Perkenalan

> Go is an attempt to combine the safety and performance of statically typed languages with the convenience and fun of dynamically typed interpretative languages. — Rob Pike

Persiapan Lingkungan Pengembangan

Sebelum memulai kursus ini, saya pastikan kita memakai editor dan versi SDK go yang sama. Editor yang akan kita pakai adalah:

1. Visual Studio Code atau NeoVim
2. Go 1.17.

Bagian ini akan menuntun instalasi keduanya. Penggunaan editor lain diperbolehkan, tetapi saya tidak akan melayani pertanyaan di luar VSCode dan NeoVim.

Kursus ini ditulis dengan asumsi peserta memakai Sistem operasi destop seperti Windows atau macOS untk yang memakai Linux, diharapkan sudah paham cara memasang perangkat lunaknya sesuai distro masing-masing.

Instalasi SDK Go 1.17

Untuk memasang SDK go, unduh langsung ke https://dl.golang.org. Di situs go sudah ada instruksi untuk instalasi SDK nya. Untuk memeriksa apakah versi go nya sudah sesuai dengan yang kita akan pakai coba perintah sebagai berikut.

```
> go version

go version go1.17 darwin/amd64
```

Docker & Editor

Docker saya gunakan untuk keperluan remote devcontainer di Visual Studio untuk membuat *environment* yang lebih bersih. Untuk memasang docker silahkan buka situs Docker Desktop dan ikuti instruksi instalasinya di sana. Untuk menguji apakah sudah terinstalasi, buka aplikasi terminal dan kemudian ketik perintah sebagai berikut.

```
> docker version

Client:
 Cloud integration: 1.0.17
 Version:           20.10.8
 API version:       1.41
 Go version:        go1.16.6
 Git commit:        3967b7d
 Built:             Fri Jul 30 19:55:20 2021
 OS/Arch:           darwin/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.8
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.6
  Git commit:       75249d8
  Built:            Fri Jul 30 19:52:10 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.9
  GitCommit:        e25210fe30a0a703442421b0f60afac609f950a3
 runc:
  Version:          1.0.1
  GitCommit:        v1.0.1-0-g4144b63
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

Go Secara Singkat

Dalam subbab kali ini saya akan menerangkan Go secara sangat singkat. Ini diperlukan untuk menyegarkan kembali kemampuan Go sebelum kita lanjut ke kursus rekayasa perangkat lunak.

Go adalah bahasa pemrograman yang dibuat oleh Google. Fokus go adalah *simplicity*.

Hello World

Program pertama yang kita tulis adalah program 'Hello World', tetapi kali ini ada sedikit ekstra karena hello world kita akan memakai karakter Unicode dan Emoji.

hello.go

```
Unresolved directive in training-00.adoc - include::{sourcedir}/hello_unicode.go[]
```

Untuk menjalankan cukup dengan menulis perintah sebagai berikut

```
> go run hello.go

Hello, □□!□
```

Go adalah bahasa yang modern dan mendukung *string* dengan pengkodean UTF-8. Dengan begitu string seperti di atas akan ditampilkan dengan benar. Dalam satu program go minimal harus ada *entry point* yang berupa fungsi `main()` di dalam *package* `main` juga.

Variabel & Tipe Data

Dalam go ada beberapa tipe data. Tipe data yang umum dipakai adalah:

Table 1. Tipe Data Go

<table>
<tr><th>Nama Tipe Data</th><th>Deskripsi</th></tr>
<tr><td>string</td><td>kumpulan karakter</td></tr>
<tr><td>bool</td><td>berisi true atau false</td></tr>
<tr><td>int</td><td rowspan="4">bilangan bulat. bilangan dengan angka di belakang adalah jumlah kapasitas bitnya. Bilangan tipe dengan prefiks u artinya unsigned yang berarti tidak termasuk bilangan negatif. Berikut jangkauan masing-masing tipe data
<table>
<tr><th>Panjang bit</th><th>Jangkauan signed</th><th>Jangkauan unsigned</th></tr>
<tr><td>8 bit</td><td>-127 .. 127</td><td>0 .. 255</td></tr>
<tr><td>16 bit</td><td>-32768 .. 38767</td><td>0 .. 65535</td></tr>
<tr><td>32 bit</td><td>-2147483648 .. 2147483647</td><td>0 .. 4294967295</td></tr>
<tr><td>64 bit</td><td>-9223372036854775808 .. 9223372036854775807</td><td>0 .. 18446744073709551615</td></tr>
</table></td></tr>
<tr><td>int8 int16 int32 int64</td></tr>
<tr><td>uint</td></tr>
<tr><td>uint8 uint16 uint32 uint64</td></tr>
<tr><td>byte</td><td>ukurannya sama dengan uint8 rata-rata dipakai untuk menyimpan data binari</td></tr>
<tr><td>rune</td><td>ukurannya sama dengan uint32 dipakai untuk menyimpan Unicode Code Point masing-masing karakter dalam string</td></tr>
<tr><td>float32</td><td rowspan="2">bilangan pecahan desimal, berdasarkan IEEE 754. float32 terdiri dari 23 bit mantissa, 8 bit eksponen, dan 1 bit tanda. float64 terdiri dari 52-bit mantissa, 8 bit eksponen, dan 1 bit tanda.</td></tr>
<tr><td>float64</td></tr>
</table>

Penggunaan variabel adalah dengan menggunakan kata kunci `var` seperti di bawah ini.

```
var name string
```

Kita juga bisa menginisiasi variabel dengan memakai operator `=`. Operator `=` juga bisa dipakai untuk mengubah nilai sebuah variabel

```
// Inisialisasi nama sbg 'Brett'
var name string = "Brett"

// Diubah jadi 'Dylan'
name = "Dylan"
```

Cara lain menginisialisasi variabel adalah dengan menggunakan operator `:=`

```
name := "Brett"
```

Program pendek yang menunjukkan pemakaian variabel sebagai berikut.

ex01_var.go

```
Unresolved directive in training-00.adoc - include::{sourcedir}/ex01_var.go[]
```

Ketika dijalankan hasilnya adalah sebagai berikut:

```
> go run ex01_var.go

Waktu sekarang sejak 1 Januari 1970:  1630410  detik
Nama: Brett, umur 30, berat 58.77
```

Untuk keterangan dari package `fmt` dan `time` bisa dilihat di dokumentasinya di dokumentasi go.

Modul, Package, dan Kompilasi

Beberapa berkas kode sumber go dapat dikumpulkan dalam satu *package*. *Package* utama adalah package `main`. Mulai Go 1.13, dependensi dalam satu proyek go didefinisikan dalam satu module dengan fitur `go module`. Semua program dan pustaka berada di dalam `go module`.

Membuat module dan package

Kita akan coba membuat satu directory dan menginisialisasi modulnya.

```
> mkdir berhitung
> go mod init berhitung
```

Di dalam direktori, nanti akan ada berkas `go.mod` yang merupakan daftar modul dan

dependensinya. Karena kita tidak memakai dependensi apapun saat ini isinya akan sangat sederhana.

berhitung/go.mod

```
Unresolved directive in training-00.adoc - include::{sourcedir}/berhitung/go.mod[]
```

Di dalam direktori tersebut, kita buat lagi sebuah direktori dengan nama `math`. Di sini kita akan simpan semua berkas yang berhubungan dengan hitung-menghitung.

```
> mkdir math
```

Kemudian kita buat sebuah berkas yang isinya seperti di bawah ini. Kode sumber ini akan menjumlahkan semua nomor yang dimasukkan ke parameter fungsi. Untuk pembahasan fungsi akan dibahas di bagian selanjutnya.

berhitung/math/sum.go

```
Unresolved directive in training-00.adoc -
include::{sourcedir}/berhitung/math/sum.go[]
```

Nama fungsi `Sum` dimulai dengan huruf besar karena fungsinya kita ekspor. Baris pertama menunjukkan nama `package` tempat kode sumber ini berada. `math` adalah nama *package* yang baru saja kita buat. Di sini kita bisa menyimpan semua berkas dengan peran atau fungsi yang serupa.

Di direktori `berhitung` kita simpan dulu berkas `main.go`. Berkas ini yang akan mengimpor *package* `math`.

berhitung/main.go

```
Unresolved directive in training-00.adoc - include::{sourcedir}/berhitung/main.go[]
```

Baris nomor 2 adalah cara mengimpor modul `math` yang ada di dalam modul `berhitung`. Kemudian fungsi `Sum` dipanggil dengan menggunakan `math` sebagai prefiksnya.

Menjalankan dan mengkompilasi modul

Modul `berhitung` kita bisa dijalankan dengan menggunakan `go run` seperti di bawah ini:

```
> go run .
```

Tanda titik `.` artinya kompilator akan mencari fungsi `main` di dalam modul dalam direktori ini dan menjalankannya. Go adalah bahasa yang dikompilasi menjadi bahasa mesin. Kode sumber akan dikompilasi menjadi bahasa mesin berupa berkas biner yang bisa dieksekusi. Caranya adalah sebagai berikut:

```
> go build -o berhitung .
```

Berkas biner yang dihasilkan dari proses kompilasi ini bisa dijalankan di dalam **sistem operasi dan prosesor yang sama** dengan tempat kita menulis kodenya.

```
> ls berhitung
-rwxr-xr-x  1 lynxluna  staff  1869504 Aug 31 22:54 berhitung

> file berhitung
berhitung: Mach-O 64-bit x86_64 executable
```

Ini artinya berkas `berhitung` adalah berkas yang bisa dieksekusi berformat Mach-O. Format ini adalah format *executable* untuk sistem operasi macOS. Kita bisa mengkompilasi untuk target lain seperti Windows atau Linux. Ini yang disebut dengan *cross compilation*. Caranya adalah dengan mengeset `GOOS` dan `GOARCH`.

```
> GOOS=linux GOARCH=amd64 go build -o berhitung-linux .

> ls berhitung-linux
-rwxr-xr-x  1 lynxluna  staff  1770713 Aug 31 23:05 berhitung-linux

> file berhitung-linux
berhitung-linux: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically
linked, Go
BuildID=2-H-qQg6rZJY935TLO5y/LroB_qh-
tIdrun7Y9dan/s1GqZmGQD1kV0KPrMfHu/zXA6vJD_JOpO5JBjN5hk, not
stripped
```

Berkas ini bisa kemudian bisa disalin ke server dan dijalankan.

Fungsi

Fungsi adalah sebagian kode yang kita panggil berkali-kali. Fungsi dalam go sama saja dengan fungsi dalam bahasa pemrograman lain. Di bagian pembuatan module dan package di atas kita sudah punya fungsi `Sum`.

berhitung/math/sum.go

```
Unresolved directive in training-00.adoc -
include::{sourcedir}/berhitung/math/sum.go[]
```

Untuk membuat fungsi kita memakai kata kunci `func` diikuti nama fungsi dan nilai kembalian. Untuk mengembalikan nilai, kita memakai kata kunci `return`.

Array dan Slice

Baik *array* maupun *slice* menyimpan sebuah kumpulan nilai dengan tipe data tertentu. *Array* mempunyai ukuran **tetap** sementara *slice* mempunyai ukuran yang bisa berubah. Indeks dalam bahasa go dimulai dari `0`. Berikut contoh membuat dan menginisialisasi *array*.

```
// deklarasi - assign
var fruits [3]string

fruits[0] = "Orange"
fruits[1] = "Rambutan"
fruits[2] = "Mango"

// inisialisasi dengan ukuran yang yang diketahui
cars := [4]string{"Honda", "Toyota", "Peugeot", "Vauxhall"}

// biarkan compiler menghitung ukurannya
brands := [...]string{"Dell", "Apple", "IBM"}
```

Untuk slice, ukurannya belum diketahui ketika program dimulai, bisa naik dan bisa turun. Ada beberapa cara untuk menginisalisasi dan menggunakan slice sebagai berikut

```
// deklarasi
var drinks []string

// pembuatan
drinks = make([]string, 3)

// assignment
drinks[0] = "Coke"
drinks[1] = "Pepsi"
drinks[2] = "7Up"

// inisialisasi langsung
elements := []string{"Aluminium", "Copper", "Iron", "Silver"}

// mengubah array menjadi slice
brandSlice := brands[:]

// menambahkan nilai dalam slice
drinks = append(drink, "Pocari Sweat")

// menampilkan subslice
fmt.Printf("%+v\n", elements[1:])  // 1 dan seterusnya [Copper Iron Silver]
fmt.Printf("%+v\n", elements[:3])  // sebelum indeks 3 (0-2) [Alumunium Copper Iron]
fmt.Printf("%+v\n", elements[1:3]) // indeks 1..2 [Copper Iron]
```

Percabangan

Percabangan dalam Go ada dua cara yaitu dengan `if` dan `switch`. Kata kunci `if` mengecek satu kondisi, sedangkan `switch` beberapa kondisi sekaligus.

```
x := 5
y := 20

// if - else
if x < y {
  fmt.Printf("%d lebih kecil daripada %d", x, y)
} else {
  fmt.Printf("%d lebih besar atau sama dengan %d", x, y)
}

// if - else if - else

if x < y {
  fmt.Printf("%d lebih kecil daripada %d", x, y)
} else if x == y {
  fmt.Printf("%d sama dengan %d", x, y)
} else {
  fmt.Printf("%d lebih besar daripada %d", x, y)
}

// switch

ticker:= "AAPL"

switch(brands) {
case "AAPL":
  fmt.Println("Apple")
case "GOOG":
  fmt.Println("Google")
case "BUKA":
  fmt.Println("Bukalapak")
default:
  fmt.Println("Tidak tahu")
}
```

Perulangan

Perulangan dalam go dilakukan dengan menggunakan kata kunci `for`. For menerima kondisi, kata kunci `range` atau kosong sama sekali untuk *infinite loop*. Go **hanya mempunyai** perulangan `for` saja.

```
const n = 10
var nums [n]int

// cara panjang
i := 0

for i < n {
  nums[i] = i * 10
  i++
}

// cara pendek
for i:=0; i < n; i++ {
  nums[i] = i * 10
}

// infinite loop dengan break
i := 0

for {
  nums[i] = i * 10
  i++
  if i > n {
    break
  }
}
```

Map

Map adalah pasangan kunci dan nilai (*key value pairs*). Untuk setiap kunci unik dalam map dia mengandung sebuah nilai.

```
// deklarasi map dgn kunci string dan nilai string
var emails map[string]string

// inisialialisasi map jd siap dipakai
emails = make(map[string]string)

// deklarasi dan inisialisasi sekaligus
emails = map[string]string {
  "Fred":"f@red.com",
  "Gord":"gord@some.bro",
  "Matt":"matt@geemail.com",
  "Reed":"ree@domain.me",
}

// mengakses map
d = emails["Fred"]

// mengakses dan mengecek apakah ada di map
d, ok = emails["Xyla"] // d = "", ok = false

d, ok = emails["Matt"] // d = "matt@geemail.com", ok = true
```

Range

Range digunakan untuk mengulang *array*, *slice*, atau *map*. Kata kunci `range` akan menghasilan pasangan (indeks, nilai) untuk *array* dan *slice*, sementara untuk *map* akan mengembalikan (kunci, nilai).

```
// Slice atau Array
names := []string{"Brett", "Brad", "Xyla", "Park", "Chaz"}

// Pakai indeks dan nilai
for idx, name := range names {
  fmt.Printlf("%d - %s\n", idx + 1, name)
}

// Pakai nilai saja
for _, name := range names {
  fmt.Println(name)
}

// Untuk map
for k, v := range emails {
  fmt.Printf("Email '%s' adalah <%s>\n", k, v)
}
```

Pointer

Pointer mengandung alamat memori dari satu nilai atau variabel. Pointer dalam Go tidak berbeda dengan C. Perbedaan mendasarnya adalah: dalam go, tidak bisa melakukan *pointer aritmethics*.

```
a := 500
b := &a

fmt.Println(a, b) // b -> nilai heksadesimal alamat a

fmt.Printf("a: %T b: %T\n", a, b) // a int, b *int.

// Karena b mengarah ke alamat a maka, mengubah b sama dengan mengubah a
*b = 200  // Dereferensi pointer b

fmt.Printf("Nilai a sekarang = %d\n", a)
```

Closure

Go mengizinkan fungsi anonim dan fungsi yang disimpan dalam variabel. Kita bisa mendefinisikan fungsi tanpa menamainya.

```
Unresolved directive in training-00.adoc - include::{sourcedir}/closure/main.go[]
```

Fungsi `accum` mengembalikan nilai bertipe `func (int) int` yang artinya "sebuah fungsi yang menerima *integer* sebagai parameter dan mengembalikan *integer* lagi sebagai nilai kembaliannya.

Variabel `accumulate` tipenya adalah `func (int) int` dan menerima hasil kembalian dari fungsi `accum`. Nilai `accumulator` hasil kembalian dari fungsi `accum` tidak berubah.

TIP

Memoisation

Cara menyimpan `accumulator` seperti di atas disebut **Memoisation**. Memoisation (AS: Memoization) adalah cara menyimpan sementara hasil kalkulasi sebelumnya untuk dilanjutkan ke kalkulasi selanjutnya.

Struct

Kita bisa mengumpulkan data-data yang berkaitan satu sama lain dalam satu **struktur**. Untuk membuat struktur, kata kunci yang dipakai adalah `struct`.

Untuk membuat struktur, kita bisa mendefinisikan tipe seperti ini:

```
type Person struct {
  Name  string
  Age   int
}
```

Struktur juga bisa dibuat anonim dan bersarang (*nested*)

```
type Person struct {
  Name  string
  Age   int
  Address struct {
    PostalCode  string
    State       [2]rune
  }
}
```

Struktur bisa diinisalisasi layaknya *array* dan *slice*. Struct juga bisa dibuat inline

```
// inisalisasi
me := Person{
  Name: "Brad",
  Age: 44,
  Address: {
    PostalCode: "77561",
    State: {'I','L'},
  },
}

// deklarasi dan inisalisasi
drink := struct {
  Brand string
  Capacity  int
}{"Coke", 500}
```

WARNING

Struktur tidak ada hubungannya dengan PBO

Banyak buku dan artikel di luar yang mendefinisikan struktur sebagai kumpulan data **dan method** atau bahkan menyamakannya dengan kata kunci `class` di bahasa lain seperti Java. Definisi ini tidak tepat. Go bukan bahasa yang berbasis *Class*.

Ada juga yang berpendapat jika Go bukan bahasa berorientasi objek. Ini lebih salah lagi, karena syarat orientasi objek bukanlah keberadaan `class` atau method. Struktur di Go bukanlah *Class* dan metode di go punya *behaviour* yang berbeda dengan metode dalam bahasa berbasis *Class*.

Metode

Metode adalah fungsi dengan argumen spesial yang dinamakan *receiver*. Semua tipe bisa mempunyai metode, termasuk alias dari tipe bawaan.

```
// Vector menunjukkan arah dan panjang
type Vector struct {
  x, y, z float64
}

// Panjang vektor = √(x²+y²+z²)
func (v Vector) Magnitude() float64 {
  return math.Sqrt(x*x+y*y+z*z)
}

// Normal vektor adalah vektor dengan panjang 1
func (v *Vector) Normalise() {
  d = 1.0/v.Magnitude()

  v.x *= d
  v.y *= d
  v.z *= d
}

// OnOff adalah alias dari Bool
type OnOff bool

// Mengubah onoff menjadi string
func (b OnOff) String() string {
  if(!b) {
    return "off"
  }

  return "on"
}
```

Seperti dalam kode di atas, ada dua jenis *receiver* yaitu *value receiver* dan *pointer receiver*. Secara singkat, kita memakai *value receiver* sebanyak mungkin sampai kita perlu memakai *pointer receiver*. Kasus umum pemakaian *pointer receiver* adalah ketika kita ingin mengubah nilai dari *receiver*nya.

Interface

Interface atau antar muka adalah kumpulan *method signature*. Sebuah variabel dengan tipe `interface` bisa menyimpan nilai yang mengimplementasikan **seluruh** metode yang didefinisikan dalam *interface* tersebut.

Contoh yang akan saya tulis adalah contoh klasik: Greeter. Dalam contoh ini saya akan menunjukkan kalau *interface* dalam bahasa go **sangat berbeda** dengan bahasa lain yang berbasis *class*. *Interface* dalam go bersifat implisit yang artinya semua tipe yang mengandung nama fungsi dengan *signature* yang sama artinya **sudah mengimplementasikan** *interface* tersebut. Dengan demikian, umumnya *interface* didefinisikan di belakang dan merupakan evolusi dari beberapa implementasi yang mirip.

Anggap awalnya kita hanya punya Greeter yang berbahasa Indonesia

```
type IDGreeter struct {}

func (g IDGreeter) Greet(){
  fmt.Println("Halo Semua!")
}

func main() {
  greeter := IDGreeter{}
  greeter.Greet()
}
```

Lalu, kita punya Greeter yang bisa berbahasa Inggris

```
type IDGreeter struct{}

func (g IDGreeter) Greet() {
  fmt.Println("Halo Semua!")
}

type ENGreeter struct{}

func (g ENGreeter) Greet() {
  fmt.Println("Hello, Everybody!")
}

func main() {
  idGreeter := IDGreeter{}
  enGreeter := ENGreeter{}

  idGreeter.Greet()
  enGreeter.Greet()
}
```

Karena kita tau kalau kedua greeter tersebut mempunyai method `Greet()` dengan *signature* yang sama. Kita bisa menyimpan array dari dua objek tersebut. Jadi kita definisikan `Greeter` setelah tahu kalau kita perlu dua atau lebih tipe dengan *behaviour* yang mirip.

```
type IDGreeter struct{}

func (g IDGreeter) Greet() {
  fmt.Println("Halo Semua!")
}

type ENGreeter struct{}

func (g ENGreeter) Greet() {
  fmt.Println("Hello, Everybody!")
}

type Greeter interface {
  Greet()
}

func main() {
  greeters := []Greeter{ IDGreeter{}, ENGreeter{} }

  for _, greeter := range greeters {
    greeter.Greet()
  }
}
```

Kemudian karena kita ingin menambahkan *behaviour* lainnya, supaya bisa bilang selamat pagi. Kita definisikan interface lain. Misalnya, *morning greeter*. Kebetulan orang greeter versi Indonesia tidak suka bangun pagi.

```
type IDGreeter struct{}

func (g IDGreeter) Greet() {
  fmt.Println("Halo Semua!")
}

type ENGreeter struct{}

func (g ENGreeter) Greet() {
  fmt.Println("Hello, Everybody!")
}

func (g ENGreeter) GoodMorning() {
  fmt.Println("Good morning, everybody!")
}

type DutchGreeter struct{}

func (g DutchGreeter) GoodMorning() {
  fmt.Println("Gutten Morgen")
}

type Greeter interface {
  Greet()
}

type MorningGreeter interface {
  GoodMorning()
}

func main() {
  greeters := []Greeter{ IDGreeter{}, ENGreeter{} }

  for _, greeter := range greeters {
    greeter.Greet()
  }

  morningGreeters := []MorningGreeter{EnGreeter{}, DutchGreeter{}}

  for _, mg := range morningGreeters {
    mg.GoodMorning()
  }
}
```

Saya menulis panjang lebar tentang *interface* di sini karena prihatin karena di internet, terutama artikel-artikel berbahasa Indonesia banyak yg memakai *interface* ini layaknya bahasa lain yang merupakan bahasa eksplisit.

Web & Web API

Go sudah mempunyai fitur bawaan untuk menangani Web dan Web API. Saya tidak akan membahas terlalu banyak, saya hanya akan membahas singkat, karena Web API akan kita dalami di bab-bab selanjutnya.

Semua hal yang berbau jaringan, ditangani dengan *package* `net`. Karena HTTP adalah jenis *server* yang paling banyak dipakai, Go menyediakan *package* `net/http`. Abstraksi `net/http` cukup masuk akal dan mudah digunakan. Di dalamnya sudah ada HTTP handler, HTTP Server, HTTP Client, dan juga pengujian HTTP.

Path dalam `net/http` ditangani oleh *interface* `Handler` dengan definisi sebagai berikut:

```
type Handler interface {
    ServeHTTP(http.ResponseWriter, *http.Request)
}
```

Yang artinya semua tipe yang mengimplementasikan *interface* di atas akan dipanggil. Pustaka standar go menyediakan implementasi sederhana dari *interface* ini yaitu `http.ServeMux` yang bisa memilih handler berdasarkan path.

```
import "net/http"

func main() {
    mux := http.NewServeMux()

    mux.HandleFunc("/hello", func(w http.ResponseWriter, r *http.Request) {
      if r.Method != http.MethodGet {
        w.WriteHeader(http.StatusMethodNotAllowed)
        return
      }

      w.WriteHeader(http.StatusOK)
      w.Write([]byte("Hello, world!")
    })

    http.ListenAndServe(":3000", mux)
}
```

Seperti terlihat di atas, `http.ServeMux` fiturnya sangat sederhana. Untuk mengambil parameter dalam path atau untuk metode HTTP tertentu, kode yang ditulis cukup panjang. Karenanya biasanya, pengguna Go memakai router seperti `go-chi/chi` atau `gorilla/mux` untuk mempermudah menangani *request* HTTP dan juga untuk membuat *filter* dan *middleware* menjadi lebih mudah.

Dengan menggunakan `chi`, kode di atas bisa diubah menjadi sebagai berikut:

```
Unresolved directive in training-00.adoc - include::{sourcedir}/webserver/main.go[]
```

Kodenya memang lebih panjang, tetapi gunanya lebih banyak. Pertama tidak perlu ada *parsing* untuk path untuk mencocokkan handler mana untuk path mana. Selain itu, kita juga punya *middleware* yang akan mengolah *header* untuk mendapatkan alamat IP, membuatkan RequestID, dan menuliskan *log* ke terminal untuk setiap *request*.

Latihan

1. Buat program dengan Go untuk kasus FizzBuzz, di mana jika suatu bilangan yang dimasukkan ke terminal bisa dibagi 3 maka akan ditulis "Fizz", sementara kalo bias dibagi lima maka ditulis "Buzz" dan bila bisa dibagi 3 **dan** 5 akan ditulis "FizzBuzz".

Opsional

1. Buat program dengan Go untuk menampilkan `n` bilangan fibonacci pertama.
2. Buat program dengan Go untuk kasus *stack* dan *queue* menggunakan array atau slice. Gunakan fungsi `push` dan `pop` untuk *stack* dan `enqueue` dan `dequeue` untuk *queue*.