
Pemrograman Go

Dr. Bambang Purnomosidi D. P.



Daftar Isi

1	Tentang Buku Ini	3
2	Pengenalan Go	4
	Apa itu Go?	4
	Lisensi Go	4
	Instalasi Go	5
	Distribusi Go	5
	Download dan Install Go	6
	Konfigurasi Variabel Lingkungan Sistem Operasi, Compiler Go, dan Workspace	7
	Menguji Instalasi Go	14
	Memahami Lingkungan Peranti Pengembangan Go	15
	go	15
	godoc	17
	gofmt	18
3	IDE Untuk Go	20
	Menggunakan Vim	20
	Instalasi dan Konfigurasi Pathogen	20
	Instalasi dan Konfigurasi Plugin Golang dan Plugin Pendukung	21
	Autocompletion	22
	Menggunakan LiteIDE	24
	Software IDE Lain	25
4	Dasar-dasar Pemrograman Go	26
	Struktur Program Go	26
	Program Aplikasi Sederhana - 1 File binary executable Utama	26
	Tanpa Proses Kompilasi	27
	Mengkompilasi Menjadi <i>Binary Executable</i>	27
	Pustaka / Library / Package	28
	Mengatur Workspace	28
	Membuat Pustaka	29

Membuat Aplikasi yang Memanfaatkan Pustaka	29
Konstruksi Dasar Bahasa Pemrograman Go	30
Komentar	30
Tipe Data Angka / Numerik	30
String	32
Boolean	33
Variabel	33
Konstanta	36
Pointer	36
Struktur Kendali	36
Perulangan dengan for	36
Seleksi Kondisi	36
Defer	36
5 Fungsi / Function	37
6 Penanganan Kesalahan	38
Penggunaan Kode Error	38
Panic dan Recover	38
7 Struktur Data Lanjut	39
Arrays	39
Slices	39
Maps	39
Struct	39
8 Method	40
9 Testing	41
10 Konkurensi	42

1 Tentang Buku Ini



Buku ini berisi materi pemrograman Go dengan lisensi Creative Commons Attribution-ShareAlike 4.0 International License - CC-BY-SA 4.0. Secara umum, penggunaan lisensi ini mempunyai implikasi bahwa pengguna materi:

1. Harus memberikan atribusi ke penulis (* Dr. Bambang Purnomosidi D. P.*).
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.

Untuk penggunaan selain ketentuan tersebut, silahkan menghubungi:

- 1 Dr. Bambang Purnomosidi D. P.
- 2 Magister Teknologi Informasi
- 3 STMIK AKAKOM
- 4 Jl. Raya Janti no 143 Yogyakarta
- 5 bdp@akakom.ac.id
- 6 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 **Go^lang** atau **go^lang**. Istilah **Go^lang** atau **go^lang** 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 **#go^lang** 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

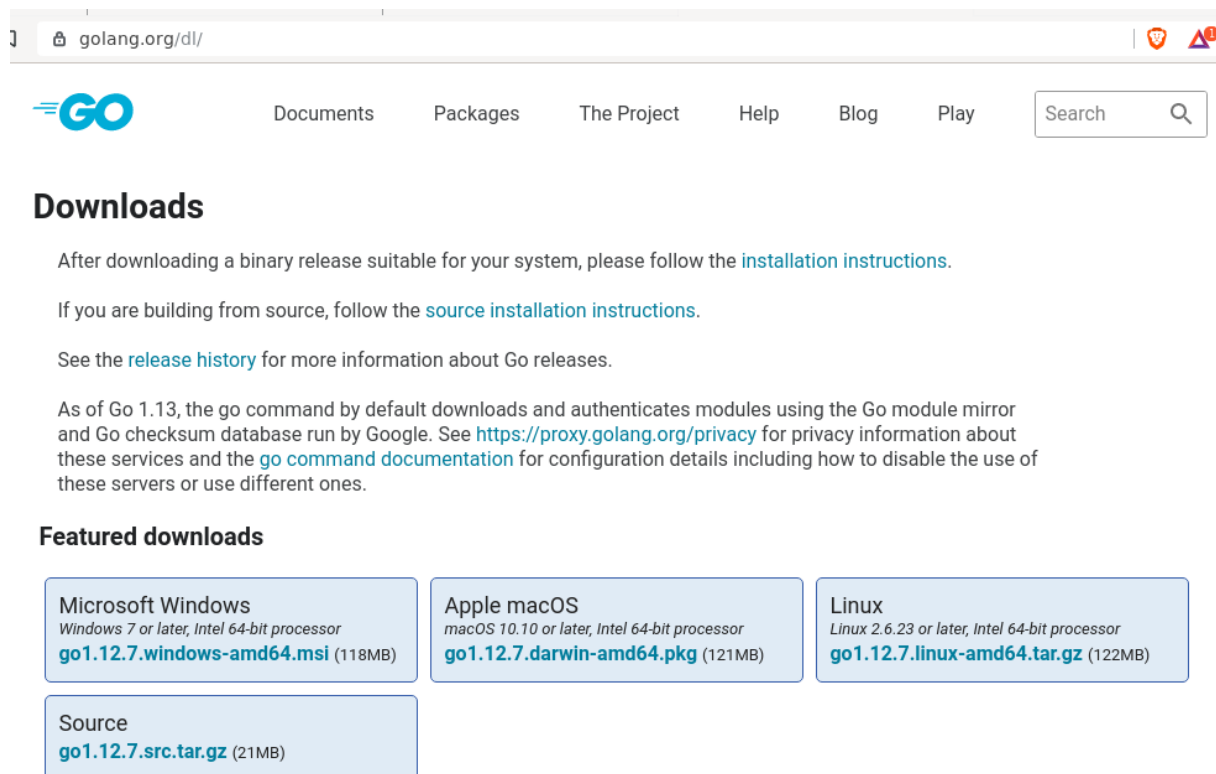
Go didistribusikan dengan menggunakan lisensi modifikasi dari BSD. Lisensi lengkap dari Go bisa diakses di URL Lisensi Go. Secara umum, penggunaan lisensi ini mempunyai implikasi sebagai berikut:

- 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.

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 mencampurkan 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.

```
1 Catatan:
2
3 Go akan diinstall di direktori $HOME/software/go-dev-tools/goVERSI
4
5 VERSI = versi dari Go yang akan diinstall, misalnya go1.12.7
6
7 Lokasi instalasi tersebut digunakan penulis karena penulis mempunyai
  lebih dari 1 versi Go, jika
8 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:

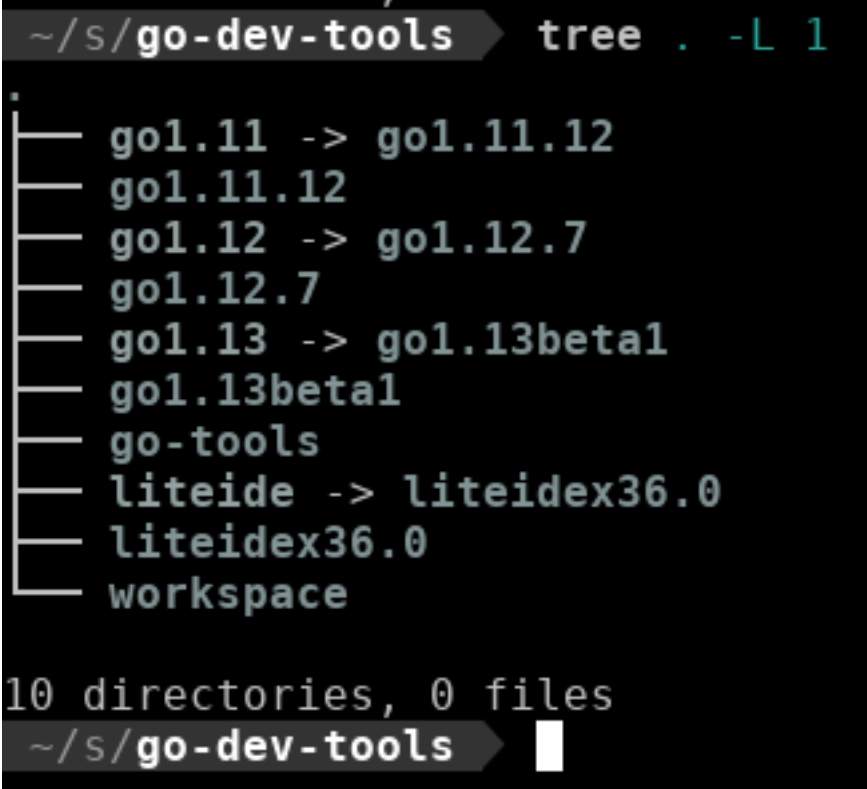
```
1 $ ls -la
2 total 475520
3 drwxr-xr-x  4 bdpd bdpd    4096 Jul 21 08:16 ./
4 drwxr-xr-x 88 bdpd bdpd    4096 Jul 12 14:17 ../
5 -rw-r--r--  1 bdpd bdpd 127959471 Jul 19 04:41 go1.12.7.linux-amd64.
   tar.gz
6 ...
7 ...
8 $ mkdir -p $HOME/software/go-dev-tools
9 $ cd $HOME/software/go-dev-tools
10 $ tar -xvf ~/master/go/go1.12.7.linux-amd64.tar.gz
11 $ 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 compiler, 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):

A terminal window with a dark background. The prompt is `~/s/go-dev-tools`. The command `tree . -L 1` has been executed. The output shows a tree structure of directories: `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 -> liteindex36.0`, `liteindex36.0`, and `workspace`. At the bottom, it says `10 directories, 0 files`. The prompt `~/s/go-dev-tools` is shown again at the bottom.

```
~/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 -> liteindex36.0
├── liteindex36.0
└── workspace

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

Gambar 2.2: Struktur direktori

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
```

Gambar 2.3: Struktur workspace

Isi dari file env.sh menyesuaikan shell yang digunakan::

Bash

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

Fish

```
1 set -x GOPATH (pwd)
2 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`.

```
1 set GODEVTOOLS_HOME /home/bdpd/software/go-dev-tools
2
3 set GO_HOME $GODEVTOOLS_HOME/go1.12
4 set LITEIDE_HOME $GODEVTOOLS_HOME/liteide
5 set GOTTOOLS $GODEVTOOLS_HOME/go-tools
6
7 set -x GOROOT $GO_HOME
8 set -x GOOS linux
9 set -x GOARCH amd64
10 set -x GOHOSTOS linux
11 set -x GOHOSTARCH amd64
12
13 alias godev='cd $GODEVTOOLS_HOME/workspace/go1.12'
14 alias godevtools='cd $GOTTOOLS'
15
16 set -x PATH $PATH $GO_HOME/bin $LITEIDE_HOME/bin $GOTTOOLS/bin
```

Jika menggunakan *Bash*:

```
1 GODEVTOOLS_HOME=/home/bdpd/software/go-dev-tools
2
3 GO_HOME=$GODEVTOOLS_HOME/go/go1.12.7
4 LITEIDE_HOME=$GODEVTOOLS_HOME/liteide
5 GOTTOOLS=$GODEVTOOLS_HOME/go-tools
6
7 export GOROOT=$GO_HOME
8 export GOOS=linux
9 export GOARCH=amd64
10 export GOHOSTOS=linux
11 export GOHOSTARCH=amd64
12
13 export PATH=$PATH:$GO_HOME/bin:$LITEIDE_HOME/bin:$GOTTOOLS:
    $GO3RDPARTYTOOLS/bin
14
15 alias godev='cd $GODEVTOOLS_HOME/workspace/go1.12'
16 alias godevtools='cd $GOTTOOLS'
```

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

```
1 $ source ~/.env/fish/go/go1.12.7
```

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

```
1 $ go env
2 GOARCH="amd64"
3 GOBIN=""
4 GOCACHE="/home/bdp/.cache/go-build"
5 GOEXE=""
6 GOFLAGS=""
7 GOHOSTARCH="amd64"
8 GOHOSTOS="linux"
9 GOOS="linux"
10 GOPATH="/home/bdp/go"
11 GOPROXY=""
12 GORACE=""
13 GOROOT="/home/bdp/software/go-dev-tools/go1.12"
14 GOTMPDIR=""
15 GOTOOLDIR="/home/bdp/software/go-dev-tools/go1.12/pkg/tool/linux_amd64"
16 GCCGO="gccgo"
17 CC="gcc"
18 CXX="g++"
19 CGO_ENABLED="1"
20 GOMOD=""
21 CGO_CFLAGS="-g -O2"
22 CGO_CPPFLAGS=""
23 CGO_CXXFLAGS="-g -O2"
24 CGO_FFLAGS="-g -O2"
25 CGO_LDFLAGS="-g -O2"
26 PKG_CONFIG="pkg-config"
27 GOGCCFLAGS="-fPIC -m64 -pthread -fmessage-length=0 -fdebug-prefix-map=/tmp/go-build584380045=/tmp/go-build -gno-record-gcc-switches"
28 $
```

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/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
~/s/g/go-tools
```

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:

```
1 #!/usr/bin/fish
2
3 # ganti di atas dengan #!/usr/bin/bash jika anda menggunakan Bash
4 go get -u -v github.com/stamblerre/gocode
5 go get -u -v github.com/rogppe/godef
6 go get -u -v golang.org/x/lint/golint
7 go get -u -v github.com/lukehoban/go-outline
8 go get -u -v github.com/sqs/goreturns
9 go get -u -v golang.org/x/tools/...
10 go get -u -v github.com/uudashr/gopkgs
11 go get -u -v github.com/newhook/go-symbols
12 go get -u -v github.com/go-delve/delve/cmd/dlv
13 go get -u -v github.com/pointlander/peg
14 go get -u -v github.com/songgao/colargo
15 go get -u -v github.com/motemen/gore
16 go get -u -v github.com/onsi/ginkgo/ginkgo
17 go get -u -v github.com/onsi/gomega/...
18 go get -u -v github.com/smartystreets/goconvey
19 go get -u -v github.com/blynn/nex
20 go get -u -v github.com/zmb3/gogetdoc
```

Dengan konfigurasi seperti itu, kerjakan berikut ini untuk install:

```
1 $ source env/fish/go/go1.12.7
2 $ godevtools
3 $ source env.sh
4 $ ./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 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.

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 bdp bdp 4096 Jul 21 10:41 ./
7 drwxr-xr-x 3 bdp bdp 4096 Jul 21 10:40 ../
8 -rwxr-xr-x 1 bdp bdp 2014135 Jul 21 10:41 hello*
9 -rw-r--r-- 1 bdp bdp 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 bdp bdp 4096 Jul 21 10:42 ./
18 drwxr-xr-x 3 bdp bdp 4096 Jul 21 10:40 ../
```

```
19 -rwxr-xr-x 1 bdp bdp 1420104 Jul 21 10:42 hello*
20 -rw-r--r-- 1 bdp bdp      86 Jul 21 10:40 hello.go
21 $ ./hello
22 hello, world
23 $ file hello
24 hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically
    linked, Go
25 BuildID=-WQRW-exSunj5kUQwAX9/zYf98wtRiMVNHHsFRqn-/1wN0h--29c_4cVsSKleo
    /4LgNCTrEswXoVuMCJgkH,
26 stripped
27 $
```

Memahami Lingkungan Peranti Pengembangan Go

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

```
1 $ pwd
2 /home/bdp/software/go-dev-tools/go1.12.7/bin
3 $ ls -la
4 total 34744
5 drwxr-xr-x  2 bdp bdp    4096 Jul  9 04:32 ./
6 drwxr-xr-x 10 bdp bdp    4096 Jul  9 04:29 ../
7 -rwxr-xr-x  1 bdp bdp 14617729 Jul  9 04:31 go*
8 -rwxr-xr-x  1 bdp bdp 17422226 Jul  9 04:32 godoc*
9 -rwxr-xr-x  1 bdp bdp  3525802 Jul  9 04:31 gofmt*
10 $
```

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:

```
1 $ go version
2 go version go1.12.7 linux/amd64
3 $ go
4 Go is a tool for managing Go source code.
5
6 Usage:
7
```



```
8      go <command> [arguments]
9
10 The commands are:
11
12      bug          start a bug report
13      build        compile packages and dependencies
14      clean        remove object files and cached files
15      doc          show documentation for package or symbol
16      env          print Go environment information
17      fix          update packages to use new APIs
18      fmt          gofmt (reformat) package sources
19      generate      generate Go files by processing source
20      get          download and install packages and dependencies
21      install      compile and install packages and dependencies
22      list         list packages or modules
23      mod          module maintenance
24      run          compile and run Go program
25      test        test packages
26      tool         run specified go tool
27      version      print Go version
28      vet          report likely mistakes in packages
29
30 Use "go help <command>" for more information about a command.
31
32 Additional help topics:
33
34      buildmode    build modes
35      c            calling between Go and C
36      cache        build and test caching
37      environment  environment variables
38      filetype     file types
39      go.mod       the go.mod file
40      gopath       GOPATH environment variable
41      gopath-get   legacy GOPATH go get
42      goproxy      module proxy protocol
43      importpath   import path syntax
44      modules      modules, module versions, and more
45      module-get   module-aware go get
46      packages     package lists and patterns
47      testflag     testing flags
48      testfunc     testing functions
49
50 Use "go help <topic>" for more information about that topic.
```

```
51 $
```

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).

```
1 $ godoc --help
2 usage: godoc -http=localhost:6060
3   -analysis string
4       comma-separated list of analyses to perform (supported: type,
5       pointer). See http://golang.org/lib/godoc/analysis/help.html
6   -goroot string
7       Go root directory (default "/home/bpdp/software/go-dev-tools/go1.12")
8   -http string
9       HTTP service address (default "localhost:6060")
10  -index
11      enable search index
12  -index_files string
13      glob pattern specifying index files; if not empty, the index is
14      read from these files in sorted order
15  -index_interval duration
16      interval of indexing; 0 for default (5m), negative to only
17      index once at startup
18  -index_throttle float
19      index throttle value; 0.0 = no time allocated, 1.0 = full
20      throttle (default 0.75)
21  -links
22      link identifiers to their declarations (default true)
23  -maxresults int
24      maximum number of full text search results shown (default
25      10000)
26  -notes string
27      regular expression matching note markers to show (default "BUG")
28  -play
29      enable playground
30  -templates string
31      load templates/JS/CSS from disk in this directory
32  -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:

```
1 // hello-formatted.go
2 package main
3
4 import "fmt"
5
6 func main() {
7     fmt.Printf("halo\n") // menampilkan tulisan
8     fmt.Printf("dunia")  // ini tulisan baris kedua
9 }
```

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. Sebaiknya, menggunakan *pathogen* untuk mempermudah pengelolaan berbagai plugin tersebut. Bagian ini akan menjelaskan berbagai konfigurasi serta instalasi yang diperlukan sehingga Vim bisa menjadi peranti untuk pengembangan aplikasi menggunakan Go.

Instalasi dan Konfigurasi 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 https://raw.githubusercontent.com/tpope/vim-pathogen/master/
  autoload/pathogen.vim
6 --2016-08-15 11:16:50-- https://raw.githubusercontent.com/tpope/vim-
  pathogen/master/autoload/pathogen.vim
```

```
7 Resolving raw.githubusercontent.com (raw.githubusercontent.com)...  
  151.101.100.133  
8 Connecting to raw.githubusercontent.com (raw.githubusercontent.com)  
  |151.101.100.133|:443... connected.  
9 HTTP request sent, awaiting response... 200 OK  
10 Length: 12474 (12K) [text/plain]  
11 Saving to: 'pathogen.vim'  
12  
13 pathogen.vim  
   100%[=====>]  12.18K  
   --.-KB/s    in 0.02s  
14  
15 2016-08-15 11:16:56 (512 KB/s) - 'pathogen.vim' saved [12474/12474]  
16  
17 $
```

Setelah itu, untuk menggunakan Pathogen, letakkan aktivasinya di `$HOME/.vimrc` atau di `$HOME/.vim/vimrc` (saya pilih lokasi yang kedua) sebagai berikut:

```
1 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`.

Instalasi dan Konfigurasi Plugin Golang dan Plugin Pendukung

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

- Colorschemes: untuk tema warna dari Vim.
- Nerdtree: untuk menampilkan file-file dalam struktur pohon di sebelah kiri sehingga memudahkan navigasi.
- vim-go: plugin utama agar Vim mengenali kode sumber Go.

Cara instalasi:

```
1 $ cd  
2 $ cd .vim/bundle  
3 $ git clone <masing-masing lokasi plugin>
```

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

```

1 /*
2  aplikasi.go
3
4  Contoh program sederhana untuk menjelaskan
5  struktur program Go untuk aplikasi executable
6
7  (c) bdp.name
8
9  */
10
11 // Program Go diawali dengan nama pake.
12 // Paket untuk aplikasi executable selalu berada
13 // pada paket main.
14 package main
15
16 // pustaka standar yang diperlukan
17 // Jika hanya satu:
18 // import "fmt"
19 // Jika lebih dari satu:
20 import (
21     "fmt"
22     "os"
23 )
24
25 // "Fungsi" merupakan satuan terintegrasi dari
26 // 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, no need to
32     // care a lot right now.
33     var (
34         user string
35         homeDir string
36         goHome string
37     )
38
39     user = os.Getenv("USER")
40     homeDir = os.Getenv("HOME")

```

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 Gocode. Gocode sudah terinstall setelah selesai mengerjakan instalasi di bab 1. Hasil dari instalasi Gocode adalah file `executable binary $GOPATH/bin/gocode` (sesuai letak GOPATH di `env.sh`). Sebelum menggunakan Vim, aktifkan dulu gocode dengan mengeksekusi `gocode` melalui shell. Setelah itu, tambahkan satu baris di `$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 execute pathogen#infect()
2
3 syntax on
4 filetype plugin indent on
5 set ofu=syntaxcomplete#Complete
6
7 if has("gui_running")
8     colorscheme asmanian_blood
9 else
10    colorscheme slate
11 endif
12
13 set smartindent
14 set tabstop=2
15 set shiftwidth=2
16 set expandtab
17
18 autocmd vimenter * NERDTree
19 autocmd vimenter * if !argc() | NERDTree | endif
20 autocmd bufenter * if (winnr("$") == 1 && exists("b:NERDTreeType") && b
    :NERDTreeType == "primary") | q | endif
21
22 let g:NERDTreeDirArrows=0
23
24 let g:cssColorVimDoNotMessMyUpdatetime = 1
25 set guifont=Liberation\ Mono\ 11
26
27 set number
28 set numberwidth=4
29 set coptions+=n
30 highlight LineNr term=bold cterm=NONE ctermfg=DarkGrey ctermbg=NONE gui
    =NONE guifg=DarkGrey guibg=NONE
31
32 set grepprg=grep\ -nH\ $*
```

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:

The screenshot shows the LiteIDE Go IDE interface. The top menu bar includes 'Edit', 'Tools', 'Syntax', 'Buffers', 'Window', 'Plugin', 'Themes', and 'Help'. Below the menu is a toolbar with various icons for file operations, editing, and running. The main editor area displays Go code for the `fmt` package's `Scanln` function. The code is as follows:

```

1 func Scanln(a ...interface{}) (n int, err error)
2
3     func Errorf(format string, a ...interface{}) error
4     func Fprintf(w io.Writer, format string, a ...interface{}) (n int, err error)
5     func Fprintln(w io.Writer, a ...interface{}) (n int, err error)
6     var func Fscanf(r io.Reader, format string, a ...interface{}) (n int, err error)
7     fmt func Fscanf(r io.Reader, format string, a ...interface{}) (n int, err error)
8     fmt func Fscanln(r io.Reader, a ...interface{}) (n int, err error)
9     fmt func Print(a ...interface{}) (n int, err error)
10    fmt func Printf(format string, a ...interface{}) (n int, err error)
11    var func Println(a ...interface{}) (n int, err error)
12    fmt func Scan(a ...interface{}) (n int, err error)
13    fmt func Scanf(format string, a ...interface{}) (n int, err error)
14    fmt func Scanln(a ...interface{}) (n int, err error)
15    fmt func Sprint(a ...interface{}) string
16    var func Sprintf(format string, a ...interface{}) string
17    fmt func Sprintln(a ...interface{}) string
18    fmt func Sscan(str string, a ...interface{}) (n int, err error)
19    fmt func Sscanf(str string, format string, a ...interface{}) (n int, err error)
20    fmt func Sscanln(str string, a ...interface{}) (n int, err error)
21    var type Formatter interface
22    fmt type GoStringer interface
23    ptr type ScanState interface
24    fmt type Scanner interface
25    fmt type State interface
26    type Stringer interface
27    fmt.Errorf()
28
29 }

```

At the bottom of the editor, a status bar shows the file path `e/bdpd/src/go/pointer/pointer.go` and indicates `mini completion (AOANAP) match 13 of 25`.

Gambar 3.2: Go completion

Menggunakan LiteIDE



Gambar 3.3: 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 `gocode` 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).

4 Dasar-dasar Pemrograman Go

Struktur Program Go

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 berikut ini:

```
1  /*
2      aplikasi.go
3
4      Contoh program sederhana untuk menjelaskan
5      struktur program Go untuk aplikasi executable
6
7      (c) bdpd.xyz
8
9  */
10
11 // Program Go diawali dengan nama paket.
12 // Paket untuk aplikasi executable selalu berada
13 // pada paket main.
14 package main
15
16 // pustaka standar yang diperlukan
17 // Jika hanya satu:
18 // import "fmt"
19 // Jika lebih dari satu:
20 import (
21     "fmt"
22     "os"
23 )
24
25 // "Fungsi" merupakan satuan terintegrasi dari
26 // 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         goHome  string
37     )
38
39     user = os.Getenv("USER")
40     homeDir = os.Getenv("HOME")
41     goHome = os.Getenv("GOROOT")
42
43     fmt.Printf("Halo %s", user)
44     fmt.Printf("\nHome anda di %s", homeDir)
45     fmt.Printf("\nAnda menggunakan Go di %s", goHome)
46     fmt.Printf("\n")
47
48 }
```

Untuk menjalankan kode sumber di atas, ikuti langkah-langkah berikut:

Tanpa Proses Kompilasi

```
1 $ go run aplikasi.go
2 Halo bdpd
3 Home anda di /home/bdpd
4 Anda menggunakan Go di /home/bdpd/software/go-dev-tools/go/go1.7.3
```

Mengkompilasi Menjadi *Binary Executable*

```
1 $ go build aplikasi.go
2 $ ls -la
3 total 1620
4 drwxr-xr-x 1 bdpd users      38 Sep 12 22:49 .
5 drwxr-xr-x 1 bdpd users    288 Aug 15 11:24 ..
6 -rwxr-xr-x 1 bdpd users 1654480 Sep 12 22:49 aplikasi
7 -rw-r--r-- 1 bdpd users    900 Sep 12 22:49 aplikasi.go
8 $ ./aplikasi
```

```
9  Halo bdpd
10 Home anda di /home/bdpd
11 Anda menggunakan Go di /opt/software/go-dev-tools/go/go1.7.3
12 $ file aplikasi
13 aplikasi: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
    statically linked, not stripped
14 $
```

Pustaka / Library / Package

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 (hello) yang menggunakan suatu pustaka yang sebelumnya sudah kita bangun (stringutil/Reverse - untuk membalik kata). Kode sumber diambil dari How to write Go code.

Mengatur Workspace

Ada 3 direktori di workspace yang disiapkan: bin, pkg, dan src: * bin: berisi hasil kompilasi aplikasi (hello) * pkg: berisi hasil kompilasi pustaka (stringutil) * src: kode sumber untuk pustaka serta aplikasi. Pada direktori tersebut, juga dibuat `env.sh`.

```
1  $ ls
2  total 4
3  drwxr-xr-x 1 bdpd users 30 Sep 12 23:16 .
4  drwxr-xr-x 1 bdpd users 56 Sep 12 23:05 ..
5  drwxr-xr-x 1 bdpd users 10 Sep 12 23:16 bin
6  -rw-r--r-- 1 bdpd users 50 Sep 12 23:05 env.sh
7  drwxr-xr-x 1 bdpd users 22 Sep 12 23:16 pkg
8  drwxr-xr-x 1 bdpd users 32 Sep 12 23:07 src
9  $ cat env.sh
10 export GOPATH=`pwd`
11 export PATH=$PATH:$GOPATH/bin
12 $ source ~/env/go/go1.7.3
13 $ source env.sh
14 $
```

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.

Membuat Pustaka

Kode sumber untuk pustaka ini akan diletakkan di `src/github.com/bdp/stringutil`. Paket yang dibuat dengan penamaan ini, nantinya akan diacu dalam `import` sebagai `github.com/bdp/stringutil`.

```
1  /*
2      src/github.com/bdp/stringutil/reverse.go
3      diambil dari https://golang.org/doc/code.html
4  */
5  package stringutil
6
7  // Reverse returns its argument string reversed rune-wise left to right
8  .
9  func Reverse(s string) string {
10     r := []rune(s)
11     for i, j := 0, len(r)-1; i < len(r)/2; i, j = i+1, j-1 {
12         r[i], r[j] = r[j], r[i]
13     }
14     return string(r)
15 }
```

Untuk mengkompilasi:

```
1  $ go build github.com/bdp/stringutil
2  $
```

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.

```
1  /*
2      src/github.com/bdp/hello/hello.go
3      hello.go
4      diambil dari https://golang.org/doc/code.html
```

```
5  */
6  package main
7
8  import (
9      "fmt"
10
11      "github.com/bdp/stringutil"
12  )
13
14  func main() {
15      fmt.Printf(stringutil.Reverse("!oG ,olleH"))
16  }
```

Untuk mengkompilasi:

```
1  $ go install github.com/bdp/hello
2  $ hello
3  Hello, Go!
4  $ ls bin/
5  total 1612
6  drwxr-xr-x 1 bdp users      10 Sep 12 23:16 .
7  drwxr-xr-x 1 bdp users     30 Sep 12 23:16 ..
8  -rwxr-xr-x 1 bdp users 1650409 Sep 12 23:16 hello
9  $
```

Konstruksi Dasar Bahasa Pemrograman Go

Komentar

Bagian komentar dimaksudkan untuk dokumentasi dari *source code*. Ada dua cara untuk memberikan komentar: * Menggunakan `/* ... */` untuk komentar yang meliputi lebih dari satu baris * Menggunakan `//` di awal baris untuk komentar yang meliputi satu baris saja. 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).

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	

Tipe	Arti	Jangkauan
rune	alias dari int32	

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

String

String digunakan untuk men

```

1 package main
2
3 import (
4     "fmt"
5     "reflect"
6     s "strings"
7 )
8
9 // Definisi string
10 var str1 string = "STMIK AKAKOM"
11 var str2 = "Yogyakarta"
12 var str3 = "stmik akakom"
13
14 func main() {
15
16     // Lihat https://golang.org/pkg/strings/
17     fmt.Println(str1)
18     fmt.Println(len(str1))
19     fmt.Println(s.Contains(str1, "AKAKOM"))
20     fmt.Println(s.Title(str3))
21     fmt.Println(str1[2])
22     fmt.Println(s.Join([]string{str1, str2}, " "))

```

```
23     fmt.Println(reflect.TypeOf(str1))
24     fmt.Println(reflect.TypeOf(str2))
25     fmt.Println()
26
27 }
```

Hasil:

```
1 $ go run contoh-string.go
2 STMIK AKAKOM
3 12
4 true
5 Stmik Akakom
6 77
7 STMIK AKAKOM Yogyakarta
8 string
9 string
```

Boolean

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

```
1 package main
2
3 import "fmt"
4
5 var hasilPerbandingan bool
6 var angka1 uint8 = 21
7 var angka2 uint8 = 17
8
9 func main() {
10     hasilPerbandingan = angka1 < angka2
11     fmt.Println(hasilPerbandingan)
12 }
```

Hasil:

```
1 $ go run boolean.go
2 false
```

Variabel

```
1 // nilai-default-variabel.go
2 package main
3
4 import "fmt"
5
6 func main() {
7
8     // unsigned-integer
9     var defUint8 uint8
10    var defUint16 uint16
11    var defUint32 uint32
12    var defUint64 uint64
13    var defUint uint
14
15    // signed-integer
16    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:

```
1  $ go run nilai-default-variabel.go
2
3  Nilai default untuk uint8 =  0
4  Nilai default untuk uint16 = 0
5  Nilai default untuk uint32 = 0
6  Nilai default untuk uint64 = 0
7  Nilai default untuk uint =  0
8
9  Nilai default untuk int8 =  0
10 Nilai default untuk int16 = 0
11 Nilai default untuk int32 = 0
12 Nilai default untuk int63 = 0
13 Nilai default untuk int =  0
14
15 Nilai default untuk string =
16
17 Nilai default untuk float32 = 0
18 Nilai default untuk float64 = 0
19
20 Nilai default untuk complex64 = (0+0i)
21 Nilai default untuk complex128 = (0+0i)
22
```

```
23 Nilai default untuk byte = 0
24 Nilai default untuk rune = 0
25 $
```

Konstanta

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

Pointer

Konsep *pointer* sebenarnya sudah ada pada bahasa pemrograman lain, khususnya C/C++ (dengan kompleksitas yang lebih tinggi). Suatu *pointer* merupakan suatu nilai yang menunjuk ke suatu nilai lain.

Struktur Kendali

Perulangan dengan **for**

Seleksi Kondisi

Pernyataan **if**

Pernyataan **switch**

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.

5 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).

6 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.

7 Struktur Data Lanjut

Arrays

Slices

Maps

Struct

8 Method

9 Testing

10 Konkurensi