



Assignment – A01d

# Introduction to Go Programming Language

**Penulis** : NT

**Versi** : 4 (20240904-2215)



## Riwayat Versi

Versi	Tanggal dan Waktu	Halaman	Perubahan
<b>1</b>	20240826-0800	Semua	Rilis pertama
<b>2</b>	20240828-1437	37	Mengubah ketentuan pengembalian pesan sukses pada function AddTransaction
<b>3</b>	20240831-1420	36 35, 42 42	Mengubah isi pesan eror pada function AddItem, pesan eror dan sukses pada function AddMember.  Menambahkan informasi untuk mengunduh template sumber kode.  Mengubah input pada testcase 3. Awal: ADD_TRANSACTION 15 MR134 MM921 Menjadi: ADD_TRANSACTION 15 MR134 MM921
<b>4</b>	20240904-2215	37	Mengubah ketentuan pengembalian nilai pada function GetTransactionMember dan GetTransactionItem

## Daftar Isi

---

Riwayat Versi .....	2
Daftar Isi.....	3
Informasi Umum .....	5
Ekspektasi Hasil Pembelajaran .....	5
⭐ Prasyarat .....	5
Deskripsi .....	5
Golang Installation.....	6
Golang Installation in Windows.....	6
Golang Installation in Mac OS.....	7
Golang Installation in Linux.....	8
Golang Commands.....	9
First Program with Golang .....	10
Working with Data.....	11
Data Type.....	11
Variable.....	12
Casting.....	13
Conditional Statement.....	14
Basic Condition .....	14
Condition With Short Statement .....	14
Switch Case .....	14
Looping.....	15
Pointer .....	16
Array.....	18
Slice .....	19
Map .....	20
Function.....	21
Struct .....	23
Method.....	24
Interface.....	26
Basic Interface .....	26

Empty Interface .....	27
Public dan Private .....	28
Error Handling .....	28
Basic Error .....	29
Panic .....	29
Recover .....	30
Basic I/O.....	30
Basic Input.....	30
Basic Output.....	31
Package Manager .....	32
Goroutine.....	33
❖ Spesifikasi .....	35
Studi Kasus.....	35
Asumsi dan Kesepakatan.....	42
Komponen Penilaian .....	42
☒ Informasi Pengumpulan Berkas.....	42
⚖ Peraturan .....	43
Keterlambatan.....	43
Plagiarisme.....	43

## Assignment – A01d

# Introduction to Go Programming Language

## 🔍 Informasi Umum

---

<b>Tipe Tugas</b>	: Individu
<b>Batas Waktu Pengumpulan</b>	: Jumat, 06 September 2024 pukul 17.00 WIB (SCeLE)
<b>Format Penamaan Berkas</b>	:
- <b>Laporan</b>	: A01_[NPM].pdf (Contoh: A01_2206000111.pdf)
- <b>Berkas Lainnya</b>	: A01_[NPM]_model.go A01_[NPM]_command.go A01_[NPM]_main.go
<b>Tautan Kerangka Laporan</b>	: <a href="#">Klik Di Sini</a>
<b>Tautan Kerangka Kode</b>	: <a href="#">Klik Di Sini</a>

## 🏁 Ekspektasi Hasil Pembelajaran

Setelah mengerjakan penugasan ini, mahasiswa diharapkan dapat:

1. memahami dengan baik dan mampu mengimplementasikan (C3) bahasa pemrograman Go dalam pembuatan program.

## ◆ Prasyarat

- 
1. Peserta sudah mengulas kembali materi pembelajaran yang sudah diajarkan pada mata kuliah Dasar-Dasar Pemrograman (*Data Type, Condition, Loop, Function*, dan sebagainya).
  2. Peserta sudah mengulas kembali materi pembelajaran yang sudah diajarkan pada mata kuliah Sistem Operasi (*Linux/Bash Command, Pointer, Multithread*, dan sebagainya).
  3. Peserta sudah menginstal *Integrated Development Environment* (IDE) yang kompatibel dengan bahasa pemrograman Golang (VS Code dengan ekstensi “Go”, Goland, dan sebagainya).

## 📖 Deskripsi



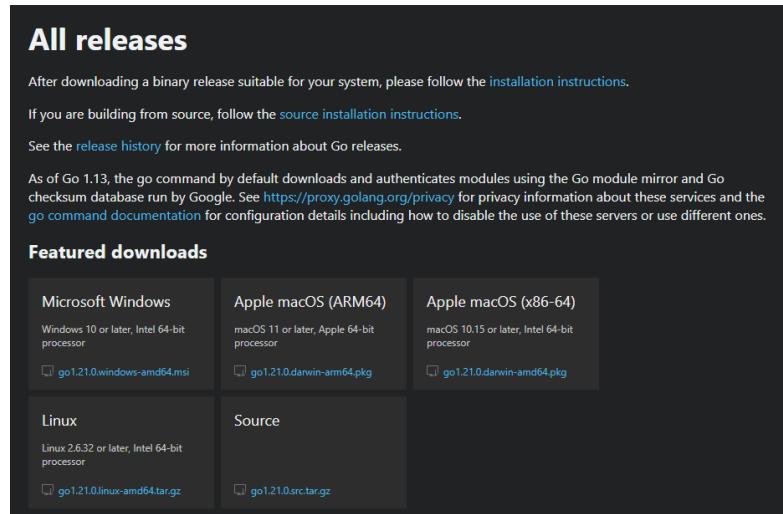
Golang (Source: [content.techgig.com](http://content.techgig.com))

Go atau Golang merupakan bahasa pemrograman yang dibuat di Google yang oleh Rob Pike, Robert Griesemer, dan Ken Thompson. Untuk saat ini, bahasa pemrograman Golang sudah sering banyak digunakan oleh tim pengembang karena mudah dipelajari, memiliki sintaks yang sederhana, dan banyaknya dukungan oleh komunitas yang mengembangkan *third-party packages* yang bersifat *open source* sehingga dapat digunakan oleh tim pengembang.

Pada mata kuliah Jaringan Komputer dan Jaringan Komunikasi Data, bahasa pemrograman Golang akan digunakan untuk mengerjakan beberapa tugas yang berkaitan dengan pengembangan *socket programming* dan *security*. Diharapkan setelah peserta telah mengerjakan tugas ini, peserta sudah mampu untuk mengembangkan sebuah program sederhana menggunakan bahasa pemrograman golang dengan baik dan bisa digunakan.

## Golang Installation

Untuk menginstal Golang pada perangkat komputer anda, anda dapat mengunduh berkas pada tautan [berikut](#) (Versi saat ini 1.23.0) yang sudah disesuaikan dengan sistem operasi yang anda gunakan.

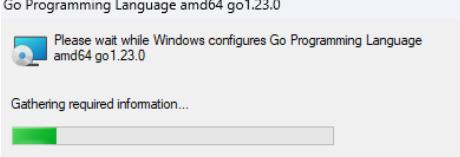
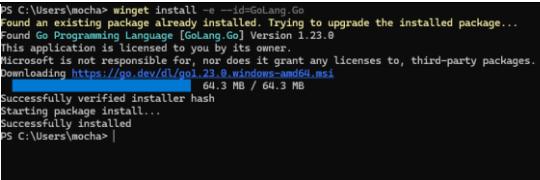


Setelah mengunduh berkas tersebut, silakan ikuti tahapan instalasi yang sesuai dengan sistem operasi yang anda gunakan.

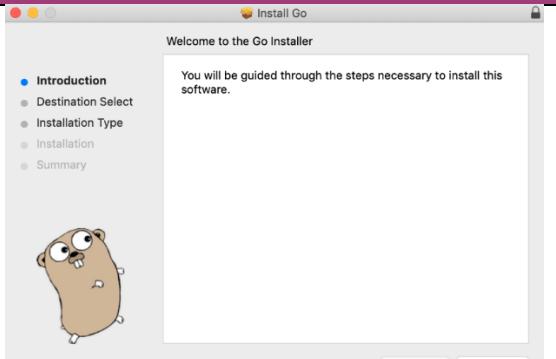
## Golang Installation in Windows (via Winget)

Terdapat cara yang dapat Anda lakukan melakukan instalasi golang pada sistem operasi Windows, namun pada tutorial ini Anda akan mencobanya menggunakan instalasi dari winget.

Step	Aktivitas	Contoh Screenshot
1.	Buka terminal (disarankan menggunakan <i>powershell</i> ), kemudian jalankan perintah berikut untuk mengunduh berkas dari winget.	<pre>PS C:\Users\mocha&gt; winget install --id=GoLang.Go Found an existing package already installed. Trying to upgrade the installed package... Found Go Programming Language [GoLang.Go] Version 1.23.0 This application is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages. Downloading https://go.dev/dl/goi.23.0.windows-amd64.msi 64.3 MB / 64.3 MB Successfully verified installer hash Starting package install...</pre>

	<pre>winget install -e -- id=GoLang.Go</pre> <p>Note: Ketik Y untuk melanjutkan instalasi jika sebelumnya Anda belum pernah menggunakan winget.</p>	
2.	Akan muncul <i>pop up</i> untuk memberikan akses kepada Administrator sehingga Anda perlu menekan tombol Yes sehingga akan memunculkan pesan “Starting package install...” yang menandakan instalasi akan dimulai.	
3.	Jika instalasi selesai maka akan memunculkan pesan “Successfully installed”	

## Golang Installation in Mac OS

Step	Aktivitas	Contoh Screenshot
1.	Buka berkas instalasi yang telah diunduh.	
2.	Tekan “Next” kemudian klik “Install” untuk memulai proses instalasi.	

3.	Setelah proses instalasi selesai, tekan “Finish” untuk menutup program.	
4.	Buatlah folder dengan nama “go” pada direktori dokumen atau lainnya untuk membuat go workspace.	
5.	<p>Buka terminal dan jalankan perintah berikut.</p> <pre>cd ~ echo "export GOPATH=/Users/...../go" &gt;&gt; .bash_profile</pre>	
6.	<p>Jalankan perintah berikut untuk mengecek go workspace sudah dibuat</p> <pre>echo \$GOPATH</pre>	

## Golang Installation in Linux

Step	Aktivitas	Contoh Screenshot
1.	<p>Hapus Golang yang sudah diisntal sebelumnya dengan menjalankan perintah berikut.</p> <pre>rm -rf /usr/local/go</pre> <p>Ekstrak berkas yang telah diunduh dengan menjalankan perintah berikut.</p> <pre>sudo tar -xvf [nama berkas yang telah diunduh]</pre>	<pre>naufal@Imai-Lisa:~\$ sudo tar -xvf go1.21.0.linux-amd64.tar.gz naufal@Imai-Lisa:~\$ ls dump.rdb go go1.21.0.linux-amd64.tar.gz go1.21.0.linux-amd64.tar.gz:Zone.Identifier naufal@Imai-Lisa:~\$</pre>

2.	Pindahkan folder go ke sistem dengan menjalankan perintah berikut.  sudo mv go /usr/local	<pre>naufal@Imai-Lisa:~\$ sudo mv go /usr/local naufal@Imai-Lisa:~\$  </pre>
3.	Tambahkan perintah berikut pada berkas .profile (umumnya berada di direktori home) untuk melakukan setup go workspace.  export GOROOT=/usr/local/go export GOPATH=\$HOME/go export PATH=\$GOPATH/bin:\$GOROOT/bin:\$PATH	<pre># set GOPATH export GOROOT=/usr/local/go export GOPATH=\$HOME/go export PATH=\$GOPATH/bin:\$GOROOT/bin:\$PATH</pre>
4.	Lakukan <i>re-load</i> kembali terminal atau jalankan perintah berikut.  Source ~/.profile	<pre>naufal@Imai-Lisa:~\$ source ~/.profile</pre>

Untuk mengecek Golang sudah terinstal pada perangkat komputer anda silakan jalankan perintah berikut.

```
go version
```

```
PS C:\Users\mocha> go version
go version go1.23.0 windows/amd64
```

## Golang Commands

Terdapat perintah-perintah pada pemrograman Golang yang dapat anda gunakan selama pengembangan program.

No.	Command	Penjelasan	Contoh Screenshot
1.	go mod init [module-name]	Melakukan inisialisasi sebuah modul untuk membuat proyek Golang yang secara default akan dibuatkan berkas yaitu go.mod secara otomatis.	<pre>naufal@Imai-Lisa:~/jarkomdat\$ go mod init latihan go: creating new go.mod: module latihan naufal@Imai-Lisa:~/jarkomdat\$ ls go.mod</pre>

2.	go get -u [module-name]	Mengunduh modul <i>package third-party</i> pada proyek yang kemudian akan disimpan pada berkas go.mod dan go.sum.	<pre>naufal@Imai-Lisa:~/jarkomdat\$ go get github.com/labstack/echo/v4 go: downloading github.com/labstack/echo/v4 v4.11.1 go: downloading github.com/labstack/echo v3.3.10+incompatible go: downloading github.com/labstack/gommon v0.4.0 go: downloading golang.org/x/crypto v0.11.0 go: downloading golang.org/x/net v0.12.0 go: downloading github.com/valyala/fasttemplate v1.2.2 go: downloading github.com/mattn/go-colorable v0.1.13 go: downloading github.com/mattn/go-isatty v0.0.19 go: downloading golang.org/x/text v0.11.0</pre>
3.	go run [file-name].go	Mengeksekusi program Golang berkestensi .go	<pre>naufal@Imai-Lisa:~/jarkomdat\$ go run main.go Selamat datang di Jarkom x Jarkomdat Gasal 2023/2024</pre>
4.	go build [file-name].go  ./[file-name]	Me-compile program menjadi berkas tanpa ekstensi (pada Windows akan berekstensi .exe) dan berkas tersebut dapat dieksekusi secara langsung.	<pre>naufal@Imai-Lisa:~/jarkomdat\$ go build main.go naufal@Imai-Lisa:~/jarkomdat\$ ls go.mod go.sum main main.go naufal@Imai-Lisa:~/jarkomdat\$ ./main Selamat datang di Jarkom x Jarkomdat Gasal 2023/2024</pre>

## First Program with Golang

Berikut merupakan contoh program Golang sederhana yang akan mencetak “Selamat datang di Jarkom x Jarkomdat Gasal 2023/2024” yang dijalankan pada berkas berekstensi Go.

Input	Output	Penjelasan
<pre>package main //1  import "fmt" //2  func main() { //3     fmt.Println("Selamat     dating di Jarkom x     Jarkomdat Gasal     2024/2025") }  /* //4    this is comment.    :) */</pre>	Selamat datang di Jarkom x Jarkomdat Gasal 2024/2025	<ol style="list-style-type: none"> <li>Untuk berkas yang diindikasi sebagai program utama yang akan dijalankan diharuskan menambahkan <i>package “main”</i>.</li> <li>Syntax “import” digunakan untuk me-import method-method dari suatu package.</li> <li>Fungsi “main” diindikasikan sebagai fungsi utama yang akan dijalankan pada program.</li> <li>Untuk memberikan komen dapat menggunakan “//” untuk satu baris dan “/* */” untuk banyak baris.</li> </ol>

## Working with Data

### Data Type

#### String

Terdapat dua jenis notasi *string* yang dapat digunakan pada pemrograman Golang:

```
//method 1 for one line text
"Hello, World"

//method 2 for multiple line text
`

Selamat Datang di
Jarkom x Jarkomdat
Gasal 2024/2025
`
```

Selanjutnya, terdapat tiga fungsi *built-in* yang dapat digunakan untuk mengolah nilai *string*:

Fungsi	Penjelasan
len(string)	Panjang dari nilai <i>string</i> .
string[index]	Mendapatkan nilai karakter dalam bentuk <i>integer</i> berdasarkan posisi <i>index</i> .
string[start_idx:end_idx]	<i>Slicing</i> nilai <i>string</i> menjadi nilai <i>substring</i> .

### Numeric

Terdapat tiga jenis notasi *numeric* yang dapat digunakan pada pemrograman Golang:

Numeric	Syntax	Penjelasan
<i>Integer</i>	int8 to int64, int, rune	Nilai bilangan yang memiliki kapasitas 8 – 64 bit, dapat didefinisikan sebagai <i>int</i> berdasarkan bilangan bit pada OS yang digunakan dan juga bisa didefinisikan sebagai <i>rune</i> yang setara dengan <i>int32</i> .
<i>Unsigned Integer</i>	uint8 to uint64, uint, byte	Nilai bilangan positif yang memiliki kapasitas 8 – 64 bit, dapat didefinisikan sebagai <i>uint</i> berdasarkan bilangan bit pada OS yang digunakan dan juga bisa didefinisikan sebagai <i>byte</i> yang setara dengan <i>uint8</i> .
<i>Float</i>	float32 to float64	Nilai bilangan desimal yang memiliki kapasitas 32 – 64 bit.

Untuk operasi aritmatika pada pemrograman Golang cukup mirip dengan pemrograman Java di mana untuk dua atau lebih nilai harus memiliki tipe data yang sama sehingga jika terdapat tipe data yang berbeda diharuskan untuk melakukan *casting* terlebih dahulu. Selain itu juga, tipe data *numeric* mendukung *Augmented Assignment* dan *Unary Operator*.

Augmented Assignment		Unary Operator	
Syntax	Math Operation	Syntax	Math Operation
a += 10	a = a + 10	a++	a = a + 1
a -= 10	a = a - 10	a--	a = a - 1
a *= 10	a = a * 10	-a	a = -a
a /= 10	a = a / 10	+a	a = +a
a %= 10	a = a % 10		

## Boolean

Tipe data ini didefinisikan sebagai `bool` yang bernilai `true` dan `false`.

## Variable

### Basic & Short Declaration

Perbedaan dasar antara mendeklarasikan suatu variable dengan metode *basic* dan *short declaration* terletak dengan keikutsertaan tipe data yang didefinisikan. Pada *short declaration* tidak perlu mendefinisikan tipe data sehingga akan didefinisikan dengan tipe data *default* berdasarkan nilai yang dimasukkan.

```
//Basic Declaration

//Method 1
var [var_name] [data_type] = [value]

//Method 2
var [var_name] [data_type]
[var_name] = [value]

//Method 3
Var [var_name_1], [var_name_2] [data_type] = [value_1], [value_2]

//Short Declaration
[var_name_1], [var_name_2] := [value_1], [value_2]
```

## Constant Variable

Variabel yang dideklarasikan sebagai konstan diharuskan untuk me-*assign* nilai secara langsung dan tidak dapat diubah.

```
const [var_name] [data_type] := [value]
```

## Underscore Variable

Dalam pemrograman Golang, suatu variabel yang dideklarasikan harus digunakan sehingga jika variabel tersebut tidak digunakan maka dapat dideklarasikan dengan *underscore*.

```
_ := [value]
[_var_name], _ := [value_1], [value_2]
```

## Type

Type dideklarasikan di luar *function* yang digunakan ketika suatu variabel ingin dideklarasikan berdasarkan tipe data yang memiliki referensi ke sebuah type.

```
type [name_type] [data_type]

function [func_name]() {
    var [name_variable] [name_type] = [value]
}
```

## Casting

### Basic Casting

Pada metode ini anda dapat menuliskan *syntax* [data\_type] (value) yang mana direkomendasikan untuk melukan *casting* antar tipe data *numeric* saja. Berikut contoh program sederhana untuk melakukan *casting* antar tipe data *numeric*:

Input	Output
<pre>package main  import (     "fmt"     "reflect" )  func main() {     var intValue = 23     intToFloatValue := float32(intValue)     fmt.Println(intValue)     fmt.Println(intToFloatValue)     fmt.Println(reflect.TypeOf(intValue))     fmt.Println(reflect.TypeOf(intToFloatValue)) }</pre>	<pre>23 23 int float32</pre>

### Methods from strconv Package

*Method-method* yang ada pada package *strconv* secara umum digunakan untuk proses *casting* dari tipe data *string* ke tipe data *numeric* dan *boolean* ataupun kebalikannya. Untuk lebih lengkapnya anda dapat melihat dokumentasi penggunaanya pada tautan [berikut](#).

## Conditional Statement

Dalam penulisan *syntax conditional statement* pada pemrograman Golang memiliki kemiripan dengan pemrograman Java. Terdapat beberapa metode yang dapat digunakan:

### Basic Condition

Input	Output
<pre>package main  import "fmt"  func main() {     finalCourseValue := 75;     if finalCourseValue &gt;= 55 {         fmt.Println("Pass")     } else {         fmt.Println("Not Pass")     } }</pre>	Pass

### Condition With Short Statement

Pada pemrograman Golang anda dapat mendefinisikan sebuah variabel yang digunakan pada *conditional statement*.

Input	Output
<pre>package main  import "fmt"  func main() {     if finalCourseValue := 75; finalCourseValue &gt;= 55     {         fmt.Println("Pass")     } else {         fmt.Println("Not Pass")     } }</pre>	Pass

### Switch Case

Metode ini digunakan untuk membandingkan persamaan nilai pada variabel dengan masing-masing nilai dari setiap *case* yang ada, jika tidak sama maka anda dapat menambahkan *syntax default* sebagai lainnya.

Input	Output
-------	--------

```
package main

import "fmt"

func main() {
    booleanValue := 1
    switch booleanValue {
    case 0:
        fmt.Println(false)
    case 1:
        fmt.Println(true)
    default:
        fmt.Println("invalid")
    }
}
```

true

## Looping

Untuk membuat *syntax looping* pada pemrograman Golang terdapat tiga istilah yang cukup familiar dengan pemrograman Java:

1. *Initial Statement* digunakan sebagai pendanda mulainya iterasi berdasarkan nilai yang *di-assign* pada variabel.
2. *Conditional Statement* digunakan untuk pengecekan dari setiap iterasi yang mana jika bernilai *false* maka iterasi tersebut akan berhenti.
3. *Post Statement* digunakan untuk mengubah nilai variabel yang dideklarasikan pada *initial statement*.

```
for init_state; cond_state; post_state {
    //execute code
}
```

Selain itu juga terdapat dua *syntax* yang dapat digunakan selama proses iterasi:

- `break` untuk menghentikan iterasi secara paksa.
- `continue` untuk melewati iterasi yang sedang berjalan.

Terdapat tiga metode *looping* yang dapat anda gunakan:

Metode	Input	Output
Basic Looping	<pre>for i := 0; i &lt; 3; i++ {     if i%2 == 0 {         fmt.Println("Even")     } else {         fmt.Println("Odd")     } }</pre>	Even Odd Even

Condition Only in For Loop	<pre>i := 0 //init_state for i &lt; 3 { //cond_state     if i%2 == 0 {         fmt.Println("Even")     } else {         fmt.Println("Odd")     }     i++ //post_state }</pre>	Even Odd Even
Empty in For Loop	<pre>i := 0 //init_state for {     if i == 3 { //cond_state         break     }     if i%2 == 0 {         fmt.Println("Even")     } else {         fmt.Println("Odd")     }     i++ } //post_state</pre>	Even Odd Even

## Pointer

Secara umum, setiap nilai yang dimasukkan ke sebuah variabel akan disimpan ke *memory address* yang mana untuk mengakses nilai tersebut anda dapat menggunakan sebuah pointer. Terdapat dua *syntax* yang digunakan pada saat menggunakan pointer:

- `&[var_name]` berfungsi untuk mendapat nilai alamat memori dari sebuah variabel.
- `*[var_name]` berfungsi untuk mendapat nilai yang disimpan pada alamat memori dari sebuah variabel.

Berikut contoh program yang menerapkan konsep pointer:

Input	Output
<pre>package main  import "fmt"  func main() {     x := 3     y := &amp;x     fmt.Println(&amp;x)     fmt.Println(y)     fmt.Println(*y) }</pre>	<pre>0xc00001a098 0xc00001a098 3</pre>

Kemudian terdapat tiga konsep yang perlu dipahami dalam penggunaan pointer:

Konsep	Penjelasan
Pass by Value	Terjadi pada saat ingin menyalin nilai dari satu variabel ke variabel lainnya yang mana kedua variabel tersebut akan memiliki <i>address memory</i> yang berbeda sehingga ketika nilai variabel terjadi perubahan maka nilai pada variabel lainnya tidak akan ikut berubah.
Pass by Reference	Terjadi pada saat ingin menyalin nilai alamat memori dari satu variabel ke variabel lainnya yang mana kedua variabel tersebut akan memiliki <i>memory address</i> yang sama sehingga ketika nilai variabel terjadi perubahan maka nilai pada variabel lainnya juga akan ikut berubah.
Pointer Zero Value	Terjadi ketika mengakses <i>memory address</i> yang tidak didefinisikan sehingga akan menyebabkan eror karena nilai <i>memory address</i> tersebut adalah zero atau nil.

Berikut contoh program yang menerapkan ketiga konsep tersebut:

Pass by Value	
Input	Output
package main  import "fmt"  func main() { x := 3 y := &x fmt.Println(&x) fmt.Println(y) fmt.Println(*y) }	x: 3 y: 4
Pass by Reference	
Input	Output
package main  import "fmt"  func main() { x := 3 y := &x *y++ fmt.Println("x:", x) fmt.Println("y:", *y) }	x: 4 y: 4
Pointer Zero Value	
Input	Output
package main	panic: runtime error: invalid memory address or

```
import "fmt"

func main() {
    var x *int
    *x = 3
    fmt.Println(*x)
}
```

nil pointer  
dereference  
[signal 0xc0000005  
code=0x1 addr=0x0  
pc=0x85e496]

## Array

Terdapat beberapa metode untuk mendeklarasikan sebuah array pada pemrograman Golang:

```
//Method 1
var array_name[length_array]data_type
array_name := {data_1, data_2, ...., data_n}

//Method 2
array_name := [length_array]data_type{data_1, data_2, ...., data_n}

//Method 3
Array_name := [...]data_type{data_1, data_2, ...., data_n}

//Method 4
array_name := make([]data_type, length array)
```

Kemudian untuk fungsi *built-in* yang dapat digunakan untuk mengolah *array*:

Fungsi	Penjelasan
len(array)	Mendapatkan panjang sebuah <i>array</i>
array[index]	Mendapatkan nilai berdasarkan posisi <i>index array</i>
array[index] = value	Merubah nilai pada posisi <i>index array</i>
array	Mendapatkan semua nilai pada <i>array</i>

Selanjutnya untuk melakukan proses iterasi dapat dilakukan dengan dua metode berikut:

Input	Output
<pre>package main  import "fmt"  func main() {     teachAsstCode := [...]string{"MD", "RR", "MYM"}      //method 1     for i := 0; i &lt; len(teachAsstCode); i++ {         fmt.Printf("index: %d, code: %s\n", i,</pre>	<pre>index: 0, code: MD index: 1, code: RR index: 2, code: MYM  index: 0, code: MD index: 1, code: RR index: 2, code: MYM</pre>

```

    teachAsstCode[i])
}
fmt.Println()

//method 2
for i, code := range teachAsstCode {
    fmt.Printf("index: %d, code: %s\n", i, code)
}
}

```

## Slice

*Slice* merupakan sebuah *subarray* yang mana ukuran yang telah dideklarasikan bisa berubah. Perlu diperhatikan pada saat menggunakan *slice*:

1. *Pointer index*.
2. *Length* sebagai banyaknya nilai yang akan disimpan pada *slice*.
3. *Capacity* sebagai akomodasi banyaknya nilai yang dapat disimpan pada *slice* yang mana ukuran *length* harus lebih kecil dari *capacity*.

Untuk mendeklarasikan sebuah *slice* secara berdiri sendiri tanpa mengambil nilai dari *array* yang sudah dideklarasikan sebelumnya:

```

//method 1
var slice_name = []data_type{data_1, data_2, ..., data_n}

//method 2
var slice_name = make([]data_type, length, capacity)

```

Kemudian untuk fungsi *built-in* dan iterasi pada *slice* dapat menggunakan yang ada pada *array* dan juga ada penambahan fungsi baru yang dapat digunakan:

Fungsi	Penjelasan
append(slice, data)	Membuat <i>slice</i> baru dengan menambahkan nilai di posisi <i>index</i> terakhir.
Copy(dest_slice, src_slice)	Menyalin nilai dari <i>slice</i> lama ke <i>slice</i> baru.
array[srt_idx:end_idx]	Melakukan <i>slicing array</i> menjadi <i>slice</i> dimulai <i>index</i> awal ( <i>inclusive</i> ) dan <i>index</i> akhir ( <i>exclusive</i> ).

Perlu diperhatikan saat menjalankan *syntax* *append* jika ukuran *length* kurang dari *capacity* maka akan mengakibatkan *slice* tersebut akan mereferensi ke sebuah *array* sehingga nilai dari *index* terakhir pada *slice* akan berpengaruh terhadap nilai dari *index array*. Jika tidak, maka akan dibuatkan *slice* baru tanpa mereferensi *array* tersebut dan kapasitasnya akan menjadi dua kali lipat. Berikut contoh program yang menjelaskan penjelasan sebelumnya:

Input	Output
package main	-----
import "fmt"	[MD RR MYM]
func main() {	len: 3, cap: 3
teachAsstCode := [...]string{"MD", "RR", "MYM"}	[MD RR]
sTeachAsstCode := teachAsstCode[0:2]	len: 2, cap: 3
fmt.Println("-----")	-----
fmt.Println(teachAsstCode)	[MD RR NR]
fmt.Printf("len: %d, cap: %d\n",	len: 3, cap: 3
len(teachAsstCode), cap(teachAsstCode))	[MD RR NR]
fmt.Println(sTeachAsstCode)	len: 3, cap: 3
fmt.Printf("len: %d, cap: %d\n",	len: 3, cap: 3
len(sTeachAsstCode), cap(sTeachAsstCode))	-----
fmt.Println("-----")	[MD RR NR]
sTeachAsstCode = append(sTeachAsstCode, "NR")	len: 3, cap: 3
fmt.Println(teachAsstCode)	[MD RR NR CRS]
fmt.Printf("len: %d, cap: %d\n",	len: 4, cap: 6
len(teachAsstCode), cap(teachAsstCode))	-----
fmt.Println(sTeachAsstCode)	[MD RR NR]
fmt.Printf("len: %d, cap: %d\n",	len: 3, cap: 3
len(sTeachAsstCode), cap(sTeachAsstCode))	[MD RR NR CRS]
fmt.Println("-----")	len: 4, cap: 6
sTeachAsstCode = append(sTeachAsstCode, "CRS")	-----
fmt.Println(teachAsstCode)	[MD RR NR CRS]
fmt.Printf("len: %d, cap: %d\n",	len: 5, cap: 6
len(teachAsstCode), cap(teachAsstCode))	-----
fmt.Println(sTeachAsstCode)	[MD RR NR CRS]
fmt.Printf("len: %d, cap: %d\n",	len: 5, cap: 6
len(sTeachAsstCode), cap(sTeachAsstCode))	-----
}	[MD RR NR CRS]

## Map

Terdapat beberapa metode untuk mendeklarasikan sebuah *map* pada pemrograman Golang:

```
//Method 1
var map_name [key_type_data]value_type_data
Map_name := {key_1:value_1, key_2:value2, ...., key_n:value_n}

//Method 2
var map_name = make([key_type_data]value_type_data)

//Method 3
var map_name = [key_type_data]value_type_data{key_1:value_1,
key_2:value2, ...., key_n:value_n}
```

Kemudian untuk fungsi *built-in* yang dapat digunakan untuk mengolah *map*:

Fungsi	Penjelasan
<code>len (map)</code>	Mendapatkan panjang <i>map</i> .
<code>map [key]</code>	Mendapatkan <i>value</i> berdasarkan <i>key</i> .
<code>map [key] = value</code>	Merubah <i>value</i> berdasarkan <i>key</i> atau menambah <i>value</i> jika <i>key</i> tersebut tidak ada pada <i>map</i> .
<code>map</code>	Mendapatkan semua data <i>key</i> dan <i>value</i> .
<code>delete (map, key)</code>	Menghapus <i>value</i> berdasarkan <i>map</i> .

Selanjutnya untuk proses iterasi pada *map* dapat melihat contoh dari program berikut:

Input	Output
<pre>package main  import "fmt"  func main() {     teachAsstCode := map[string]string{"Rafi": "MD", "Rara": "RR", "Yoga": "MYM"}     teachAsstCode["Naufal"] = "NR"      for key, value := range teachAsstCode {         fmt.Printf("Name: %s Code: %s\n", key, value)     }     delete(teachAsstCode, "Naufal")     fmt.Println(teachAsstCode) }</pre>	<pre>Name: Rafi Code: MD Name: Rara Code: RR Name: Yoga Code: MYM Name: Naufal Code: NR map[Rafi:MD Rara:RR Yoga:RR]</pre>

## Function

*Function* atau fungsi pada pemrograman Golang memiliki peran yang sama dengan yang ada pada pemrograman lainnya yaitu untuk mengelompokkan baris kode agar lebih sederhana dan tidak redundan. Fungsi pada pemrograman Golang tidak mendukung dengan penerapan konsep *overloading* namun tetap mendukung konsep *overriding*. Berikut metode-metode yang dapat digunakan untuk mendeklarasikan sebuah fungsi:

```
func function_name(param_name type_data) return_type_data {
    //execute code
}
```

Input	Output
<pre>package main  import "fmt"</pre>	<pre>Hello Yogs t1: -7 t2: 0 t3: 36</pre>

```
func hello() { //simple function
    fmt.Println("Hello Yogs")
}

func hello2() (string, string) {
//simple function with multiple return
    return "Hello", "Hello"
}

func subtraction(x int) int {
//simple function with single parameter
    return -5 - x
}

func division(a int, b int) int {
//function with multiple parameter
    return a / b
}

func multiplication(numbers ...int) result int {
//function with variadic parameter and named return
    result = 1
    for _, number := range numbers {
        result *= number
    }
    return
}

func addition(a int, numbers ...int) int {
//function with parameter and variadic parameter
    result := 0
    for _, number := range numbers {
        result += number
    }
    return result + a
}

func main() {
    x := 2
    y := 3
    hello()
    a,b := hello2()
    fmt.Println("t1:", subtraction(x))
    fmt.Println("t2:", division(x, y))
    fmt.Println("t3:", multiplication(x, y, 2, 3))
    fmt.Println("t4:", addition(x, y, 3, 5))
    fmt.Println(a + " " + b)
}
```

```
t4: 13
Hello Hello
```

Pada pemrograman Golang juga terdapat *syntax defer* yang akan selalu dieksekusi dari sebuah fungsi meskipun fungsi tersebut menagalami eror.

Input	Output
<pre>package main  import (     "fmt" )  func main() {     fullname := "Yogs Mahendse"      hello := func(fullname string) string {         defer fmt.Println("Thank you")         fmt.Println("Hello,", fullname)         return "-----"     }(fullname)      fmt.Println(hello) }</pre>	<pre>Hello, Yogs Mahendse Yo Thank you -----</pre>

## Struct

*Struct* merupakan bagian dari data yang memiliki variabel dengan beragam tipe data yang mana konsepnya sedikit mirip dengan *class* pada pemrograman Java. *Struct* didefinisikan sebagai *type*. Berikut metode-metode untuk mendeklarasikan variabel dengan tipe data *struct*:

```
type name_struct struct {
    field_1 type_data_1
    field_2 type_data_2
    .....
    field_n type_data_n
}

//Method 1
var var_name_struct name_struct

//Method 2
var var_name_struct = name_struct{field_1:data_1, field_2:data_2,
....., field_n:data_n}

//Method 3
var var_name_struct = name_struct{data_1, data_2, ....., data_n}
```

Berikut fungsi *built-in* yang dapat digunakan untuk mengolah *struct*:

Fungsi	Penjelasan
<code>var_struct.field</code>	Mendapatkan nilai berdasarkan <i>field</i> .

<code>var _struct.field value</code>	= Mengubah nilai berdasarkan <i>field</i> .
--	---

Struct juga mendukung konsep *embedded* yang mengizinkan sebuah *struct* menjadi *field* untuk *struct* lain dan juga dapat dijadikan sebagai *array of structs*. Berikut contoh program sederhana penerapan *struct*:

Input	Output
<pre>package main  import "fmt"  type Person struct {     name    string     age     int     dateOfBirth Date }  type Date struct {     day    int     month int     year  int }  func main() {     teachAssts := []Person{         {"Rafi", 23, Date{1, 1, 2000}},         {"Rara", 23, Date{2, 2, 2000}},         {"Yogs", 23, Date{3, 3, 2000}},     }      for i, teachAsst := range teachAssts {         fmt.Println("Person", i+1)         fmt.Printf("name: %s, age: %d, date of birth: %d-%d-%d\n",             teachAsst.name, teachAsst.age,             teachAsst.dateOfBirth.day,             teachAsst.dateOfBirth.month,             teachAsst.dateOfBirth.year)     } }</pre>	<pre>Person 1 name: Rafi, age: 23, date of birth: 1-1-2000  Person 2 name: Rara, age: 23, date of birth: 2-2-2000  Person 3 name: Yogs, age: 23, date of birth: 3-3-2000</pre>

## Method

*Method* merupakan sebuah fungsi yang terikat dengan sebuah tipe data (*struct*, *interface*, dan sebagainya). Terdapat dua tipe untuk mendefinisikan sebuah *method*:

Tipe	Penjelasan
Basic Method	Tipe ini memiliki kemiripan dengan konsep <i>pass by value</i> yang mana saat adanya perubahan nilai pada <i>field</i> tidak akan berpengaruh terhadap nilai <i>field</i> aslinya.
Pointer Method	Tipe ini memiliki kemiripan dengan konsep <i>pass by reference</i> yang mana saat adanya perubahan nilai pada <i>field</i> akan berpengaruh terhadap nilai <i>field</i> aslinya.

Berikut contoh program yang menerapkan kedua tipe *method* tersebut:

Input	Output
<pre>package main  import "fmt"  type Person struct {     name    string     age     int     dateOfBirth Date }  type Date struct {     day    int     month int     year  int }  func (p Person) ChangeAge1(newAge int) {     p.age = newAge }  func (p *Person) ChangeAge2(newAge int) {     p.age = newAge }  func main() {     teachAssts := []Person{         {"Rafi", 23, Date{1, 1, 2000}},     }     teachAssts[0].ChangeAge1(24)     fmt.Println("change rafi age to:",     teachAssts[0].age)     teachAssts[0].ChangeAge2(24)     fmt.Println("change rafi age to:",     teachAssts[0].age) }</pre>	<pre>change rafi age to: 23 change rafi age to: 24</pre>

Pada versi 1.18 keatas, Golang menyediakan metode pendefinisian suatu *function*, *method*, dan *struct* secara *generic* yang di mana *generic* ini memungkinkan jenis tipe data pada dapat didefinisikan secara pasti saat pemanggilan *function*, *method* dan *field* pada *struct* sehingga ketiga hal tersebut dapat Anda *reuse* dengan berbagai tipe data dengan syarat tipe data tersebut kompatibel dengan *syntax* yang telah Anda buat. Untuk memperdalam mengenai penerapan *generic* ini dapat Anda pelajari secara mandiri pada tautan [berikut](#).

## Interface

### Basic Interface

Interface merupakan sekumpulan *method signature* yang konsepnya cukup mirip dengan *interface* pada pemrograman Java. Umumnya, *interface* dideklarasikan untuk menampung *method* yang nantinya akan digunakan oleh berbagai *struct* yang mana semua *method* harus diimplementasikan sehingga jika terdapat satu *method* yang tidak diimplementasikan akan menyebabkan eror. Berikut cara mendeklarasikan sebuah *interface* dan penerapannya pada contoh program dibawah.

```
type name_interface struct {
    method_interface_1
    method_interface_2
    .....
    method_interface_n
}

type name_struct struct {
    field_1 type_data_1
    field_2 type_data_2
    .....
    field_n type_data_n

    method_interface_1
    method_interface_2
    .....
    method_interface_n
}
```

Input	Output
<pre>package main  import (     "fmt"     "math" )  type Geometry interface {     Area() float64 }</pre>	<pre>Luas s1 : 600 Keliling s1 : 120 volume s2 ke-1 : 1728 volume s2 ke-2 : 2197</pre>

```

        Circumference() float64
    }

type ThreeDimension interface {
    Geometry
    Volume() float64
}

type Cube struct {
    Side float64
}

func (c Cube) Volume() float64 {
    return math.Pow(c.Side, 3)
}

func (c Cube) Area() float64 {
    return math.Pow(c.Side, 2) * 6
}

func (c Cube) Circumference() float64 {
    return c.Side * 12
}

func (c *Cube) ChangeSide(newSide float64) {
    c.Side = newSide
}

func main() {
    //metode 1 hanya bisa menggunakan method yang
    ada di interface ThreeDimension
    var shape1 ThreeDimension = Cube{Side: 10}

    //metode 2 bisa menggunakan semua method pada
    struct Cube
    var shape2 Cube = Cube{Side: 12}
    fmt.Println("Luas s1 :", shape1.Area())
    fmt.Println("Keliling s1 :", shape1.Circumference())
    fmt.Println("volume s2 ke-1 : ",
    shape2.Volume())
    shape2.ChangeSide(13)
    fmt.Println("volume s2 ke-2 : ",
    shape2.Volume())
}

```

## Empty Interface

Selain sebagai *method signature*, *interface* juga merupakan sebuah tipe data yang dapat menampung berbagai nilai tipe data apapun yang dideklarasikan dengan *syntax interface{}*  atau *any*. Meskipun *interface* dapat menampung berbagai nilai namun tidak dapat mengubah sifat atau fungsi bawaan dari tipe data yang diisi sehingga perlu dilakukan *assertion* dengan *syntax*

variable. (*type*). Nilai *default* yang disimpan pada *interface* adalah *nil*. Berikut contoh program dari penerapan *empty interface*.

Input	Output
<pre>package main  import (     "fmt" )  func main() {     var inf any = 10     fmt.Println(inf.(int) + 10) }</pre>	20

Kemudian anda juga dapat menggunakan *switch case* untuk melakukan pengecekan tipe data dari nilai yang disimpan pada *interface*.

Input	Output
<pre>package main  import "fmt"  func main() {     var inf any = 10      switch inf {     case inf.(int):         fmt.Printf("Nilai %d memiliki tipe data integer\n", inf.(int))     case inf.(string):         fmt.Printf("Nilai %s memiliki tipe data string\n", inf.(string))     } }</pre>	Nilai 10 memiliki tipe data integer

## Public dan Private

Untuk menegatur fungsi, method, *type*, dan variabel (dideklarasikan secara global) sebagai *public* atau *private* dapat dilihat dari penulisan huruf awal nama fungsi yang mana jika ditulis sebagai kapital maka ditentukan sebagai *public* dan sebaliknya sebagai *private*. Untuk fungsi, method, *type*, dan variabel (dideklarasikan secara global) yang diatur sebagai *public* maka dapat diakses oleh *package* lain dan jika sebagai *private* maka hanya dapat diakses satu *package* saja.

## Error Handling

Secara umum untuk mendefinisikan sebuah eror dapat menggunakan *interface error* yang memiliki *method Error()*. Namun untuk memudahkan pengembangan, anda dapat

mendeklarasikan *error* dengan menggunakan *method* yang ada pada *package errors* atau *fmt* yang lebih lengkapnya sebagai berikut:

```
Method 1: Use function New from package errors
var name_err error = errors.New("Error message")
```

```
Method 2: Use function Errorf from package fmt
var name_err error = fmt.Errorf("Error message")
```

Berikut beberapa metode *error handling* yang dapat digunakan pada pemrograman GoLang.

## Basic Error

Metode ini digunakan untuk me-*handle* eror dengan mengembalikan nilai eror pada suatu fungsi.

Input	Output
<pre>package main  import (     "fmt"     "strconv" )  func main() {     str := "string"     strToInt, err := strconv.Atoi(str)     if err != nil {         fmt.Println(err.Error())     } else {         fmt.Println(strToInt)     } }</pre>	strconv.Atoi: parsing "string": invalid syntax

## Panic

Metode ini digunakan jika sekiranya eror yang dikembalikan tidak bisa di-*handle* oleh program sehingga *program* akan dihentikan secara paksa.

Input	Output
<pre>package main  import (     "fmt"     "strconv" )  func main() {     str := "string" }</pre>	panic: strconv.Atoi: parsing "string": invalid syntax  goroutine 1 [running]: main.main() C:/xxxx/main.go:12 +0x85

```

strToInt, err := strconv.Atoi(str)
if err != nil {
    panic(err)
} else {
    fmt.Println(strToInt)
}
}

```

## Recover

Metode ini akan digunakan ketika ingin me-*handle panic error* dengan menggunakan *syntax defer* sehingga program yang seharusnya berhenti akan tetap berjalan.

Input	Output
<pre> package main  import "fmt"  func DivisionByZero() {     defer func() {         if err := recover(); err != nil {             fmt.Println("Error:", err)         }     }()     a := 2     b := 0     fmt.Println(a / b)     fmt.Println("It is Printed?") }  func main() {     fmt.Println("Start Programs")     DivisionByZero()     fmt.Println("End Programs") } </pre>	Start Programs Error: runtime error: integer divide by zero End Programs

## Basic I/O

### Basic Input

Untuk menerima sebuah teks input pengguna dari terminal, anda dapat memanfaatkan *method* dari dua *package* yaitu `fmt` untuk teks input yang menerima hanya satu kata saja dan `bufio` untuk teks input yang lebih dari satu kata. Berikut contoh program sederhana yang menggunakan kedua *package* tersebut.

Input	Output
<pre> package main  import (     "fmt"     "bufio"     "os" ) </pre>	Hello, what is your firstname: Imai

```

    "bufio"
    "fmt"
    "os"
)

func main() {
    var fname string
    var lname string
    fmt.Println("Hello, what is your firstname: ")
    fmt.Scanln(&fname)
    fmt.Println("Then, what is your lastname: ")
    fmt.Scanln(&lname)
    fmt.Println("Last, where do you come from: ")
    reader := bufio.NewReader(os.Stdin)
    text, err := reader.ReadString('\n')
    if err != nil {
        panic(err)
    }
    fmt.Println("Welcome " + fname + " " + lname +
" from " + text)
}

```

Then, what is your  
lastname: Lisa  
Last, where do you  
come from: Depok,  
Indonesia  
Welcome Imai Lisa  
from Depok, Indonesia

Untuk contoh penggunaan method-method lain pada package bufio dapat anda lihat pada tautan [berikut](#).

## Basic Output

Untuk mencetak teks input anda dapat menggunakan method `print`, `println`, dan `printf` dari package `fmt`.

Input	Output
<pre> package main  import (     "bufio"     "fmt"     "os" )  func main() {     //method 1 use fmt.Print     fmt.Println("Hello, ", "Imai ")     fmt.Println("Lisa\n")      //methode 2 use fmt.Println     fmt.Println("===== ")     fmt.Println("Hello, ", "Imai ", "Lisa")     fmt.Println("===== ")      //methode 3 use fmt.Printf     fmt.Printf("Hello, %s %s\n", "Imai", "Lisa") } </pre>	<pre> Hello, Imai Lisa ===== Hello, Imai Lisa ===== Hello, Imai Lisa </pre>

## Package Manager

Saat anda membuat program yang membutuhkan *method* dari *package* lain, anda dapat menambahkan *syntax* “*import*” dibawah “*package*” pada kode anda. Setiap anda me-*import* suatu *package* maka anda diharuskan untuk menggunakan satu *method* yang ada pada *package* tersebut karena jika tidak anda gunakan maka program anda akan mengalami eror. Terdapat dua metode *import* yang dapat anda gunakan:

```
//method 1, import one package
import "[package_name]"

//method 2, import more packages
import (
    "package_name_1"
    .....
    "package_name_n"
)
```

Secara umum, terdapat tiga jenis *package* yang dapat digunakan:

No.	Package	Penjelasan
1.	<i>Standard Library</i>	<i>Package</i> bawaan yang sudah tersedia oleh Golang. Untuk <i>package-package</i> yang termasuk kategori ini dapat anda lihat pada tautan <a href="#">berikut</a> .
2.	<i>Customization Packages</i>	<i>Package</i> yang dibuat pada proyek sendiri.
3.	<i>Third-Party Packages</i>	<i>Package</i> yang telah diunduh setelah menjalankan perintah <code>go get</code> yang akan disimpan pada berkas <code>go.mod</code> .

Berikut contoh program sederhana yang menerapkan ketiga jenis *package*:

```
// project structure
- gin
  - calculator
    - calculator.go
  - go.mod
  - main.go
```

Input (calculator.go)	Output
package calculate	Result of convert 100 USD to Rupiah: 1500000 IDR
func Multiplication(a int, b int) int { return a * b }	Result of multiplication amount in Rupiah with 2: 300000000
Input (main.go)	
package main	
import (	

```

"fmt"                                //standard library
"gin/calculate"                      //customization package
cr "github.com/bojanz/currency"      //third-party package
)

func main() {
    dollar, err := cr.NewAmount("100", "USD")
    if err != nil {
        panic(err)
    }

    usdToIdr, err := dollar.Convert("IDR", "15000")
    if err != nil {
        panic(err)
    }

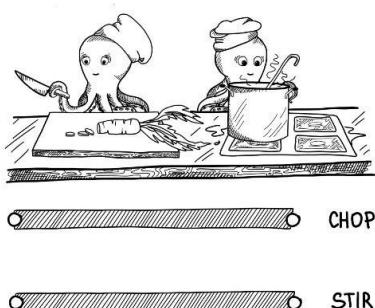
    idrInt, err := usdToIdr.Int64()
    if err != nil {
        panic(err)
    }

    fmt.Println("Result of convert 100 USD to Rupiah:",
    usdToIdr)
    fmt.Println("Result of multiplication amount in Rupiah
with 2:", calculate.Multiplication(int(idrInt), 2))
}

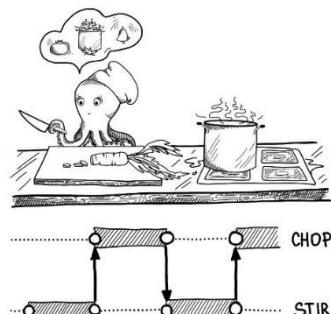
```

## Goroutine

Sebelum membahas *goroutine* anda perlu memahami bagaimana cara kerja dari sebuah *concurrency*. *Concurrency* merupakan Teknik yang digunakan untuk menyelesaikan *task-task* dengan rentang waktu yang sama di mana setiap *task* dikerjakan secara bergantian. Berbeda dengan konsep paralel di mana *task-task* akan dikerjakan secara bersamaan.



Cara kerja paralel (source:  
[freecontent.manning.com](http://freecontent.manning.com))



Cara kerja *concurrency* (source:  
[freecontent.manning.com](http://freecontent.manning.com))

*Goroutine* merupakan sebuah *thread* yang bersifat “*lightweight*” yang dikelola oleh *Go Runtime*. Untuk mendeklarasikan fungsi sebagai *goroutine* dapat menambahkan *syntax go* di awal fungsi. Secara umum, fungsi `main()` merupakan *goroutine* utama yang mana saat mendeklarasikan

*goroutine* di fungsi lain dan *goroutine* utama selesai dieksekusi akan menyebabkan *goroutine* dari fungsi lain juga akan ikut berhenti. Berikut contoh program penerapan *goroutine* sederhana.

Input	Output
<pre>package main  import (     "fmt"     "time" )  func main() {     fmt.Println("Welcome to main goroutine")      go func() { // execute go routine         fmt.Println("Welcome to new goroutine")     }() }  fmt.Println("Prepare sleep") time.Sleep(10 * time.Millisecond) // wait 10 ms to execute new goroutine fmt.Println("Main go routine stopped")</pre>	<pre>Welcome to main goroutine Prepare sleep Welcome to new goroutine Main goroutine stopped</pre>

Untuk melihat perbedaan *response time* antara program yang dijalankan secara sekuensial dan konkuren dapat dilihat pada program berikut.

Sequential Program	
Input	Output
<pre>package main  import (     "fmt"     "time" )  var start time.Time  func init() {     start = time.Now() }  func main() {     func() {         time.Sleep(100 * time.Millisecond)         fmt.Println("time to execute new goroutine:",         time.Since(start))     }() }  time.Sleep(200 * time.Millisecond)</pre>	<pre>time to execute new goroutine: 112.0524ms time to execute main goroutine: 326.428ms</pre>

```

    fmt.Println("time to execute main goroutine:",
    time.Since(start))
}

```

### Concurrency Program

Input	Output
<pre> package main  import (     "fmt"     "time" )  var start time.Time  func init() {     start = time.Now() }  func main() {     go func() {         time.Sleep(100 * time.Millisecond)         fmt.Println("time to execute new goroutine:",         time.Since(start))     }() }  time.Sleep(200 * time.Millisecond) fmt.Println("time to execute main g </pre>	<pre> time to execute new goroutine: 106.0294ms time to execute main goroutine: 213.6495ms </pre>

Untuk lebih mendalami mengenai *goroutine*. Anda bisa melakukan pembelajaran mandiri terkait penggunaan *channel* dan *buffered channel* pada tautan [berikut](#) serta penggunaan *package sync* pada tautan [berikut](#).

## Spesifikasi

**Catatan:** Pada bagian ini, Anda akan mendemonstrasikan kemampuan membuat program sederhana dengan menggunakan bahasa pemrograman Go. Tugas ini akan dinilai dan **Anda diminta untuk melaporkan hasil penggerjaan tugas Anda sesuai dengan instruksi yang diberikan. Harap luangkan waktu untuk membaca spesifikasi terlebih dahulu dan pastikan bahwa Anda telah memiliki salinan lembar jawaban sendiri.**

## Studi Kasus

Sigmart POS merupakan sistem *Point of Sale* yang digunakan oleh minimarket Sigmart yang berbasis *command line* di mana pengguna dapat melakukan penambahan item, member, transaksi, dan sebagainya. Adapun struktur **berkas template** yang dapat [diunduh](#) pada program Sigmart POS ini tertera sebagai berikut:

- command.go
- main.go
- model.go

- a. Berkas command.go digunakan untuk memproses perintah masukan yang diterima oleh pengguna. Berkas ini berisi:
- Variabel Items digunakan untuk menyimpan data-data item.
  - Variabel Members digunakan untuk menyimpan data-data member.
  - Function AddItem(SKU string, itemName string, price int32, stockQty int32) digunakan untuk memproses penambahan data item ke Items. Terdapat skenario yang dilakukan pada function ini:
    1. Jika data item yang akan ditambahkan sudah ada di Items (berdasarkan SKU) maka kembalikan **string kosong** dan pesan error yang berisi "**item [SKU] is already in list of items**".
    2. Tambahkan data item ke Items dan kembalikan **string "successfully added item [SKU] to list of items"** dan pesan error bernilai nil.  - Function DeleteItem(SKU string) digunakan untuk memproses penghapusan data item dari Items. Terdapat skenario yang dilakukan pada function ini:
    1. Jika data item yang akan dihapus tidak ada di Items (berdasarkan SKU) maka kembalikan **string kosong** dan pesan error yang berisi "**item [SKU] is not in list of items**".
    2. Jika data yang akan dihapus memiliki data transaksi maka kembalikan **string kosong** dan pesan error yang berisi "**there is at least one transaction taking item [SKU]**".
    3. Hapus data item dari Items dan kembalikan **string "successfully deleted item [SKU] from list of items"** dan pesan error bernilai nil.  - Function AddMember(idMember string, memberName string) digunakan untuk memproses penambahan data member ke Members. Terdapat skenario yang dilakukan pada function ini:
    1. Jika data member yang akan ditambahkan sudah ada di Members (berdasarkan ID Member) maka kembalikan **string kosong** dan pesan error yang berisi "**member [ID\_MEMBER] is already in list of members**".
    2. Tambahkan data member ke Members dan kembalikan **string "successfully added member [ID\_MEMBER] to list of members"** dan pesan error bernilai nil.  - Function DeleteMember(idMember string) digunakan untuk memproses penghapusan data member dari Members. Terdapat skenario yang dilakukan pada function ini:
    1. Jika data member yang akan dihapus tidak ada di Members (berdasarkan ID Member) maka kembalikan **string kosong** dan pesan error yang berisi "**member [ID\_MEMBER] is not in list of members**".
    2. Jika data yang akan dihapus memiliki data transaksi maka kembalikan **string kosong** dan pesan error yang berisi "**there is at least one transaction taking member [ID\_MEMBER]**".
    3. Hapus data member dari Members dan kembalikan **string "successfully deleted member [ID\_MEMBER] from list of members"** dan pesan error bernilai nil.  - Function AddTransaction(qty int32, data string...) digunakan untuk memproses penambahan transaksi. Terdapat skenario yang dilakukan pada function ini:

1. Variabel data memiliki tipe data *array of string* yang bisa berisi SKU item dan ID member atau SKU item saja.
  2. Jika data item tidak ada di `Items` (berdasarkan SKU) maka kembalikan **string kosong** dan pesan error yang berisi "**item [SKU] is not in list of items**".
  3. Jika pada variabel data terdapat ID Member maka lakukan pengecekan jika data member tidak ada di `Members` (berdasarkan ID Member) maka kembalikan **string kosong** dan pesan error yang berisi "**member [ID Member] is not in list of members**".
  4. Cek jumlah stok item (berdasarkan SKU) yang tersedia, jika stok tidak mencukupi maka kembalikan **string kosong** dan pesan error yang berisi "**stock qty for item [SKU] is not sufficient**".
  5. Tambahkan data transaksi tersebut ke data item (berdasarkan SKU) dan data member (berdasarkan ID Member jika ada) dan kurangi stock item dengan jumlah qty yang diinput oleh pengguna.
  6. Kembalikan **string "successfully added transaction item [SKU] for member [ID\_MEMBER]"** (jika ID Member) atau **"successfully added transaction item [SKU]"** (jika tidak diisi ID member) dan pesan error bernilai `nil`.
- *Function* `RestockItem(SKU string, qty int32)` digunakan untuk memproses penambahan stock qty pada item. Terdapat skenario yang dilakukan pada *function* ini:
1. Jika data item tidak ada di `Items` (berdasarkan SKU) maka kembalikan **string kosong** dan pesan error yang berisi "**item [SKU] is not in list of items**".
  2. Tambahkan qty baru ke data item dan kembalikan **string "successfully restock qty for item [SKU]"** dan pesan error bernilai `nil`.
- *Function* `GetTransactionItem(SKU string)` digunakan untuk mendapatkan data-data transaksi dari item. Terdapat skenario yang dilakukan pada *function* ini:
1. Jika data item tidak ada di `Items` (berdasarkan SKU) maka kembalikan **nilai nil** dan pesan error yang berisi "**item [SKU] is not in list of items**".
  2. Kembalikan data transaksi dan pesan error bernilai `nil`.
- *Function* `GetTransactionMember(idMember string)` digunakan untuk mendapatkan data-data transaksi dari member. Terdapat skenario yang dilakukan pada *function* ini:
1. Jika data item tidak ada di `Members` (berdasarkan Member) maka kembalikan **nilai nil** dan pesan error yang berisi "**member [ID\_MEMBER] is not in list of members**".
  2. Kembalikan data transaksi dan pesan error bernilai `nil`.
- b. Berkas `main.go` digunakan sebagai berkas yang akan dieksekusi pertama kali. Terdapat dua *function* yang digunakan pada berkas ini:
- *Function* `main()` digunakan untuk mengeksekusi program dan menerima masukan dari pengguna.
  - *Function* `executeCommand(command string, spl[]string)` digunakan untuk mengeksekusi perintah yang telah dimasukan oleh pengguna. Terdapat sembilan perintah yang digunakan pada program ini:
    - ADD\_ITEM [SKU] [ITEM\_NAME] [PRICE] [STOCK\_QTY]  
Digunakan untuk memproses penambahan data item ke `Items`. Terdapat skenario untuk memproses perintah ini:

1. Jika format masukan tidak sesuai, maka akan mencetak pesan “[FAILED] your input command is incorrect”.
  2. Memproses penambahan data item ke Items .
  3. Mencetak hasil proses yang telah dilakukan.
- **DELETE\_ITEM [SKU]**  
Digunakan untuk memproses penghapusan data item dari Items . Terdapat skenario untuk memproses perintah ini:
    1. Jika format masukan tidak sesuai, maka akan mencetak pesan “[FAILED] your input command is incorrect”.
    2. Memproses penghapusan data item dari Items .
    3. Mencetak hasil proses yang telah dilakukan.
  - **ADD\_MEMBER [ID\_MEMBER] [MEMBER\_NAME]**  
Digunakan untuk memproses penambahan data member dari Members . Terdapat skenario untuk memproses perintah ini:
    1. Jika format masukan tidak sesuai, maka akan mencetak pesan “[FAILED] your input command is incorrect”.
    2. Memproses penambahan data member ke Members .
    3. Mencetak hasil proses yang telah dilakukan.
  - **DELETE\_MEMBER [ID\_MEMBER]**  
Digunakan untuk memproses penghapusan data member dari Members . Terdapat skenario untuk memproses perintah ini:
    1. Jika format masukan tidak sesuai, maka akan mencetak pesan “[FAILED] your input command is incorrect”.
    2. Memproses penghapusan data member dari Members .
    3. Mencetak hasil proses yang telah dilakukan.
  - **ADD\_TRANSACTION [QTY] [SKU] [ID\_MEMBER] / ADD\_TRANSACTION [QTY] [SKU]**  
Digunakan untuk memproses penambahan data transaksi ke item dan member (jika ada). Terdapat skenario untuk memproses perintah ini:
    1. Jika format masukan tidak sesuai, maka akan mencetak pesan “[FAILED] your input command is incorrect”.
    2. Memproses penambahan data transaksi ke item dan member (jika ada) .
    3. Mencetak hasil proses yang telah dilakukan.
  - **RESTOCK\_ITEM [SKU] [QTY]**  
Digunakan untuk memproses penambahan data stock qty ke item. Terdapat skenario untuk memproses perintah ini:
    1. Jika format masukan tidak sesuai, maka akan mencetak pesan “[FAILED] your input command is incorrect”.
    2. Memproses penambahan stock qty member ke item.
    3. Mencetak hasil proses yang telah dilakukan.

- TRANSACTION\_ITEM\_RECAP [SKU]

Digunakan untuk mencetak data-data transaksi dari item. Terdapat skenario untuk memproses perintah ini:

1. Jika format masukan tidak sesuai, maka akan mencetak pesan “[FAILED] your input command is incorrect”.
2. Memproses mendapatkan data transaksi dari item .
3. Mencetak hasil proses yang telah dilakukan.

- TRANSACTION\_MEMBER\_RECAP [ID\_MEMBER]

Digunakan untuk mencetak data-data transaksi dari item. Terdapat skenario untuk memproses perintah ini:

1. Jika format masukan tidak sesuai, maka akan mencetak pesan “[FAILED] your input command is incorrect”.
2. Memproses mendapatkan data transaksi dari member .
3. Mencetak hasil proses yang telah dilakukan.

- EXIT

Perintah ini digunakan untuk menghentikan program yang berjalan.

Note:

- Beberapa data yang masukannya perlu di *cast* menjadi integer, gunakan *method Atoi* dari *package strconv* untuk melakukan *cast* tersebut. Jika mengembalikan eror maka cetak “[FAILED] your input command is incorrect”.
  - Untuk jenis perintah selain yang disebutkan diatas maka program akan berhenti.
- *Function PrintMessage(successMsg string, errMsg error)* digunakan untuk mencetak pesan yang akan ditampilkan. Terdapat skenario yang dilakukan pada fungsi ini:
1. Jika menerima eror maka akan mencetak pesan dengan format "[FAILED] [ERR\_MSG]", jika tidak maka akan mencetak pesan dengan format "[SUCCESS] [SUCCESS\_MSG]".
- *Function PrintTransactionRecap(transactions []Transaction, errMsg err)* digunakan untuk mencetak rekapan transaksi yang akan ditampilkan. Terdapat skenarion yang dilakukan pada fungsi ini:
1. Jika menerima eror maka akan mencetak pesan dengan format "[FAILED] [ERR\_MSG]".
  2. Jika tidak ada data transaksi maka akan mencetak pesan “[FAILED] does not have transaction data”
  3. Cetak data-data transaksi dengan format sebagai berikut:

-X-X-X-X-X-X-X-X-X-X-X-

SKU: [SKU-1], ID Member: [ID\_MEMBER-1], Qty: [QTY-1], Total Price: [QTY-1\*PRICE-1]

SKU: [SKU-2], ID Member: [ID\_MEMBER-2], Qty: [QTY-2] , Total Price: [QTY-2\*PRICE-2]

.....

SKU: [SKU-N], ID Member: [ID\_MEMBER-N], Qty: [QTY-N] , Total Price: [QTY-N\*PRICE-N]

-X-X-X-X-X-X-X-X-X-X-X-

Note: Jika pada transaksi tidak mencantumkan ID Member maka ID Member akan diisi “-“.

- c. Berkas `model.go` digunakan untuk memanipulasi data untuk `struct Item`, `Member`, dan `Transaction`. Berkas Ini berisi:
- *Interface Tool()* yang berisi kumpulan *method* yang akan digunakan pada `struct item` dan `member`.
  - *Struct Item* merupakan *object* untuk menyimpan informasi item dan memanipulasi informasi tersebut. *Struct* ini berisi:
    - *Field SKU* untuk menyimpan data SKU item.
    - *Field ItemName* untuk menyimpan data nama item.
    - *Field StockQty* untuk menyimpan data jumlah stock pada item.
    - *Field Price* untuk menyimpan harga satuan item.
    - *Field Transactions* untuk menyimpan data-data transaksi pada item.
    - *Method AddTransaction (Transaction)* untuk menambahkan data transaksi ke `Transactions`. Terdapat skenario dilakukan pada method ini:
      1. Jika data ID Member dan SKU dari data transaksi yang sudah ada ini sama dengan data transaksi baru maka cukup untuk menambahkan qty saja.
      2. Jika tidak sama maka tambahkan data transaksi tersebut.
    - *Method GetData ()* untuk mendapatkan data objek item.
  - *Struct Member* merupakan *object* untuk menyimpan informasi member dan memanipulasi informasi tersebut. *Struct* ini berisi:
    - *Field IdMember* untuk menyimpan data id member.
    - *Field MemberName* untuk menyimpan data nama member.
    - *Field Transactions* untuk menyimpan data-data transaksi pada member.
    - *Method AddTransaction (Transaction)* untuk menambahkan data transaksi ke `Transactions`. Terdapat skenario dilakukan pada method ini:
      1. Jika data ID Member dan SKU dari data transaksi yang sudah ada ini sama dengan data transaksi baru maka cukup untuk menambahkan qty saja.
      2. Jika tidak sama maka tambahkan data transaksi tersebut.
    - *Method GetData ()* untuk mendapatkan data objek member.
  - *Struct Transaction* merupakan struktur untuk menyimpan informasi transaksi. *Struct* ini berisi:
    - *Field SKU* untuk menyimpan data SKU item.
    - *Field IdMember* untuk menyimpan data id member.
    - *Field Qty* untuk menyimpan data jumlah stok item yang sudah dibeli.
    - *Field Price* untuk menyimpan harga satuan item.

Contoh masukan dan luaran yang diimplementasikan pada program ini:

```
Name: Bintang Skibidi, ID Student: 2206123456
=====
Welcome to Sigmart Point of Sales
Please input your command below
=====
ADD_MEMBER M001 Jason
[SUCCESS] successfully added member M001 to list of members
ADD_ITEM I001 SIROUP 20000 2
[SUCCESS] successfully added item I001 to list of items
ADD_ITEM
[FAILED] your input command is incorrect
ADD_MEMBER M002 James
[SUCCESS] successfully added member M002 to list of members
ADD_TRANSACTION 1 I001 M001
[SUCCESS] successfully added transaction item I001 for member M001
DELETE_MEMBER M002
[SUCCESS] successfully deleted member M002 from list of members
TRANSACTION_ITEM_RECAP I001
-X-X-X-X-X-X-X-X-X-X-X-
SKU: I001, ID Member: M001, Qty: 1, Total Price: 20000
-X-X-X-X-X-X-X-X-X-X-X-
EXIT
exit status 1
```

Berikut tugas yang akan Anda kerjakan:

1. Lengkapi potongan-potongan kode yang ada pada berkas command.go, model.go dan main.go agar program dapat berjalan dengan semestinya.
2. Mencoba testcase masukan perintah yang diberikan dan hasil luaran yang didapatkan seperti contoh yang telah diberikan. Berikut testcase yang akan anda coba:

- *Testcase 1*

```
ADD_ITEM MK201 MASKER 12500 20
ADD_MEMBER MM001 JOJO
ADD_TRANSACTION 2 MK201
ADD_TRANSACTION 15 MK201 MM001
DELETE_ITEM MK201
EXIT
```

- *Testcase 2*

```
ADD_MEMBER MM010 JAKA
ADD_ITEM XI021 XIAOMAY_14 14500
ADD_MEMBER MM010 JOKO
ADD_TRANSACTION 2 XI021 MM010
ADD_ITEM SS910 SUMSANG_GALAXY_S24 9500000 20
ADD_TRANSACTION 1 SS910
TRANSACTION_ITEM_RECAP SS910
DELETE_MEMBER MM010
EXIT
```

- *Testcase 3*

```
ADD_MEMBER MM921 JONO
ADD_ITEM MR134 MIRONAGA_BALITA 21500 10
ADD_ITEM BB732 BOBOLAC_BATITA 34300 5
ADD_TRANSACTION 15 MR134 MM921
ADD_MEMBER MM934 JONI
ADD_TRANSACTION 5 BB732 MM934
ADD_TRANSACTION 2 MR134
RESTOCK_ITEM BB732 2
ADD_TRANSACTION 1 BB732
DELETE_MEMBER MR134
TRANSACTION_MEMBER_RECAP MM934
TRANSACTION_ITEM_RECAP BB732
EXIT
```

Untuk menjalankan program masukan perintah di terminal anda: go run .

## Asumsi dan Kesepakatan

- Anda **tidak diperkenankan** untuk menambahkan *method* atau *function* baru pada program.
- Anda **tidak diperkenankan** menggunakan *third-party packages*.
- Masukan yang diterima dari pengguna bersifat **case-sensitive**.
- Untuk pemisah antar data yang dimasukan menggunakan spasi.
- Gunakan *template* yang dapat anda unduh [di sini](#) karena akan berpengaruh pada penilaian kebenaran kode oleh grader.

## Komponen Penilaian

- [42] Kebenaran kode (pengecekan kode menggunakan *grader* dan manual).
- [25] *Self-Testing* dari ketiga *testcase* dan *screenshot* hasil luaran ke lembar jawaban.
- [33] *Testing* dari beberapa *testcase* menggunakan *grader*.



## Informasi Pengumpulan Berkas

Lembar jawaban yang harus dikumpulkan untuk tugas kali ini harus disatukan dengan berkas lainnya (A01a, A01b, dan A01c). Perlu diperhatikan bahwa A01 ini menggunakan satu *template* lembar jawaban untuk semua bagian dan Anda harus menjawab semua bagian dalam satu dokumen tersebut. Untuk bagian **Introduction to Go Programming Language** ini, Anda perlu untuk mengumpulkan berkas berikut:

1. Berkas laporan (lembar jawaban) yang merupakan gabungan dengan tugas-tugas A01 lainnya. Laporan harus diekspor dalam format PDF.

**Format Penamaan:** A01\_[NPM].pdf

**Contoh Penamaan:** A01\_1906111111 pdf

2. Berkas command.go, model.go, dan main.go yang semua potongan kodennya sudah diisi.

**Format Penamaan:** A01\_[NPM]\_command.go, A01\_[NPM]\_model.go, A01\_[NPM]\_main.go

**Contoh Penamaan:** A01\_1906111111\_command.go, A01\_1906111111\_model.go,

A01\_1906111111\_main.go

## Peraturan

---

### Keterlambatan

Anda diharapkan dapat mengumpulkan hasil pekerjaan yang dilakukan sebelum batas waktu pengumpulan. Jika terdapat kondisi di mana Anda terpaksa terlambat mengumpulkan hasil pekerjaan, terdapat jangka waktu tambahan di mana Anda masih diperbolehkan mengumpulkan hasil pekerjaan dengan konsekuensi tertentu. Jika X adalah durasi setelah batas waktu pengumpulan yang ditetapkan sampai waktu Anda mengumpulkan hasil pekerjaan, Anda akan menerima penalti nilai pekerjaan sebagaimana diatur pada peraturan berikut ini:

- |   |  |
|---|--|
| • $X < 10$ menit                            | : Tidak ada penalti                            |
| • $10 \text{ menit} \leq X < 2 \text{ jam}$ | : 25% penalti                                  |
| • $2 \text{ jam} \leq X < 4 \text{ jam}$    | : 50% penalti                                  |
| • $4 \text{ jam} \leq X < 6 \text{ jam}$    | : 75% penalti                                  |
| • $X \geq 6 \text{ jam}$                    | : Cut-off (Pekerjaan anda tidak akan diterima) |

### Plagiarisme

Anda diperbolehkan berdiskusi tentang pekerjaan Anda dengan peserta kuliah lain atau pihak lainnya, namun Anda harus memastikan bahwa **semua pekerjaan yang dikumpulkan adalah murni hasil pekerjaan Anda sendiri**. Anda dilarang keras melakukan tindak plagiarisme atau kecurangan akademik lainnya. Menurut kamus daring Merriam-Webster, plagiarisme berarti:

- Mencuri dan mengklaim (ide atau kata orang lain) sebagai milik sendiri
- Menggunakan hasil (karya/pekerjaan orang lain) sebagai milik sendiri
- Melakukan pencurian literatur/sastra
- Merepresentasikan ulang sebuah ide/produk yang sudah ada sebagai sesuatu yang bersifat baru dan orisinal.

Tim pengajar memiliki hak untuk meminta klarifikasi terkait dugaan ketidakjujuran akademik, terutama plagiarisme, dan memberikan konsekuensi berupa **pengurangan nilai hasil pekerjaan atau pencabutan nilai (nilai diubah menjadi nol)** untuk hasil pekerjaan yang terkonfirmasi dikerjakan secara tidak jujur.