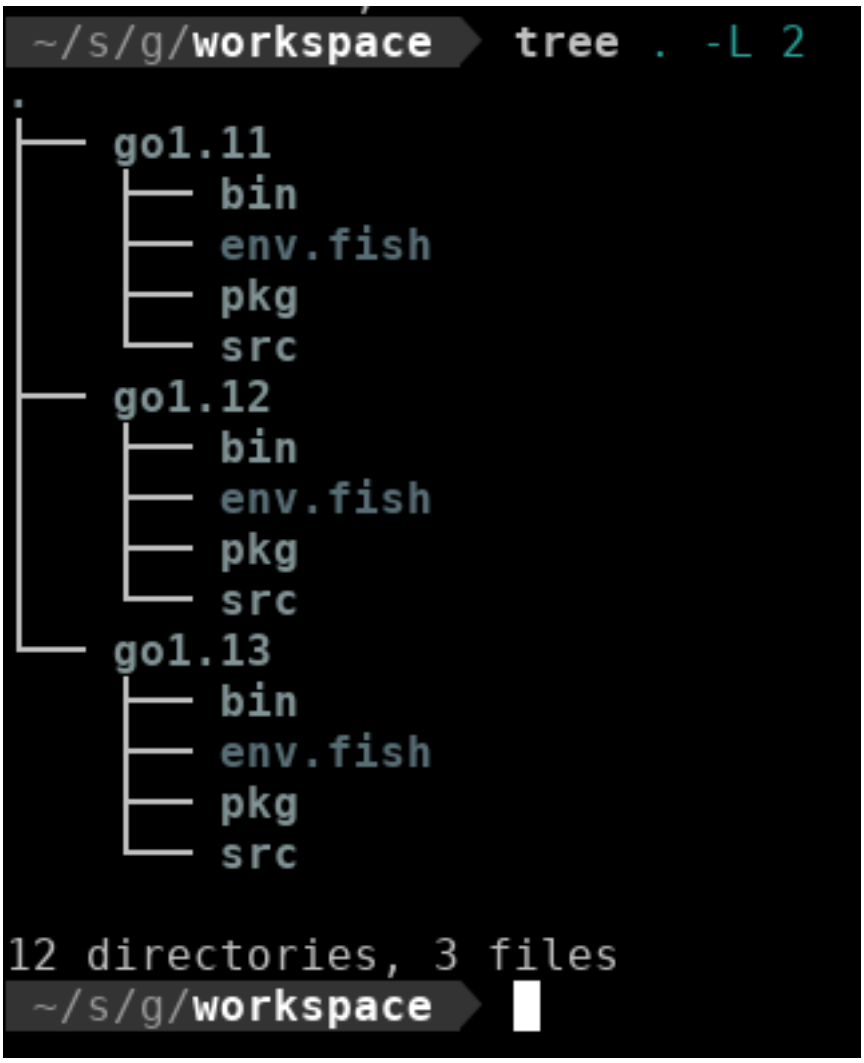
Konfigurasi Variabel Lingkungan Sistem Operasi, Compiler Go, dan Workspace

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):



Gambar 2.2: Struktur direktori

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



Gambar 2.3: Struktur 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 GOTOOLS $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 $GOTOOLS'

set -x PATH $PATH $GO_HOME/bin $LITEIDE_HOME/bin $GOTOOLS/bin
```

Jika menggunakan *Bash*:

```
GODEVTOOLS_HOME=/home/bpdp/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:
    $GO3RDPARTYTOOLS/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.

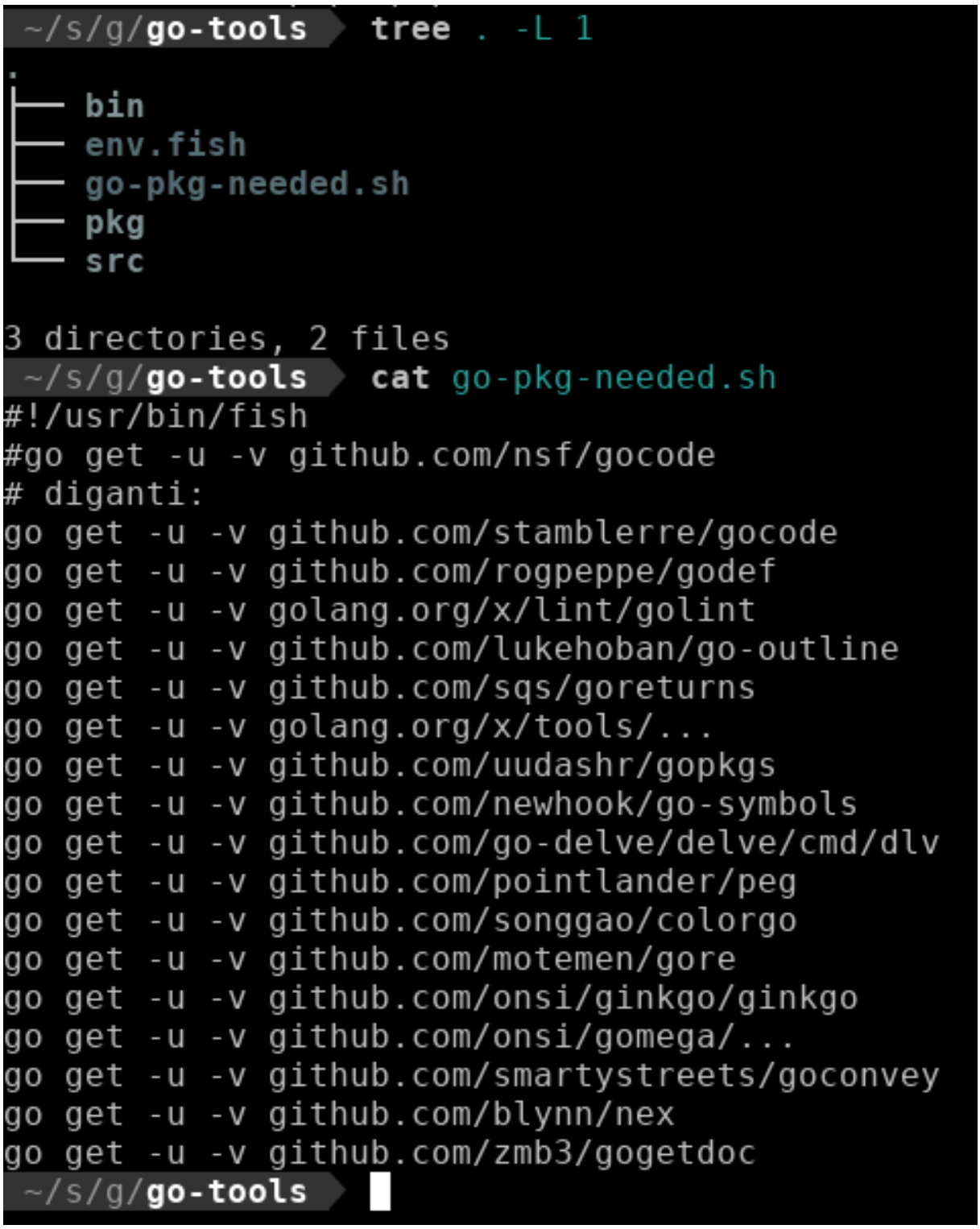
Saat bekerja menggunakan Go, pada dasarnya kita akan menemukan berbagai macam proyek yang bisa dikategorikan menjadi 2 berdasarkan output dari proyek tersebut:

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:



Gambar 2.4: Struktur direktori untuk 3rd party 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/rogpeppe/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/smartystreets/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.

Catatan: 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 kembangkang 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.

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`.

```
1  // hello.go
2  package main
3
4  import "fmt"
5
6  func main() {
7      fmt.Printf("hello, world\n")
8  }
```

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

```
1  $ go run hello.go
2  hello, world
3  $ go build hello.go
4  $ ls -la
5  total 1980
6  drwxr-xr-x 2 bpdp bpdp    4096 Jul 21 10:41 ./
7  drwxr-xr-x 3 bpdp bpdp    4096 Jul 21 10:40 ../
8  -rwxr-xr-x 1 bpdp bpdp 2014135 Jul 21 10:41 hello*
9  -rw-r--r-- 1 bpdp bpdp      86 Jul 21 10:40 hello.go
10 $ file hello
11 hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically
       linked, Go
12 BuildID=-WQRW-exSunj5kUQwAX9/zYf98wtRiMVNHHsFRqn-/1wN0h--29c_4cVsSKleo
       /4LgNCTrEswXoVuMCJgkH, not
13 stripped
14 $ strip hello
15 $ ls -la
16 total 1400
17 drwxr-xr-x 2 bpdp bpdp    4096 Jul 21 10:42 ./
18 drwxr-xr-x 3 bpdp bpdp    4096 Jul 21 10:40 ../
```

```
-rwxr-xr-x 1 bpdp bpdp 1420104 Jul 21 10:42 hello*
-rw-r--r-- 1 bpdp bpdp      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
$
```

Memahami Lingkungan Peranti Pengembangan Go

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

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

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

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.
```

```
$
```

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
```

```
28          show timestamps with directory listings
29    -url string
30          print HTML for named URL
31    -v    verbose mode
32    -write_index
33          write index to a file; the file name must be specified with -
               index_files
34    -zip string
35          zip file providing the file system to serve; disabled if empty
36  $
```

gofmt

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

```
1  $ gofmt --help
2  usage: gofmt [flags] [path ...]
3    -cpuprofile string
4          write cpu profile to this file
5    -d    display diffs instead of rewriting files
6    -e    report all errors (not just the first 10 on different lines)
7    -l    list files whose formatting differs from gofmt's
8    -r string
9          rewrite rule (e.g., 'a[b:len(a)] -> a[b:]')
10   -s    simplify code
11   -w    write result to (source) file instead of stdout
12 $
```

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

```
1  // hello-unformatted.go
2  package main
3  import "fmt"
4  func main() {
5      fmt.Printf("halo\n") // menampilkan tulisan
6         fmt.Printf("dunia")  // ini tulisan baris kedua
7  }
```

Format file kode sumber di atas sebagai berikut:

```
1  $ 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
}
```

3 IDE Untuk Go

IDE (*Integrated Development Environment*) 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.

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*.

Menggunakan Pathogen

Pathogen adalah plugin dari Tim Pope yang digunakan untuk mempermudah pengelolaan plugin. Kode sumber dari Pathogen bisa diperoleh di repository Github. Untuk instalasi, ikuti langkah berikut:

```
1 $ cd
2 $ mkdir .vim/autoload
3 $ mkdir .vim/bundle
4 $ cd .vim/autoload
5 $ wget -c https://raw.githubusercontent.com/tpope/vim-pathogen/master/
    autoload/pathogen.vim
6 --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 `$HOME/.vimrc` atau di `$HOME/.vim/vimrc` (saya pilih lokasi yang kedua) sebagai berikut:

```
execute pathogen#infect()
```

Setelah itu, semua plugin tinggal kita ambil dari repository (bisa dari github, bitbucket, dan lain-lain) langsung di-copy satu direktori ke direktori `$HOME/.vim/bundle`.

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.

Instalasi Plugin

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

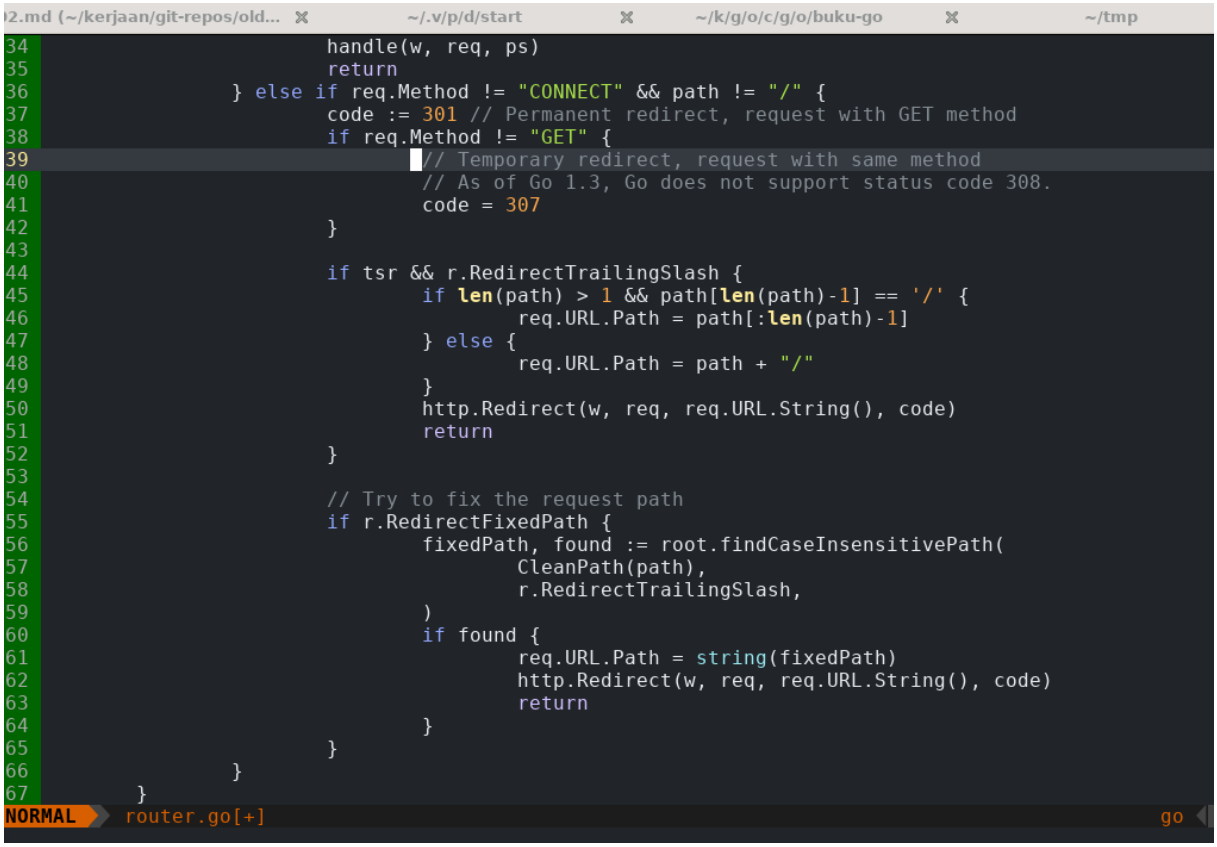
- nerdtree: untuk menampilkan file-file dalam struktur pohon di sebelah kiri sehingga memudahkan navigasi.
- nerdtree-git-plugin: untuk menampilkan status Git.
- vim-go: plugin utama agar Vim mengenali kode sumber Go.
- vim-airline: untuk menampilkan status/tabline dengan format yang lebih bagus.
- vim-airline-themes: 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:



Gambar 3.1: vim-go

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`.

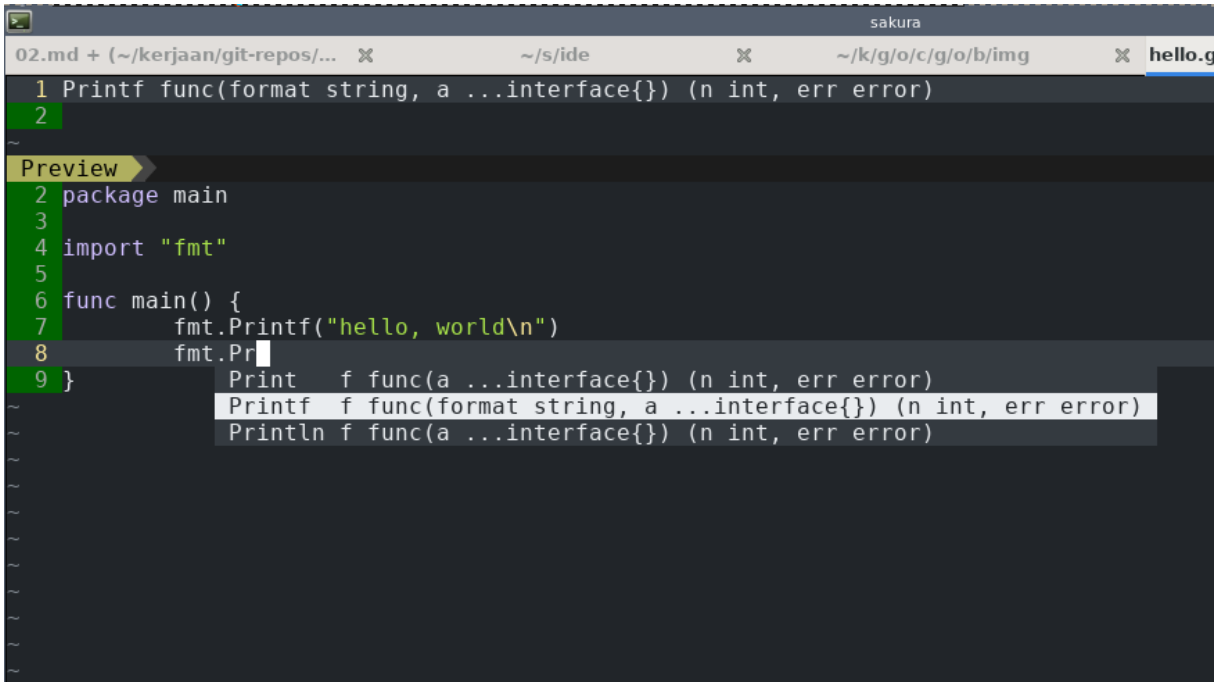
Kode sumber lengkap dari `$HOME/.vim/vimrc` yang penulis gunakan bisa dilihat pada listing berikut ini:

```
1  set number
2  set linebreak
3  set showbreak=+++
4  set textwidth=100
5  set showmatch
6  set nocompatible
7
8  set hlsearch
9  set smartcase
10 set ignorecase
11 set incsearch
12
13 set autoindent
14 set expandtab
15 set shiftwidth=2
16 set smartindent
17 set smarttab
18 set softtabstop=2
19
20 set ruler
21
22 set undolevels=1000
23 set backspace=indent,eol,start
24
25 filetype plugin indent on
26 set ofu=syntaxcomplete#Complete
27
28 " Use 24-bit (true-color) mode in Vim/Neovim when outside tmux or
      screen.
29 " If you're using tmux version 2.2 or later, you can remove the
      outermost $TMUX
30 " check and use tmux's 24-bit color support
31 " (http://sunaku.github.io/tmux-24bit-color.html#usage for more
      information.)
32 if empty($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 gRNERDTreeNodeDelimiter = "\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:



Gambar 3.2: Go completion

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*:

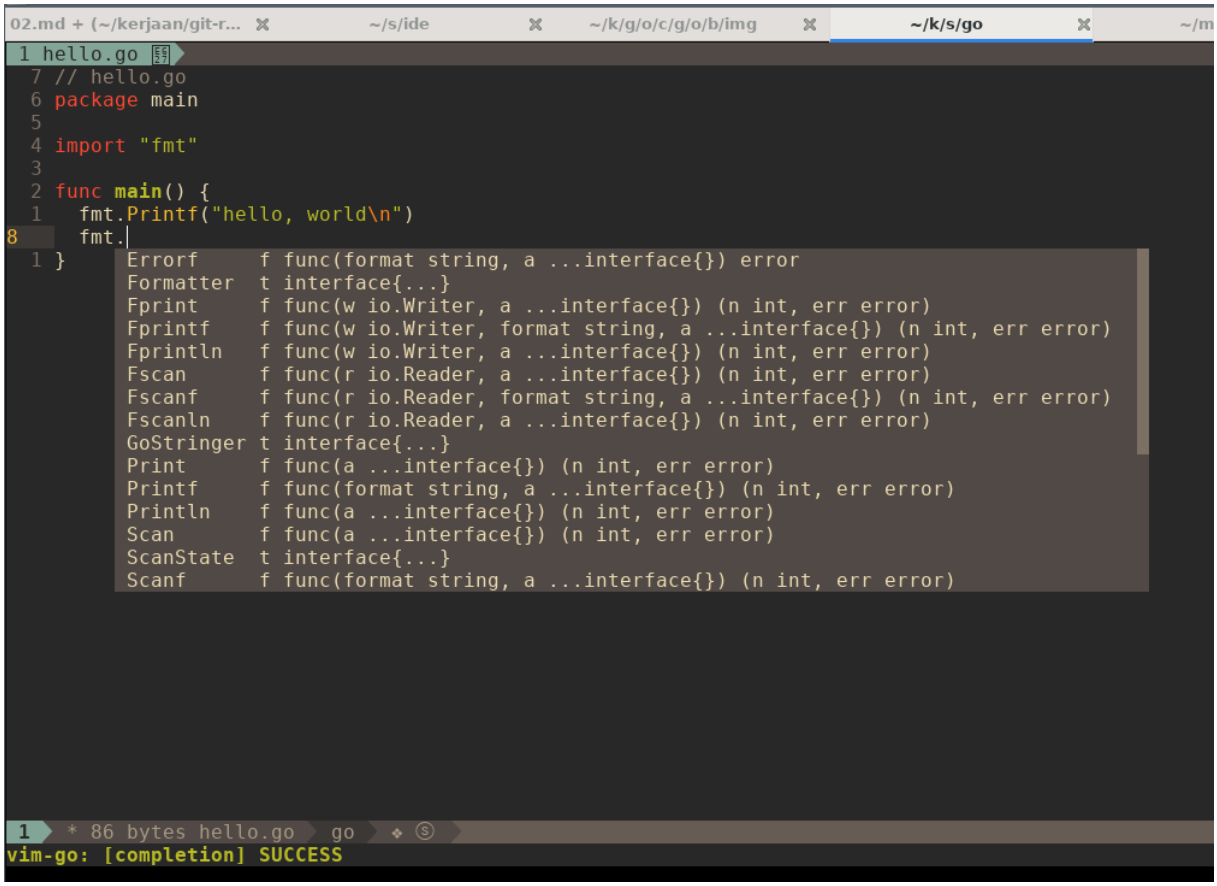
```
$ cd
$ curl -sLf https://spacevim.org/install.sh | bash -s -- --install
    neovim
```

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 24
       bit
    # 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:



Gambar 3.3: Neovim

Menggunakan LiteIDE

Gambar 3.4: LiteIDE

LiteIDE dibuat oleh visualfc dan tersedia dalam bentuk kode sumber maupun binary. Kode sumber bisa diperoleh di repo GitHub. Installer executable bisa diperoleh di Sourceforge.

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

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 Go for Visual Studio Code.

4 Struktur Program Go: Tanpa *Modules* dan Dengan *Modules*

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 $HOPATH ini adalah sebagai berikut:

1. Lokasi pembuatan kode sumber disebut *workspace*.
2. Setiap *workspace* berisi direktori `bin`,`pkg`, dan `src`.
3. Jika $GOPATH tidak ditetepkan, 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 bpdp bpdp 4096 Jan  7  2019 ./
drwxr-xr-x 5 bpdp bpdp 4096 Jul 19 04:48 ../
drwxr-xr-x 2 bpdp bpdp 4096 Jan  7  2019 bin/
-rwxr-xr-x 1 bpdp bpdp   50 Jan  7  2019 env.sh
drwxr-xr-x 2 bpdp bpdp 4096 Jan  7  2019 pkg/
drwxr-xr-x 2 bpdp bpdp 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

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

    (c) 2019 - bpdp.name

*/

// 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
```

```
27 // aplikasi executable.
28 func main() {
29
30     // ini adalah kode sumber / program Go
31     // akan dijelaskan lebih lanjut, abaikan
32     // jika belum paham
33     var (
34         user    string
35         homeDir string
36     )
37
38     user = os.Getenv("USER")
39     homeDir = os.Getenv("HOME")
40
41     fmt.Printf("Halo %s", user)
42     fmt.Printf("\nHome anda di %s", homeDir)
43     fmt.Printf("\n")
44
45 }
```

Nama file: showgoenv.go

```
1  /*
2      showgoenv.go
3
4      Contoh program sederhana untuk menjelaskan
5      struktur program Go untuk lebih dari satu
6          binary executable
7
8      (c) 2019 - bpdp.name
9
10 */
11
12 package main
13
14 import (
15     "fmt"
16     "os"
17 )
18
19 func main() {
20
21     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"
)
```

```
7
8  func main() {
9      fmt.Println(adder.Add(2, 4))
10 }
```

Tanpa *Modules*

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

```
1 $ go run showuserenv.go
2 Halo bpdp
3 Home anda di /home/bpdp
```

Mengkompilasi Menjadi *Binary Executable*

```
1  $ go build showuserenv.go
2  $ ls -la
3  total 1992
4  drwxr-xr-x 2 bpdp bpdp    4096 Jul 29 06:41 ./
5  drwxr-xr-x 3 bpdp bpdp    4096 Jul 29 06:36 ../
6  -rwxr-xr-x 1 bpdp bpdp 2027001 Jul 29 06:41 showuserenv*
7  -rw-r--r-- 1 bpdp bpdp     813 Jul 29 06:39 showuserenv.go
8  $ ./showuserenv
9  Halo bpdp
10 Home anda di /home/bpdp
11 $ file showuserenv
12 showuserenv: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
       statically linked, Go BuildID=9rxlkJ0BRey3wgq8FCzN/
       hTjP17z9sr3yZTvx1JEZ/KJMv7CpJpm_WaUyomyhS/rvGW2o0cNy_kmNe0caJe, not
       stripped
13 $ strip showuserenv
14 $ ls -la
```

```
total 1408
drwxr-xr-x 2 bpdp bpdp    4096 Jul 29 06:42 ./
drwxr-xr-x 3 bpdp bpdp    4096 Jul 29 06:36 ../
-rwxr-xr-x 1 bpdp bpdp 1428296 Jul 29 06:42 showuserenv*
-rw-r--r-- 1 bpdp bpdp     813 Jul 29 06:39 showuserenv.go
$ ./showuserenv
Halo bpdp
Home anda di /home/bpdp
$
```

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

Program Aplikasi: Lebih dari 1 File `binary executable`

Unruk 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:

```
.├──
```

```
bin
  ├── showgoenv
  └── showuserenv
env.fish
pkg
src
    showgoenv
      └── showgoenv.go
    showuserenv
        showuserenv.go
```

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

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

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 bpdp bpdp    4096 Jul 29 21:46 ./
drwxr-xr-x 5 bpdp bpdp    4096 Jul 29 21:26 ../
-rwxr-xr-x 1 bpdp bpdp 2014199 Jul 29 21:46 hellostringutil*
$
```

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.

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 bpdp bpdp    4096 Jul 28 06:55 ./
drwxr-xr-x 4 bpdp bpdp    4096 Jul 28 06:45 ../
-rw------- 1 bpdp bpdp      79 Jul 28 06:55 go.mod
-rw------- 1 bpdp bpdp     499 Jul 28 06:55 go.sum
-rwxr-xr-x 1 bpdp bpdp 2327743 Jul 28 06:55 go-to-hell-o*
-rw-r--r-- 1 bpdp bpdp      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│
```

```
9     ├── 0
      c8659d2f971b567bc9bd6644073413a1534735b75ea8a6f1d4ee4121f78fa5b
10    ├── 0
      c8659d2f971b567bc9bd6644073413a1534735b75ea8a6f1d4ee4121f78fa5b.info
11    ├── 0
      c8659d2f971b567bc9bd6644073413a1534735b75ea8a6f1d4ee4121f78fa5b.lock
12    ├── 4
      db0c9594744360b0eaa452d2ccfbd45b05dffb9810882957d10d69e61e66382
13    ├── 4
      db0c9594744360b0eaa452d2ccfbd45b05dffb9810882957d10d69e61e66382.info
14    ├── 4
      db0c9594744360b0eaa452d2ccfbd45b05dffb9810882957d10d69e61e66382.lock
15    ├── 5
      b03666c2d7b526129bad48c5cea095aad8b83badc1daa202e7b0279e3a5d861
16    ├── 5
      b03666c2d7b526129bad48c5cea095aad8b83badc1daa202e7b0279e3a5d861.info
17    └── 5
      b03666c2d7b526129bad48c5cea095aad8b83badc1daa202e7b0279e3a5d861.lock
      ├──
18  golang.org
19    └── x
20    └── text@v0.0.0-20170915032832-14c0d48ead0c└──
21  rsc.io├──
22      quote@v1.5.2
23        ├── buggy
24        ├── go.mod
25        ├── LICENSE
26        ├── quote.go
27        ├── quote_test.go
28        └── README.md└──
29      sampler@v1.3.0├──
30          glass.go├──
31          glass_test.go├──
32          go.mod├──
33          hello.go├──
34          hello_test.go├──
35          LICENSE└──
36          sampler.go
```

```
37
38 15 directories, 19 files
39 $
```

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:

```
1 $ go mod init github.com/oldstager/gomtk
2 $ cat go.mod
3 module github.com/oldstager/gomtk
4
5 go 1.12
6 $
```

Untuk mengkompilasi:

```
1 $ go build
2 $
```

Penggunaan Pustaka

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

```
1 $ go mod init github.com/oldstager/gomtktest
2 $
```

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

```
1 module github.com/oldstager/gomtktest
2
3 go 1.12
```

```
4
5 replace github.com/oldstager/gomtk => ../gomtk
6 $
```

Untuk mengkompilasi menjadi *binary executable*:

```
1 $ go build
2 $ ls -la
3 total 1976
4 drwxr-xr-x 2 bpdp bpdp    4096 Jul 29 22:43 ./
5 drwxr-xr-x 4 bpdp bpdp    4096 Jul 28 23:43 ../
6 -rw------- 1 bpdp bpdp     166 Jul 28 23:45 go.mod
7 -rwxr-xr-x 1 bpdp bpdp 2005767 Jul 29 22:43 gomtktest*
8 -rw-r--r-- 1 bpdp bpdp     115 Jul 28 23:45 gomtktest.go
9 $
```

5 Sintaksis Dasar Go

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.

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
...
...
```

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*).

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

Tipe	Arti	Jangkauan
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:

```
1  /*
2
3      diambil dari: https://www.callicoder.com/golang-basic-types-
           operators-type-conversion/
4
5  */
6  package main
7
8  import "fmt"
9
10 func main() {
```

```
    var myByte byte = 'a'
    var myRune rune = '♥'

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

Hasil:

```
$ go run char.go
a = 97 and ♥ = 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
$
```

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])
```

```
27      fmt.Println(s.Join([]string{str1, str2}, " "))
28      fmt.Println(s.Join([]string{str3, str2}, "\n"))
29      fmt.Println(s.Join([]string{str3backtick, str2}, "\n"))
30      fmt.Println(reflect.TypeOf(str1))
31      fmt.Println(reflect.TypeOf(str2))
32      fmt.Println()
33
34  }
```

Hasil:

```
1  $ go run varstring.go
2  UGM
3  3
4  true
5  Universitas
6  Gadjah Mada
7  85
8  UGM Yogyakarta
9  universitas
10 gadjah mada
11 Yogyakarta
12 universitas\ngadjah mada
13 Yogyakarta
14 string
15 string
16 $
```

Boolean

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

```
1  package main
2
3  import (
4      "fmt"
5      "reflect"
6  )
7
8  var (
9      hasilPerbandingan bool
10     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
$
```

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
```

```
17      var defInt16 int16
18      var defInt32 int32
19      var defInt64 int64
20      var defInt int
21
22      // string
23      var defString string
24
25      // floating-point
26      var defFloat32 float32
27      var defFloat64 float64
28
29      // complex
30      var defComplex64 complex64
31      var defComplex128 complex128
32
33      // alias
34      var defByte byte
35      var defRune rune
36
37      fmt.Println("\nNilai default untuk uint8 = ", defUint8)
38      fmt.Println("Nilai default untuk uint16 = ", defUint16)
39      fmt.Println("Nilai default untuk uint32 = ", defUint32)
40      fmt.Println("Nilai default untuk uint64 = ", defUint64)
41      fmt.Println("Nilai default untuk uint = ", defUint)
42
43      fmt.Println("\nNilai default untuk int8 = ", defInt8)
44      fmt.Println("Nilai default untuk int16 = ", defInt16)
45      fmt.Println("Nilai default untuk int32 = ", defInt32)
46      fmt.Println("Nilai default untuk int63 = ", defInt64)
47      fmt.Println("Nilai default untuk int = ", defInt)
48
49      fmt.Println("\nNilai default untuk string = ", defString)
50
51      fmt.Println("\nNilai default untuk float32 = ", defFloat32)
52      fmt.Println("Nilai default untuk float64 = ", defFloat64)
53
54      fmt.Println("\nNilai default untuk complex64 = ", defComplex64)
55      fmt.Println("Nilai default untuk complex128 = ", defComplex128)
56
57      fmt.Println("\nNilai default untuk byte = ", defByte)
58      fmt.Println("Nilai default untuk rune = ", defRune)
59
```

```
60 }
```

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 int63 =  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
$
```

Operator

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

Aritmatika

Operator	Deskripsi
+	Penambahan

Operator	Deskripsi
-	Pengurangan
*	Perkalian
/	Pembagian
%	Sisa hasil bagi
&	bitwise AND
|	bitwise OR
^	bitwise XOR
&^	bitclear (AND NOT)
<<	left shift
>>	right shift

Perbandingan

Operator	Deskripsi
==	sama dengan
!=	tidak sama dengan
<	lebih kecil daripada
<=	lebih kecil atau sama dengan
>	lebih besar daripada
>=	lebih besar atau sama dengan

Logika

Operator	Deskripsi
&&	AND
||	OR
!	NOT

Lain-lain (pointer dan channel)

Operator	Deskripsi
&	alamat pointer
*	de-referensi pointer
<-	send / receive untuk channel

Bagian yang biasanya dirasakan cukup rumit adalah operasi bit. Untuk mengetahui lebih lanjut tentang operasi bit, artikel di Medium ini menyediakan informasi serta contoh kode yang cukup lengkap.

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.

```
1  package main
2
3  import (
4      "fmt"
5  )
6
7  func main() {
8
9      const mainCodingLang = "Go"
10     const kiamatMakinDekat = true
11
12     const angka1, angka2 = 25, 8
13
14     const (
15         nomorPegawai = "P001"
16         gaji         = 50000000
17     )
18
19     const negaraKu string = "Indonesia"
20
21     const gajiBersihSetelahSetorIstri = gaji - 49000000
22
```

```
23      fmt.Println(mainCodingLang)
24      fmt.Println(kiamatMakinDekat)
25      fmt.Println(angka1)
26      fmt.Println(angka2)
27      fmt.Println(nomorPegawai)
28      fmt.Println(gaji)
29      fmt.Println(negaraKu)
30      fmt.Println(gajiBersihSetelahSetorIstri)
31
32      // ./konstanta.go:28:7: cannot assign to gaji
33      //gaji = 10000000
34
35  }
```

Hasil:

```
1  $ go run konstanta.go
2  Go
3  true
4  25
5  8
6  P001
7  50000000
8  Indonesia
9  1000000
10 $
```

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*.

```
1  package main
2
3  import "fmt"
4
5  func main() {
```

```
6       i, j := 42, 2701
7
8       // p berisi memory address dari i
9       p := &i
10      // tampilkan isi dari memory address p
11      fmt.Println(*p)
12      // isi dari memory address yang ditunjuk p diubah
13      *p = 21
14      // implikasinya pada variabel i:
15      fmt.Println(i)
16
17      // p berisi memory address dari j
18      p = &j
19      // isi dari memory address yang ditunjuk p, diubah
20      // menjadi isi memoery address yang lama, dibagi 37
21      *p = *p / 37
22      // implikasinya pada variabel j
23      fmt.Println(j) // see the new value of j
24
25      var pa *int
26
27      fmt.Printf("pointer pa dengan tipe %T dan nilai %v\n", pa, pa)
28
29  }
```

Hasil:

```
1  $ go run pointer.go
2  42
3  21
4  73
5  pointer pa dengan tipe *int dan nilai <nil>
6  $
```

Struktur Kendali

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

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!")
```

```
24      case nilaiHuruf == "B":
25          fmt.Println("Lumayan lah")
26      case nilaiHuruf == "C", nilaiHuruf == "D":
27          fmt.Println("Lulus sih ... tapi ..")
28      case nilaiHuruf == "E":
29          fmt.Println("Nangis bombay")
30      default:
31          fmt.Println("Nilai gak jelas, seperti wajah dosennya!")
32      }
33      fmt.Printf("Nilai anda =  %s\n", nilaiHuruf)
34  }
```

Hasil:

```
1  $ go run switch.go
2  Lulus sih ... tapi ..
3  Nilai anda =  D
4  $
```

Perulangan dengan `for`

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

```
1  for [condition |  ( init; condition; increment ) | Range] {
2     statement(s);
3  }
```

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

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      var i, j int
7
8      for i = 1; i < 10; i++ {
9          fmt.Println(i)
10         for j = 1; j <= i; j++ {
11             fmt.Println(j)
12         }
```

```
    }
}
```

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**.

```go
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
$
```

Defer

Defer digunakan untuk mengekesekusi 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
$
```

6 Fungsi / *Function*

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).

Deklarasi Fungsi

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

```
1  package main
2
3  import "fmt"
4
5  func findSumOfChars(strCounted string, theChar rune) int {
6      counter := 0
7      for _, c := range strCounted {
8          if c == theChar {
9              counter++
10         }
11     }
12     return counter
13 }
14
15 func main() {
16     theStr := "STMIK Akakom"
```

```
17      fmt.Printf("Jumlah karakter 'k' dari string %s adalah %d",
18          theStr, findSumOfChars(theStr, 'k'))
19  }
```

Hasil:

```
1  $ go run function-01.go
2  Jumlah karakter 'k' dari string STMIK Akakom adalah 2
3  $
```

Fungsi dengan Banyak Nilai Kembalian

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

```
1  package main
2
3  import "fmt"
4
5  func doMath(a, b float64) (float64, float64, float64, float64) {
6      return a * b, a / b, a + b, a - b
7  }
8
9  func main() {
10
11     a, b, c, d := doMath(4, 10)
12
13     fmt.Println(a)
14     fmt.Println(b)
15     fmt.Println(c)
16     fmt.Println(d)
17
18     _, nil1, _, _ := doMath(12, 21)
19
20     fmt.Println(nil1)
21 }
```

Hasil:

```
1  $ go run f-multi-retval.go
```

```
40
0.4
14
-6
0.5714285714285714
```

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))

```

```
31 }
```

Hasilnya:

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

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
$
```

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
```

```
3
4  import "fmt"
5
6  func main() {
7      n := 0
8      counter := func() int {
9          n += 1
10         return n
11     }
12     fmt.Println(counter())
13     fmt.Println(counter())
14 }
```

Hasilnya:

```
1 $ go run closures.go
2 1
3 2
4 $
```

Fungsi Rekursif

Fungsi rekursif adalah fungsi yang memanggil dirinya sendiri.

```
1  package main
2
3  import "fmt"
4
5  func factorial(i int) int {
6      if i <= 1 {
7          return 1
8      }
9      return i * factorial(i-1)
10 }
11
12 func main() {
13     var i int = 15
14     fmt.Printf("Factorial dari %d is %d", i, factorial(i))
15 }
```

Hasialnya:

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

Call by Value

Saat memmberikan 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
$
```

Call by Pointer

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

```
1  package main
2
3  import "fmt"
4
5  var theVal int = 0
6
7  func changeMe(theVal *int) int {
8
9      *theVal = 150
10
11     return *theVal
12
13 }
14
15 func main() {
16
17     theVal := 45
18
19     fmt.Println(theVal)
20
21     fmt.Println(changeMe(&theVal))
22
23     fmt.Println(theVal)
24
25 }
```

Hasilnya:

```
1  $ go run func-pass-by-pointer.go
2  45
3  150
4  150
5  $
```

7 Penanganan Kesalahan

Penggunaan Kode Error

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.

8 Struktur Data

Arrays

Slices

Maps

Struct

9 Konkurensi dan Paralelisme

10 Testing

11 I/O dan File Systems

12 Akses Basis Data

13 Sistem Terdistribusi

14 Aplikasi Web

15 Tooling