ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN

LẬP TRÌNH NÂNG CAO (CO2039)

ĐỀ TÀI:

TÌM HIỂU VỀ NGÔN NGỮ LẬP TRÌNH GOLANG

LỚP L05 – HK212

Giảng viên hướng dẫn: *Trương Tuấn Anh*

Sinh viên thực hiện: *Nguyễn Đức An*

MSSV: 2010102

TP. Hồ Chí Minh, tháng 5 năm 2022

MỤC LỤC

DANH MỤC HÌNH ẢNH

PHẦN MỞ ĐẦU

Golang là một ngôn ngữ lập trình ra đời tương đối muộn nhưng với tốc độ phát triển nhanh nên nhanh chóng trở thành một trong những ngôn ngữ lập trình được yêu thích nhất trong ngành công nghệ phần mềm ngày nay, đặc biệt là trong lĩnh vực phát triển backend cho website. Theo một cuộc khảo sát của stackoverflow, Golang chiếm vị trí thứ ba trong bảng xếp hạng các ngôn ngữ lập trình mà các nhà phát triển muốn nghiên cứu.

Vậy lý do gì khiến Golang lại được yêu thích như vậy ? Bởi vì, Golang được lên ý tưởng xây dựng và phát triển dựa trên nguyên tắc học hỏi, kế thừa những ưu điểm và khắc phục những khuyết điểm của các ngôn ngữ đi trước. Do đó, Golang có tốc độ xử lý hiệu quả như C / C++, xử lý các hoạt động song song như Java và có khả năng đọc code dễ dàng như Python, Perl và Erlang.

Trong bài tập lớn này, chúng ta sẽ cùng nhau đi tìm hiểu kỹ hơn về ngôn ngữ lập trình Go (Golang). Nội dung được trình bày trong báo cáo bao gồm 3 chương sau:

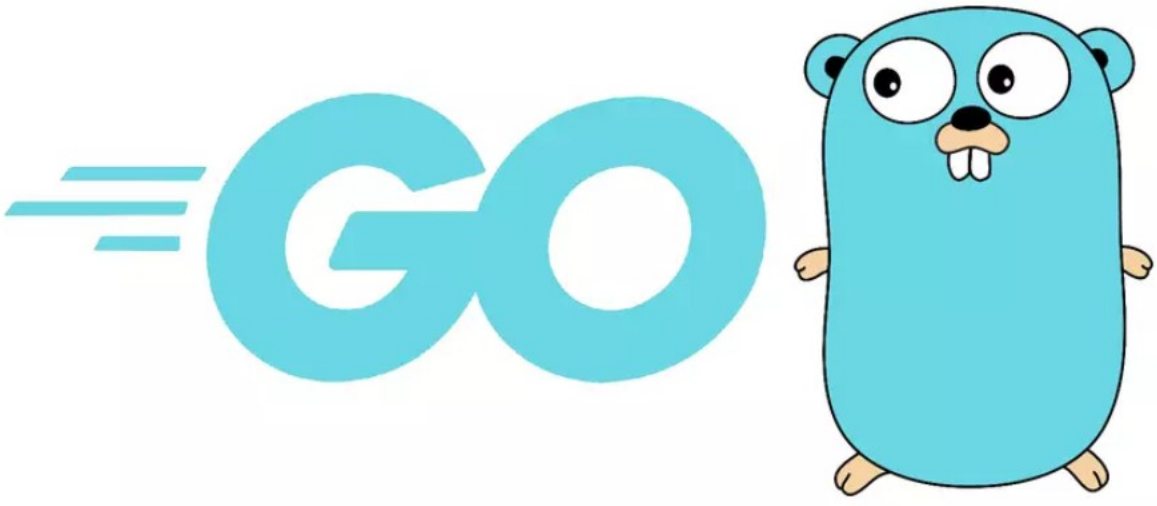
- **Chương 1:** Giới thiệu về ngôn ngữ lập trình Golang, chương này xoay quanh lịch sử hình thành và phát triển của Golang.
- **Chương 2:** Chương này bao gồm hướng dẫn cài đặt môi trường, nghiên cứu về cách sử dụng, lập trình với ngôn ngữ Golang, chương này bao gồm những vấn đề cơ bản như một số syntax phổ biến, kiểu dữ liệu, câu điều kiện, vòng lặp, đến các vấn đề nâng cao về câu trúc dữ liệu như Array, Slice, Map, Recursion,...
- **Chương 3:** Chương này đánh giá điểm mạnh, điểm yếu của ngôn ngữ lập trình Golang so với các ngôn ngữ khác và từ đó trả lời cho câu hỏi khi nào cần sử dụng Golang.

PHẦN NỘI DUNG

CHƯƠNG 1. GIỚI THIỆU VỀ NGÔN NGỮ LẬP TRÌNH GOLANG

1.1 Golang là gì ?

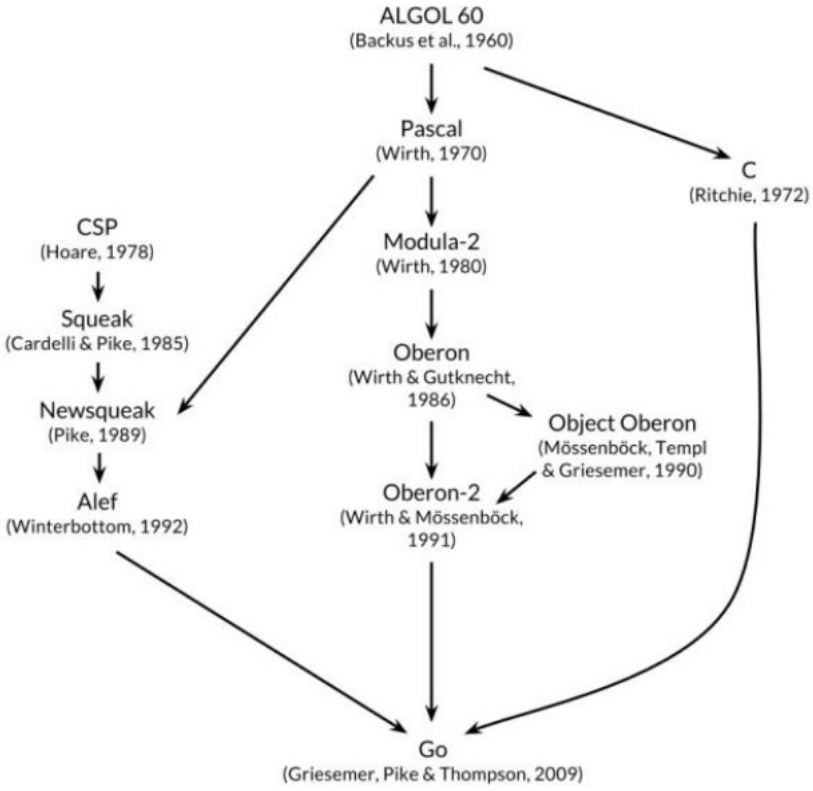
- Ngôn ngữ Go (Golang) ban đầu được **thiết kế và phát triển** bởi một nhóm kĩ sư Google bao gồm **Robert Griesemer**, **Ken Thompson** và **Rob Pike** vào năm **2007.**
- Go là một ngôn ngữ có kiểu dữ liệu tĩnh (**statically-typed language**), có cú pháp tương đối giống với C.
- Go được **chính thức ra mắt** vào tháng 11 năm **2009** và ban đầu được sử dụng trong một số hệ thống sản xuất của Google.
- Logo biểu tượng của ngôn ngữ lập trình Golang là một con **gopher** (chuột túi), đây là một loài vật dễ thương ai cũng yêu thích. Ý nghia của logo này là muốn thể hiện rằng Golang là một ngôn ngữ thân thiện người dùng, dễ học, dễ tiếp cận.



Hình 1. Logo Golang

1.2 Lịch sử hình thành và phát triển của Golang

- Ngôn ngữ Go ban đầu được thiết kế và phát triển bởi một nhóm kĩ sư Google bao gồm **Robert Griesemer**, **Ken Thompson** và **Rob Pike** vào năm **2007**. Mục đích của việc thiết kế ngôn ngữ mới bắt nguồn từ một số phản hồi về tính chất phức tạp của C++11 và nhu cầu thiết kế lại ngôn ngữ C trong môi trường network và multi-core.
- Ngay sau khi có được những ý tưởng và định hình, các kỹ sư bắt đầu nghiên cứu và tiến hành thiết kế và phát triển ngôn ngữ Go, hay còn gọi là Golang, Vào giữa năm **2008**, hầu hết các tính năng được thiết kế trong ngôn ngữ được hoàn thành, họ bắt đầu hiện thực **trình biên dịch (compiler)** và **Go runtime** với **Russ Cox** là nhà phát triển chính.
- Ngôn ngữ Go thường được mô tả là **"Ngôn ngữ C của thế kỉ 21".** Ngôn ngữ Go kế thừa những ý tưởng từ ngôn ngữ C, như là cú pháp, cấu trúc điều khiển, kiểu dữ liệu cơ bản, thủ tục gọi, trả về, con trỏ, v,v... Hình bên dưới mô tả sự liên quan của ngôn ngữ Go với các ngôn ngữ khác.

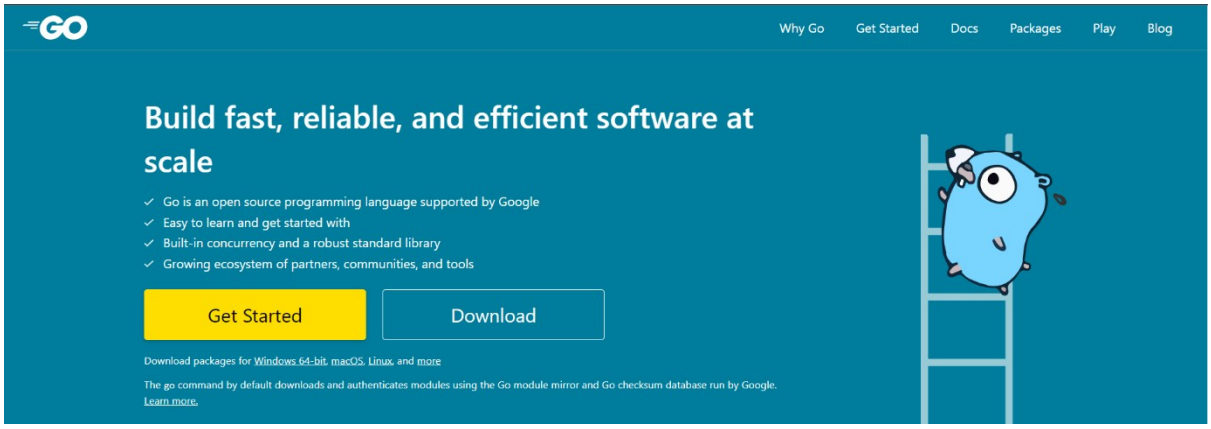


Hình 2. Golang Hierachy

- Chính giữa sơ đồ cho thấy **tính chất hướng đối tượng** và **đóng gói** của Go được kế thừa từ Pascal và những ngôn ngữ liên quan khác dẫn xuất từ chúng. Những từ khóa package, import đến từ ngôn ngữ Modula-2. Nhu vậy có thể kết luận rằng, ngôn ngữ Go được thừa kế chủ yếu từ ngôn ngữ **C** và **Pascal**.
- Vào năm 2009, Golang chính thức được ra mắt và trở thành một dự án **open-source**, dự án đã nhận được sự đóng góp của nhiều cá nhân trong cộng đồng, họ đã đóng góp ý tưởng cũng như mã nguồn để phát triển dự án trở nên hoàn thiện hơn.
- Đến năm 2012, phiên bản đầu tiên của Golang là **Golang 1.0** chính thức được phát hành. Và trong những năm sau đó đến hiện tại, ngôn ngữ Golang ngày càng được cải tiến với nhiều phiên bản mới, phát triển một số tính năng mới cũng như khắc phục được những sai sót trong những phiên bản cũ. Phiên bản mới nhất (latest version) của Golang hiện nay là **go1.18 (released 2022-03-15).**

1.3 Offical Website của Golang

- Trang chủ chính thức (**Offical Website**) của ngôn ngữ Golang là https://go.dev/



Hình 3. Offical Website Of Golang

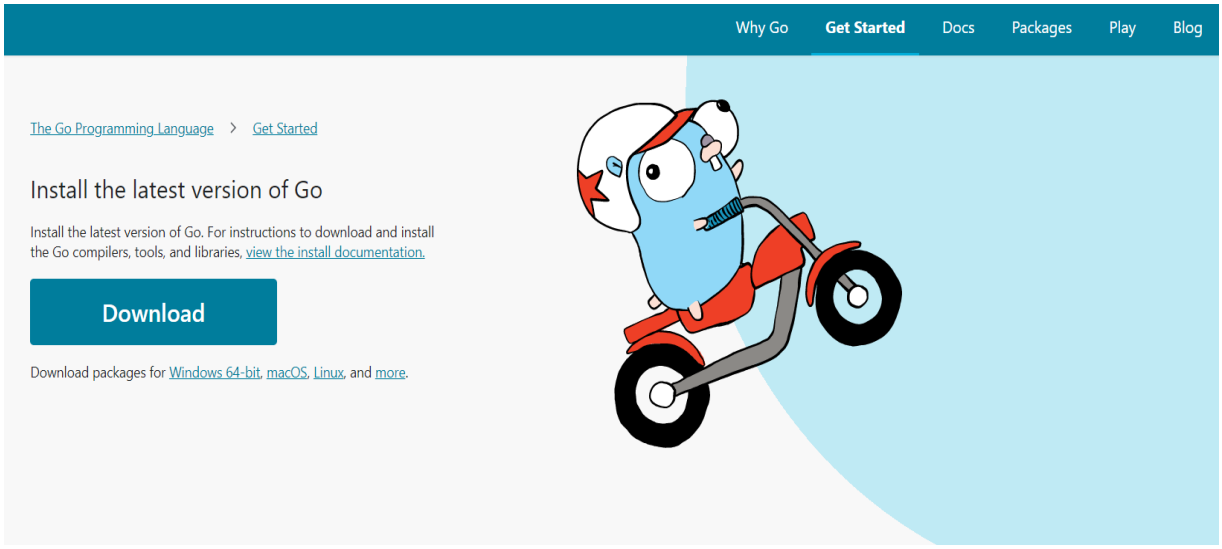
- Đây là trang mà các bạn có thể tham khảo các tutorial, update mới nhất về ngôn ngữ trực tiếp từ phía tác giả, nguồn thông tin được đảm bảo chính xác, đáng tin cậy
- Vai trò của một số mục trên header:
  - **Why Go:** Giới thiệu một số sản phẩm, dịch vụ của các công ty nổi tiếng đang được lập trình bằng ngôn ngữ Golang.

- **Get started:** Nơi chứa các phiên bản cài đặt, đặc biệt là phiên bản mới nhất **(latest version**) của Golang
- **Docs:** Chứa các **tutorial, document** chất lượng nhằm hướng dẫn học tập, lập trình với ngôn ngữ Golang cho cộng đồng.
- **Packages:** Tìm kiếm thông tin các **packages** mà chúng ta thường sử dụng trong Golang.
- **Play:** Đường link liên kết với **The Go Playground** có vai trò như một **IDE online**, giúp mọi người có thể code Golang trên nền tảng online.
- **Blog:** Thông tin về các bài báo khoa học, thông tin mới nhất về ngôn ngữ Golang như thông tin về latest version, When To Use Generics, Go Developer Survey 2021 Results,..

CHƯƠNG 2. LẬP TRÌNH VỚI GOLANG

2.1. Cài đặt môi trường Golang trên Window, MacOS, Linux

Từ trang chủ **go.dev,** chọn **Get Started.** Sau đó cài đặt Golang phù hợp với hệ điều hành của máy tính



Hình 4. Cài đặt Golang

- **Download Go Archive**

OS	Archive name
Windows	go1.4.windows-amd64.msi
MacOS	go1.4.darwin-amd64-osx10.8.pkg
Linux	go1.4.linux-amd64.tar.gz

2.2 Cách biên dịch chương trình Golang

- Ví dụ, ta có source code file **main.go** như sau

```
package main

import "fmt"

func main() {
    fmt.Println("Hello World!")
}
```

- Để compile source code, ta dùng command line như sau

```
> go run main.go
> Hello World!
```

2.3. Cấu trúc một chương trình Golang (Program Structure)

- Quay lại với chương trình **Hello World** của Go

```
package main

import "fmt"

func main() {
      // Everything start with Hello World
      fmt.Println("Hello World!")
}
```

- Dựa vào chương trình trên, ta nhận xét cấu trúc một chương trình Golang như sau:

— **package main**: Khai báo package main. Mỗi chương trình Golang bắt buộc phải khai báo **package**. Package **main** là điểm bắt đầu (**starting point**) để chạy chương trình.

— **import "fmt"**: Import package "fmt", package "fmt" được import để hỗ trợ những hàm nhập/xuất (**I/O**).

— **func main()**: Khai báo hàm main. Hàm main() trong Golang không có tham số và không có giá trị trả về, và là hàm được gọi đầu tiên khi thực thi chương trình. Trong Golang, một hàm được khai báo bằng từ khóa **func.**

- Ngoài ra, chúng ta có thể sử dụng ký hiệu // hoặc **/* */** để comment code trong Golang.

2.4. Một số tính chất cơ bản của Golang

- **Tokens:**

— Token của Go rất đa dạng, có thể là từ khóa (**keyword)**, biến (**identifier)**, hằng (**constant)**, chuỗi (**string)**, hoặc ký hiệu (**symbol**).

```
var x float64
x = 20.0
```

```
fmt.Println(5)
fmt.Println("Hello World!")
fmt.Println(x)
```

- **Line Separator:**

— Trong Golang, các câu lệnh được ngăn cách nhau theo từng dòng, **không cần** sử dụng dấu **chấm phẩy (;)** như ngôn ngữ C/C++

- **Nhập xuất bằng packages fmt:**

— Sử dụng hàm **fmt.Printf** hoặc **fmt.Println** để in ra màn hình.

— Sử dụng hàm **fmt.Scanf** để nhập dữ liệu.

- **Comments:**

— Sử dụng các ký hiệu // và /* */ để comment code trong Golang.

- **Declared variables, packages must be used:**

— Nếu đã khai báo **biến**, **packages** thì bắt buộc phải sử dụng. Nếu không chương trình sẽ báo lỗi.

2.5. Kiểu dữ liệu (Data Types)

— Trong Go, có thể chia kiểu dữ liệu thành 4 nhóm chính như sau:

  — Basic Type: Numeric, String, Boolean.

  — Aggregate type**:** Array và Structures.

  — Reference type**:** Pointers, Slices, Maps, Functions và Channels

  — Interface type

— Đối với kiểu dữ liệu cơ bản (Basic Type), có 3 loại chính:

- **Boolean:** bao gồm 2 giá trị luận lý là **true** và **false**
- **Numeric**: bao gồm các giá trị số như số nguyên (**Integer**), số thực (**Floating Point**) và số phức (**Complex**).
- **String:** là một chuỗi các ký tự (**Char**)

Ví dụ, ta có thể khai báo một số kiểu dữ liệu cơ bản sau

```
package main
import ("fmt")

func main() {
```

```
    var a bool = true      // Boolean
    var b int = 5          // Integer
    var c float32 = 3.14   // Floating point number
    var d string = "Hi!"   // String
    fmt.Println("Boolean: ", a)
    fmt.Println("Integer: ", b)
    fmt.Println("Float:   ", c)
    fmt.Println("String:  ", d)
}
```

- Các kiểu **Boolean**, **String** chỉ chứa một kiểu dữ liệu duy nhất của biến, đối Boolean – kiểu dữ liệu được khai báo của biến là **bool**, đối với String – kiểu dữ liệu được khai báo của biến là **string**.
- Đối với kiểu dữ liệu **Numeric**, ta có thể chia nhỏ thành các kiểu dữ liệu khác nhau như sau:
  - **Integer Type** bao gồm kiểu các **số nguyên có dấu** và các **số nguyên không dấu**. Số nguyên có dấu được biểu diễn bằng int và số nguyên không dấu được biểu diễn bởi uint

uint8	Số nguyên không dấu 8-bit (0 đến 255).
uint16	Số nguyên không dấu 16-bit (0 đến 65535).
uint32	Số nguyên không dấu 32-bit (0 đến 4294967295).
uint64	Số nguyên không dấu 64-bit (0 đến 18446744073709551615).
int8	Số nguyên có dấu 8-bit (-128 đến 127).
int16	Số nguyên có dấu 16-bit (-32768 đến 32767)
int32	Số nguyên có dấu 32-bit (-2147483648 đến 2147483647).
int64	Số nguyên có dấu 8-bit (-9223372036854775808

	đến 9223372036854775807).
int	Cả 2 loại với cùng kích thước 32 hoặc 64 bit.
uint	

- **Floating Type** bao gồm các số thực 32-bit và 64-bit

float32	Số thực dấu phẩy động 32-bit theo chuẩn IEEE 754.
float64	Số thực dấu phẩy động 64-bit theo chuẩn IEEE 754.

- **Complex Type:** Đối với kiểu dữ liệu số phức, float32 và float64 có thể được xem như kiểu của giá trị phần thực và phần ảo của số phức.

complex64	Số phức với phần thực và ảo kiểu float32.
complex128	Số phức với phần thực và ảo kiểu float64.

2.6. Biến (Variables)

a. Định nghĩa

— Biến (Variables) là nơi lưu giữ giá trị của một kiểu dữ liệu nào đó được khai báo. Biến thường được dùng làm đại diện trong quá trình tính toán, giá trị của biến có thể bị thay đổi nhưng kiểu dữ liệu thì không (do Golang là ngôn ngữ có kiểu dữ liệu tĩnh)

b. Quy ước đặt tên biến:

— Quy tắc đặt tên biến, tên hàm trong Golang tương đối giống với C++. Tên của biến, hàm phải được bắt đầu bằng **chữ cái (letter)** hoặc dấu gạch dưới "_"

- Tên biến không được bắt đầu bằng chữ số (0 đến 9) hoặc những ký tự khác (#, @, $,..)
- Tên biến không được trùng với tên các **keyword.**

break	default	func	interface	select
case	defer	Go	map	Struct
chan	else	Goto	package	Switch
const	fallthrough	if	range	Type
continue	for	import	return	Var

Hình 5. Keyword Golang

- Golang là ngôn ngữ có tính chất **case-sensitive,** có nghĩa là **price** và **Price** là hai biến khác nhau.

  Ví dụ:

  - Một số tên biến hợp lệ: myName, Price, numOfToys, _x, x123,…
  - Một số tên biến không hợp lệ: %hello, break, 123x, @gmail,…

c. Cách khai báo biến với từ khóa var

- Ta có thể khai báo biến với từ khóa **var** theo cú pháp sau:

```
var variable_name data_type = value;
```

- Không cần nhất thiết phải khai báo theo đúng cú pháp trên, trong một số trường hợp ta có thể bỏ đi **data_type** hoặc **value.**
  - Trong trường hợp không khai báo **data_type** thì kiểu dữ liệu của biến được suy diễn (**inferred**) từ kiểu dữ liệu của value.
  - Trong trường hợp không khai báo **value** trong trường hợp định nghĩa kiểu dữ liệu của biến trước, khai báo giá trị sau thì giá trị được khai báo phải khớp với kiểu dữ liệu của biến định nghĩa trước đó.
- Ví dụ ta có thể khai báo như sau:

```go
package main

import (
    "fmt"
    "reflect"
)

func main() {
    var var1 = true
    var var2 int
    var2 = 5
    fmt.Println(reflect.TypeOf(var1))
    fmt.Println(var2)
}
```

Output:

```
bool
5
```

- Phương thức **TypeOf** của package **reflect** có thể dùng để trả về kiểu dữ liệu của một biến.
- Ở biến **var1**, ta không khai báo **data_type** của biến => Kiểu dữ liệu của biến được suy diễn từ giá trị **true** => Kiểu dữ liệu của **var1** là **bool.**
- Ở biến **var2,** lúc đầu khai báo biến, ta không khai báo value => Lúc sau ta phải gán giá trị cho biến phù hợp với kiểu dữ liệu trước đó đã khai báo là kiểu **int** (Integer)

d. Khai báo biến với toán tử :=

- Toán tử **:=** có thể được dùng để khai báo biến thay cho câu lệnh **var**. Khi dùng toán tử **:=** thì phải khai báo lun giá trị của biến đó

```
variable_name := value;
```

Ví dụ :

```
package main

import (
    "fmt"
    "reflect"
)

func main() {
    var1 := 15
    fmt.Println(reflect.TypeOf(var1))     // Integer
}
```

e. Khai báo tĩnh (Static Type Declaration)

— Khai báo tĩnh **(Static Type Declaration)** là khai báo **kiểu dữ liệu** của biến trước, có thể khai báo giá trị của biến sau. Điều này cho compier biết có tồn tại một biến đã được khai báo, biết kiểu dữ liệu của biến đó là gì để compier cấp phát vùng nhớ tương ứng cho biến đó. Lúc này compiler **không cần quan tâm** đến **giá trị** của biến lúc khai báo.

— **Ví dụ**

```
package main

import "fmt"

func main() {
    var x float64
    x = 10.0
    fmt.Println(x)
    fmt.Printf("x is of type %T\n", x)
}
```

Output:

```
10
x is of type float64
```

f. Khai báo động (Dynamic Type Declaration)

— Khai báo động **(Dynamic Type Declaration)** là khai báo **giá trị** của biến trước, cách khai báo này yêu cầu compiler phải tự suy diễn **(inferred)** kiểu dữ liệu của biến đó dựa vào giá trị (**value**) đã khai báo của biến. Lúc này compiler **không cần quan tâm** đến **kiểu dữ liệu** của biến lúc khai báo.

— **Ví dụ**

```
package main

import "fmt"

func main() {
    var x = 10.5
    y := 24
    fmt.Println(x)
    fmt.Println(y)
    fmt.Printf("x is of type %T\n", x)
    fmt.Printf("y is of type %T\n", y)
}
```

Output:

```
20
42
x is of type float64
y is of type int
```

g. Khai báo nhiều biến (Mixed Variable Declaration)

— Có thể khai báo nhiều biến cùng lúc trong một lần khai báo (trên cùng một hàng)

— **Ví dụ**:

```
package main

import "fmt"

func main() {
```

```
        var a, b, c = 3, 3.14, "foo"
        fmt.Printf("a is of type %T\n", a)
        fmt.Printf("b is of type %T\n", b)
        fmt.Printf("c is of type %T\n", c)
    }
```

Output:

```
a is of type int
b is of type float64
c is of type string
```

2.7. Hằng (Constants)

- Hằng (**Constants**) là một giá trị cố định mà khi ta đã gán giá trị thì không thể thay đổi được nữa. Hằng có thể có một số kiểu dữ liệu như hằng số nguyên, hằng số thực, hằng chuỗi, …
- Về cách khai báo, hằng cũng được khai báo tương tự như biến nhưng với từ khóa **const** làm tiền tố.
- Hằng không thể khai báo với toán tử **:=**
- Ví dụ

```
package main

import "fmt"

func main() {
   const PI float64 = 3.14
   var RADIUS float64 = 5
   var area float64
   area = PI * RADIUS * RADIUS
   fmt.Printf("Area of Circle : %f", area)
}
```

- Output:

```
Area of Circle : 78.500000
```

- **Nhận xét:** Với những hằng số không đổi như số **PI = 3.14,** ta nên khai báo hằng **(const**) để đảm báo tính nhất quán dữ liệu (**consistent data**)

2.8. Toán tử (Operators)

Trong Golang, toán tử có thể chia thành các loại chính sau:

— **Toán tử toán học** (**Arithmetic Operators**): bao gồm các toán tử dùng trong tính toán như cộng (+), trừ (-), nhân (*), chia (/), chia lấy phần dư (%), increment operator (++), decrement operator (--)

— **Toán tử quan hệ** (**Relational Operators**): bao gồm các toán tử dùng để so sánh các mối quan hệ giữa các đối tượng như so sánh bằng (==), khác (!=), lớn hơn (>), nhỏ hơn (<), lớn hơn hoặc bằng (>=), nhỏ hơn hoặc bằng (<=)

— **Toán tử logic** (**Logic Operators**): được dùng để kết hợp logic của 2 điều kiện hoặc ràng buộc, hoặc dùng để bổ sung thêm điều kiện ban đầu. Bao gồm các toán tử như **AND** (&&) trả về **true** khi cả hai vế cùng đúng, **OR (||)** trả về **true** khi ít nhất một trong hai vế đúng, **NOT** (!) dùng để phủ định lại điều kiện của vế ban đầu.

— **Toán tử bitwise (Bitwise Operators)**: dùng để thực hiện các phép toán theo bit (bit-by-bit operation). Bao gồm một số toán tử như **AND** (&), **OR** (|), **XOR**(^), **dịch trái** (<<), **dịch phải** (>>).

— **Toán tử gán (Assignment Operators):** dùng trong các phép toán gán giá trị cho biến như phép gán (=), các toán tử kết hợp như +=, -=, *=, /=, %=, &=, ^=, |= dùng để thực hiện các phép toán cộng, trừ, nhân, chia,… giá trị bên phải toán tử xong gán lại giá trị mới cho biến.

2.9. Tầm vực (Scope)

- **Tầm vực (Scope)** là một vùng trong chương trình nơi mà các biến đã khai báo (**defined variable**) tồn tại, dựa vào tính chất tầm vực khác nhau mà quyết định quyền truy cập từ bên ngoài hoặc bên trong đối với các biến trong tầm vực.
- Dựa vào phạm vi của biến mà chúng ta có thể chia ra 3 loại tầm vực là:
- Tầm vực cục bộ (Local Scope)
- Tầm vực toàn cục (Global Scope)
- Tầm vực theo khối (Block Scope)

a. Tầm vực cục bộ (Local Scope)

- Các biến cục bộ được khai báo trong hàm sẽ được sử dụng trong phạm vi của hàm đó và sẽ được giải phóng khi hàm được thực thi xong (ra khỏi scope).
- **Ví dụ**

```go
package main

import "fmt"

func main() {
   /* local variable declaration */
   var a, b, c int

   /* actual initialization */
   a = 10
   b = 20
   c = a + b

   fmt.Printf("a = %d, b = %d, c = %d\n", a, b, c)
}
```

Output:

```
a = 10, b = 20, c = 30
```

b. Tầm vực toàn cục (Global Scope)

- Biến được khai báo trong phạm vi toàn cục có thể được sử dụng ở bất kì hàm nào, bất kì vị trí nào trong toàn bộ chương trình, và nó được giải phóng khi ta kết thúc việc thực thi chương trình.
- Nếu trong một scope có cả hai biến global và local trùng tên, chương trình sẽ ưu tiên sử dụng biến local.

- **Ví dụ 1: Trường hợp biến ở global scope được sử dụng ở hàm main**

```
package main

import "fmt"

/* global variable declaration */
var g int

func main() {
   /* local variable declaration */
   var a, b int

   /* actual initialization */
   a = 10
   b = 20
   g = a + b

   fmt.Printf("a = %d, b = %d, g = %d\n", a, b, g)
}
```

Output:

```
a = 10, b = 20, g = 30
```

- **Ví dụ 2: Trường hợp biến ở global scope và local scope trùng tên**

```go
package main

import "fmt"

/* global variable declaration */
var g int = 20

func main() {
   /* local variable declaration */
   var g int = 10

   fmt.Printf("g = %d", g)
}
```

Output:

```
g = 10
```

- **Ví dụ 3: Trường hợp biến ở global scope được truyền vào hàm**

```go
package main

import "fmt"

var name string = "Nguyen Duc An"

func getInfo() {
   var age int = 20
   fmt.Println("Name: ", name)
   fmt.Println("Age: ", age)
}
func main() {
   getInfo()
}
```

Output:

```
Name: Nguyen Duc An
Age: 20
```

— Trong ví dụ trên, khi ta khai báo biến trong phạm vi toàn cục, nó có thể được sử dụng ở bất kì vị trí nào hay trong bất kì hàm nào. Khi ta thực thi chương trình thì hàm main sẽ là hàm được thực thi trước, nội dung hàm main chính là thực hiện gọi hàm **getInfo()**, trong hàm này sẽ khai báo thêm biến cục bộ age, sau đó xuất ra màn hình thông tin tên và tuổi thông qua biến name và age.

c. Tầm vực theo khối (Block Scope)

— Ngoài tầm vực cục bộ và toàn cục, ta còn có một khái niệm nữa là block, có thể hiểu là những lệnh được đặt trong cặp ngoặc nhọn {}, tức các biến được khai báo và sử dụng trong block sẽ được giải thoát khi ra khỏi block.

```go
package main

import "fmt"

func main() {
   var x int = 5.0
   {
      var x int = 10.0
      fmt.Println("Inside Block")
      fmt.Println("x = ", x)
   }
   fmt.Println("Outside Block")
   fmt.Println("x = ", x)
}
```

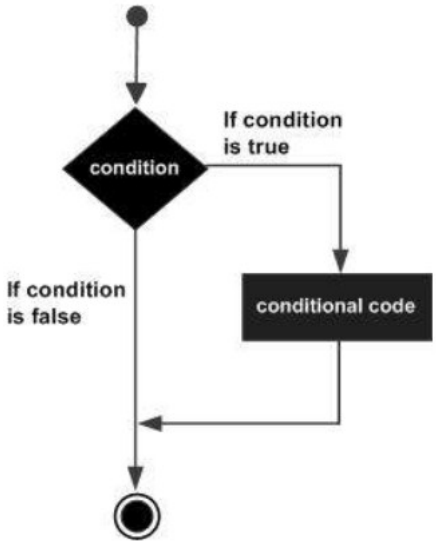
Output:

```
  Inside Block
x = 10
Outside Block
x = 5
```

- Trong ví dụ trên, có thể xem ngoài block là tầm vực cha (trong hàm main), còn trong block {} là tầm vực con. Trong tầm vực cha, biến x được khai báo **x = 5.0**. Nhưng trong tầm vực con, chúng ta khai báo lại biến x = 10.0**. Lúc này, thật sự biến x trong tầm vực con nó khác hoàn toàn với biến x trong tầm vực cha**. Lúc in ra thì trong tầm vực con (**Inside Block)**, lúc in ra thì nó in biến x trong tầm vực con x = 10 (do trong scope local và global có cùng tên biến nên nó sẽ ưu tiên in giá trị biến local). Sau khi ra khỏi scope, lúc này biến x trong tầm vực con sẽ bị giải phóng, biến x còn lại là biến x của tầm vực cha, lúc này ở tầm vực cha (**Outside Block**), in ra giá trị x = 5

2.10. Các câu lệnh rẽ nhánh (Condition Statement)

- Các câu lệnh rẽ nhánh (Condition Statement) có vai trò quyết định trong việc kiểm thử các điều kiện nhằm đưa ra các quyết định trong việc thực hiện chương trình
- Trong việc lập trình, việc kiểm thử các điều kiện và đưa ra quyết định phù hợp là một trong những yêu cầu quan trọng nhất khi giải quyết vấn đề.

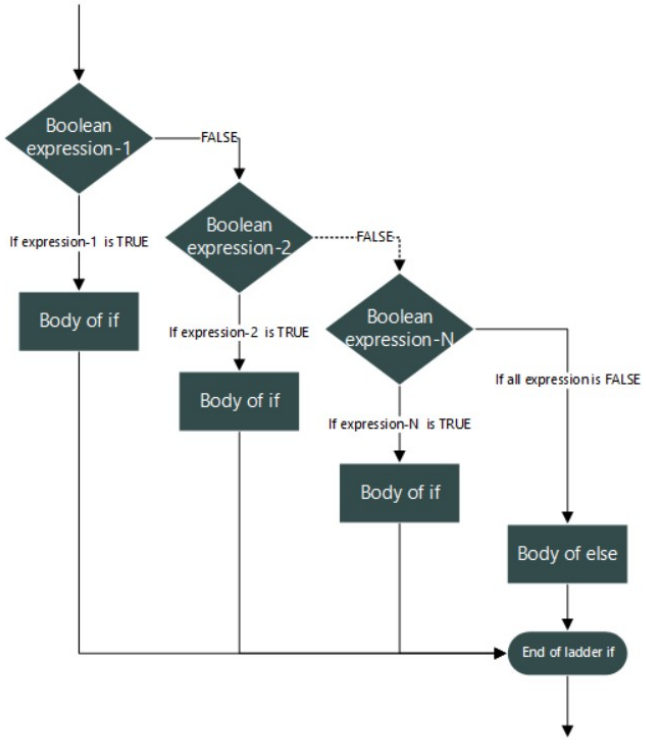


Hình 6. If Condtion

```
if (condition) {
  /* statement(s) will execute if the condition is true */
}
```

- Nếu như điều kiện condition là đúng (true), đoạn mã lệnh bên trong thân của if sẽ được thực thi. Đây là ý tưởng cơ bản nhất về rẽ nhánh.
- Nhưng trong thực tế, không phải luôn luôn chỉ có một điều kiện, nhưng có nhiều điều kiện, mỗi điều kiện sẽ được thực thi theo cách khác nhau. Vì thế ta có thêm khái nhiệm else if và else
- Trong câu cấu trúc điều kiện, câu lệnh điều kiện sẽ được thực thi theo thứ tự từ trên xuống dưới cho đến khi có điều kiện thỏa mãn thì đoạn mã lệnh trong đó sẽ được thực thi. Nếu tất cả các điều kiện phía trên đều không thỏa, mã lệnh trong else sẽ được thực thi.

```
if (condtion 1) {
    /* Executes when the condition 1 is true */
} else if (condition 2) {
     /* Executes when the condition 2 is true */
 } else if (condition 3) {
      /* Executes when the condition 3 is true */
  } else {
       /* executes when the none of the above condition is true */
}
```



Hình 7. If Else Condition

— **Ví dụ chương trình dựa vào điểm trung bình để xếp loại điểm cho sinh viên**

```
package main

import "fmt"

func main() {
   avg := 95
   if avg >= 90 {
      fmt.Println("Grade: A")
   } else if avg >= 80 && avg < 90 {
      fmt.Println("Grade: B")
   } else if avg >= 70 && avg < 80 {
      fmt.Println("Grade: C")
   } else if avg >= 60 && avg < 70 {
      fmt.Println("Grade: D")
   } else {
      fmt.Println("Grade: F")
   }
}
```

Nhận xét:

- Đối với cấu trúc rẽ nhánh trong Golang, sau mỗi lệnh if, if else, else đều bắt buộc phải có cặp dấu {}, bất kể là có 1 hay nhiều dòng lệnh trong thân hàm đi chăng nữa. Đây là điểm khác biệt so với C/C++.
- Các lệnh if else, else bắt buộc phải nằm trên cùng hàng, phía sau dấu ngoặc nhọn của lệnh if/if else đi trước.

Ví dụ nếu chúng ta viết code theo thói quen như ngôn ngữ C++ như sau

```
package main

import "fmt"

func main() {
   n := 5
   if n > 0 {
       fmt.Printf("Positive")
   }
 else if n == 0 {
    fmt.Printf("Zero")
 }
```

```
    else {
        fmt.Printf("Negative")
    }
}
```

Output:

```
.\tempCodeRunnerFile.go:10:2: syntax error: unexpected else, expecting }
```

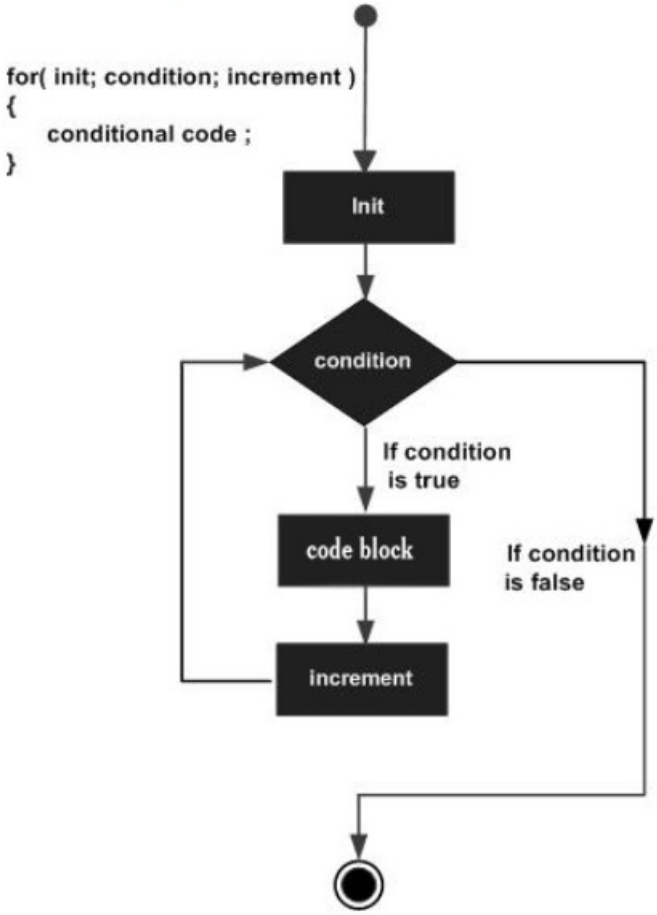
2.11. Các câu lệnh lặp (Loop Statement)

— Ở ngôn ngữ khác như C/C++, Java, Python chúng ta có for, while, do … while để hỗ trợ vòng lặp. Tuy nhiên, do vai trò của các câu lệnh trên là như nhau, để đơn giản hóa thì Golang chỉ hỗ trợ một kiểu vòng lặp duy nhất là **for**

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

— Giải thích syntax:

- **condition:** điều kiện lặp. Ví dụ for x < 5 {} với x đã được khởi tạo giá trị trước khi vào vòng lặp for. Nếu **condition** không được định nghĩa, thì đây là vòng lặp vô hạn.
- **(init, condition, increment):** khởi tạo giá trị biến chạy, điều kiện lặp, tăng giá trị biến chạy. Ví dụ for i := 0; i < 5; i++ {}, khởi tạo biến i = 0; điều kiện lặp là i < 5; biến chạy là i++ (tăng 1 đơn vị sau mỗi lần lặp)
- **Range:** dùng để lặp các đối tượng trong khoảng (chúng ta sẽ tìm hiểu ở phần sau **2.12. Range**)



Hình 8. Vòng lặp for trong Golang

- **Ví dụ 1 : Vòng lặp for có dạng condition**

```
package main

import "fmt"

func main() {
   i := 0
   for i < 5 {
       fmt.Println(i)
       i++
   }
}
```

Output:

```
0
1
2
3
4
```

- **Ví dụ 2 : Vòng lặp for có dạng (init, condition, increment)**

```go
package main

import "fmt"

func main() {
   for i := 0; i < 5; i++ {
       fmt.Println(i)
   }
}
```

Output:

```
0
1
2
3
4
```

- **Ví dụ 3: Vòng lặp vô tận (infinite loop)**

```go
package main

import "fmt"

func main() {
   for {
       fmt.Println("Looping...")
   }
}
```

Output:

```
Looping…
Looping…
Looping…
Looping…
…….
```

— Ngoài ra, để **thoát vòng lặp** ta có thể sử dụng câu lệnh **break,** để bỏ qua một số câu lệnh phía dưới ta sẽ sử dụng câu lệnh **continue.** Về cách sử dụng **break** và **continue** tương tự như bên ngôn ngữ C++.

Ví dụ về câu lệnh break - Chương trình in ra các số từ 0 đến 3

```
package main

import "fmt"

func main() {
   for i := 0; i < 5; i++ {
       if i > 3 {
          break
       }
     fmt.Println(i)
   }
}
```

Ví dụ về câu lệnh continue - Chương trình in ra các số chẵn từ 0 đến 5

```
package main

import "fmt"

func main() {
   for i := 0; i < 5; i++ {
       if i%2 != 0 {
          continue
       }
       fmt.Println(i)
   }
}
```

2.12. Khoảng (Range)

- Từ khóa **range** ra đời nhằm hỗ trợ lặp các đối tượng trong các **cấu trúc dữ liệu** như **array, slice, channel, map**
- Với kiểu dữ liệu là **array**, **slice**, sau mỗi lần lặp kết quả trả về là **index** của các **phần tử** (**items**)
- Với kiểu dữ liệu là **map**, sau mỗi lần lặp kết quả trả về là một cặp **key-value**.
- Với kiểu dữ liệu là **channel,** sau mỗi lần lặp kết quả trả về là phần tử (element) của cấu trúc dữ liệu.
- Range có thể trả về 1 hoặc 2 giá trị, tùy vào nhu cầu người lập trình mà chọn sẽ lấy giá trị trả về nào.

Range expression	1st Value	2nd Value(Optional)
Array or slice a [n]E	index i int	a[i] E
String s string type	index i int	rune int
map m map[K]V	key k K	value m[k] V
channel c chan E	element e E	none

Hình 9. Range Table

Ví dụ

```
package main

import "fmt"

func main() {
  /* create a slice */
  numbers := []int{1, 3, 5, 7, 9, 2, 4, 6, 8}

  /* print the numbers */
  for i := range numbers {
      fmt.Println("Slice item", i, "is", numbers[i])
  }

 /* create a map*/
  countryCapitalMap := map[string]string{"France": "Paris", "Italy":
"Rome", "Japan": "Tokyo"}

  /* print map using keys*/
  for country := range countryCapitalMap {
```

```
        fmt.Println("Capital of", country, "is",
countryCapitalMap[country])
    }
    /* print map using key-value*/
    for country, capital := range countryCapitalMap {
       fmt.Println("Capital of", country, "is", capital)
    }
 }
```

Output:

```
Slice item 0 is 1
Slice item 1 is 3
Slice item 2 is 5
Slice item 3 is 7
Slice item 4 is 9
Slice item 5 is 2
Slice item 6 is 4
Slice item 7 is 6
Slice item 8 is 8
Capital of France is Paris
Capital of Italy is Rome
Capital of Japan is Tokyo
Capital of France is Paris
Capital of Italy is Rome
Capital of Japan is Tokyo
```

2.13. Hàm (Function)

- Hàm (**Function**) giúp chúng ta có thể chia chương trình lớn thành nhiều hàm, nhiều công việc khác nhau, mỗi hàm chỉ thực hiện một chức năng.
- Việc chia chương trình thành nhiều hàm nhỏ giúp code mạch lạc, rõ ràng, dễ hiểu, dễ bảo trì, phát triển các tính năng của chương trình về lâu dài.
- Hàm nhận vào parameter là các biến và trả về một kiểu dữ liệu xác định hoặc là kiểu void.

Ví dụ về chương trình Calculator hai số a, b với giá trị ban đầu của a = 10, b = 5

```
package main

import "fmt"
```

```
func add(x, y int) int {
   return x + y
}
func sub(x, y int) int {
   return x - y
}
func mult(x, y int) int {
   return x * y
}
func div(x, y int) int {
   return x / y
}
func main() {
   var operator string
   var x int = 10
   var y int = 5
   fmt.Println("Nhap operator: ")
   fmt.Scanf("%s", &operator)
   if operator == "+" {
       fmt.Println("Result = ", add(x, y))
   } else if operator == "-" {
       fmt.Println("Result = ", sub(x, y))
   } else if operator == "*" {
       fmt.Println("Result = ", mult(x, y))
   } else if operator == "/" {
       fmt.Println("Result = ", div(x, y))
   } else {
       fmt.Println("Only operator is +, -, *, / ")
   }
}
```

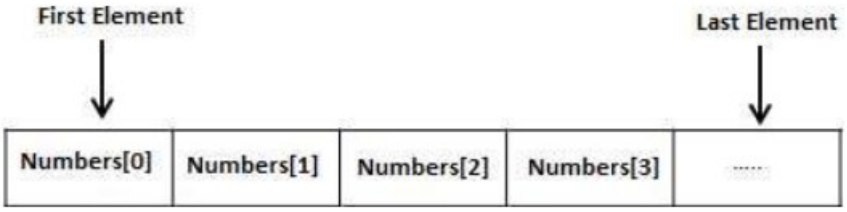
Output:

```
Nhap operator:
+
Result = 15
```

2.14. Mảng (Array)

— Array (Mảng) là một cấu trúc dữ liệu phổ biến ở hầu hết các ngôn ngữ lập trình. Đây là cấu trúc dữ liệu có kích thước cố định (**fixed-size**), có thể coi mảng như một tập hợp các phần tử (**elements**) có cùng kiểu dữ liệu.

— Mảng được cấp phát liên tục (**contiguous allocation**). **Địa chỉ của mảng** là địa chỉ của **phần tử đầu tiên** trong mảng **arr[0]** và các phần tử có địa chỉ tuần tự tăng dần tính từ địa chỉ của phần tử đầu tiên và kiểu dữ liệu của phần tử đó.



Hình 10. Array

a. Khai báo mảng (Declaring Array)

- Khai báo mảng theo cú pháp sau

```
var variable_name [SIZE] variable_type
```

Ví dụ: Khai báo mảng arr có kích thước 10 phần tử và có kiểu int :

```
var arr [10] int
```

b. Khởi tạo mảng (Declaring Array)

- Sau khi khai báo mảng, chúng ta còn có thể khởi tạo các giá trị cho mảng bằng cách để giá trị trong cặp ngoặc nhọn {}. Nếu định nghĩa size, thì số lượng các phần tử của mảng khai báo phải bằng với kích thước size đã định nghĩa

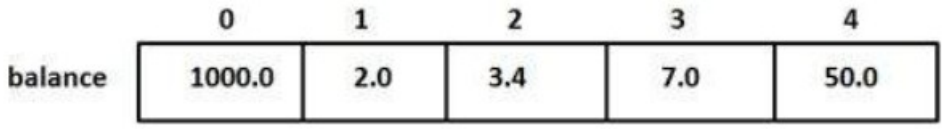
Ví dụ :

```
var balance = [5]float32{1000.0, 2.0, 3.4, 7.0, 50.0}
```

- Ngoài ra, ta cũng có thể không cần định nghĩa size của mảng trong lúc khai báo. Nếu không định nghĩa size, thì kích thước của mảng (size) sẽ được hiểu là số lượng phần tử đã khai báo trong mảng.

Ví dụ :

```
var balance = []float32{1000.0, 2.0, 3.4, 7.0, 50.0}
```



Hình 11. Khởi tạo mảng

c. Truy cập các phần tử của mảng (Accessing Array Elements)

- Trong Golang, index của mảng bắt đầu từ 0.
- Để truy cập giá trị các phần tử của mảng, ta có thể sử dụng toán tử **[]** với độ phức tạp chỉ là O(1).
- Đối với ví dụ trên, ví dụ ta muốn lấy ra phần tử 3.4, lúc này ta nhận thấy 3.4 ở index = 2. Vì vậy để lấy ra phần tử 3.4 của mảng ta sử dụng cú pháp **balance[2]**.

```
float32 salary = balance[2]        // salary = 3.4
```

Ví dụ:

```go
package main

import "fmt"

func main() {
   var arr = [5]int{1, 2, 3, 4, 5}
   for i := 0; i < 5; i++ {
       fmt.Printf("Element[%d] = %d\n", i, arr[i])
   }
}
```

Output:

```
Element[0] = 1
Element[1] = 2
Element[2] = 3
Element[3] = 4
Element[4] = 5
```

2.15. Slices

- Vì kích thước mảng là cố định (**fixed-size**), do đó ta không thể thêm hoặc bớt các phần tử vào một mảng đã đầy các phần tử, cũng không thể tăng kích thước mảng sau khi đã định nghĩa.
- Vì vậy, slice ra đời để giải quyết các vấn đề trên của mảng, kích thước của slice là động (**dynamic-size**) do đó slice cung cấp cho chúng ta một cấu trúc dữ liệu linh hoạt hơn trong quá trình tính toán

a. Khai báo Slice (Defining a slice)

- Để định nghĩa slice, chúng ta có thể định nghĩa như một array mà không khai báo cụ thể kích thước hoặc dùng hàm **make**

```
    var numbers []int /* a slice of unspecified size */
    /* numbers == []int{0,0,0,0,0}*/
    numbers = make([]int,5,5) /* a slice of length 5 and capacity
5*/
```

b. Hàm len() và cap()

```
package main

import "fmt"

func main() {
   var numbers = make([]int, 3, 5)
   printSlice(numbers)
}
func printSlice(x []int) {
   fmt.Printf("len=%d cap=%d slice=%v\n", len(x), cap(x), x)
}
```

Output:

```
len = 3 cap = 5 slice = [0 0 0]
```

c. Subslice

- Ta có thể cắt ra các slice con (**subslice**) của slice bằng toán tử **[lower-bound : upper-bound]**

```
package main

import "fmt"

func main() {
   /* create a slice */
   numbers := []int{0, 1, 2, 3, 4, 5, 6, 7, 8}

   /* print the original slice */
   fmt.Println("numbers ==", numbers)

   /* print the sub slice starting from index [1:4] */
   fmt.Println("numbers[1:4] ==", numbers[1:4])

   /* missing lower bound implies 0*/
   fmt.Println("numbers[:3] ==", numbers[:3])

   /* missing upper bound implies len(s)*/
   fmt.Println("numbers[4:] ==", numbers[4:])
}
```

Output:

```
numbers == [0 1 2 3 4 5 6 7 8]
numbers[1:4] == [1 2 3]
numbers[:3] == [0 1 2]
numbers[4:] == [4 5 6 7 8]
```

d. Hàm append() và copy()

- Hàm append dùng để thêm phần tử vào đằng sau slice
- Hàm copy dùng để tạo bản sao của slice.

```
package main

import "fmt"

func printSlice(x []int) {
```

```
    fmt.Printf("len=%d cap=%d slice=%v\n", len(x), cap(x), x)
}

func main() {
    var numbers []int
    printSlice(numbers)

    /* append allows nil slice */
    numbers = append(numbers, 0)
    printSlice(numbers)

    /* add one element to slice*/
    numbers = append(numbers, 1)
    printSlice(numbers)

    /* add more than one element at a time*/
    numbers = append(numbers, 2, 3, 4)
    printSlice(numbers)

   /* create a slice numbers1 with double the capacity of earlier
slice*/
    numbers1 := make([]int, len(numbers), (cap(numbers))*2)

    /* copy content of numbers to numbers1 */
    copy(numbers1, numbers)
    printSlice(numbers1)
}
```

Output:

```
len=0 cap=0 slice=[]
len=1 cap=1 slice=[0]
len=2 cap=2 slice=[0 1]
len=5 cap=6 slice=[0 1 2 3 4]
len=5 cap=12 slice=[0 1 2 3 4]
```

2.16. Chuỗi (String)

— String là một chuỗi các ký tự (**char**). Trong Go, ta có thể hiểu string là một **slice** các bytes vì nó mang đầy đủ tính chất của slice như len, substring,…

a. Khởi tạo chuỗi (Create String)

```
var greeting = "Hello world!"
```

b. Chiều dài của chuỗi (String Length)

- Sử dụng hàm **len** để lấy ra độ dài của chuỗi

```
package main

import "fmt"

func main() {
   var greeting = "Hello world!"
   fmt.Printf("Length = ")
   fmt.Println(len(greeting))
}
```

Output:

```
Length = 12
```

c. Kết hợp nhiều chuỗi (Concatenating String)

- Sử dụng hàm **Join** để kết hợp các chuỗi.

```
package main

import (
   "fmt"
   "strings"
)
func main() {
   greetings := []string{"Hello", "world!"}
   fmt.Println(strings.Join(greetings, " "))
}
```

Output:

```
Hello World !
```

2.17. Cấu trúc (Structures)

— Structures là một cấu trúc dữ liệu do người dùng tự định nghĩa (user-defined), dùng để kết hợp nhiều kiểu dữ liệu lại với nhau.

— Ví dụ, trong chương trình quản lý thư viện, nếu các bạn muốn định nghĩa kiểu dữ liệu **Book** thì ta có thể định nghĩa cấu trúc dữ liệu **struct Book** bao gồm các thuộc tính như title (string), author (string), subject (string), book_id (int)

a. Khởi tạo struct (Defining Structure)

```
type struct_variable_type struct {
      member definition;
      member definition;
      …
      member definition;
}
```

b. Truy cập các thành phần dữ liệu của struct (Accessing Structure Member)

- Dùng toán tử **(.)** để truy cập các thành phần của struct, tầm vực truy cập các thuộc tính của struct là **public**.

Ví dụ về chương trình quản lý thư viện với struct Book

```go
package main

import "fmt"

type Books struct {
   title   string
   author  string
   subject string
   book_id int
}

func main() {
  var Book1 Books /* Declare Book1 of type Book */

   /* book 1 specification */
   Book1.title = "Go Programming"
   Book1.author = "Mahesh Kumar"
   Book1.subject = "Go Programming Tutorial"
   Book1.book_id = 6495407
```

```
    /* print Book1 info */
    fmt.Printf("Book 1 title : %s\n", Book1.title)
    fmt.Printf("Book 1 author : %s\n", Book1.author)
    fmt.Printf("Book 1 subject : %s\n", Book1.subject)
    fmt.Printf("Book 1 book_id : %d\n", Book1.book_id)
}
```

Output:

```
Book 1 title : Go Programming
Book 1 author : Mahesh Kumar
Book 1 subject : Go Programming Tutorial
Book 1 book_id : 6495407
```

2.18. Con trỏ (Pointers)

- Do được kế thừa và phát triển từ ngôn ngữ C/C++, nên Golang là một trong số những ngôn ngữ có tồn tại khái niệm con trỏ (**pointers)**.
- Con trỏ (pointers) được dùng để trỏ đến địa chỉ của một biến, hoặc một vùng nhớ. Giá trị của con trỏ là địa chỉ của vùng nhớ pointer trỏ tới. Tính chất của con trỏ (pointer) trong Golang tương đối giống với con trỏ trong C/C++.
- Con trỏ (Pointers) thường được sử dụng trong một số task liên quan đến truyền tham trị (**call by reference**) hoặc truyền con trỏ.
- Để truy cập địa chỉ của biến hoặc vùng nhớ, ta có thể sử dụng toán tử **&**

```
package main

import "fmt"

func main() {
   var a int = 10
   fmt.Printf("Address of a variable: %x\n", &a)
}
```

Output:

```
Address of a variable: c0000aa058
```

a. Khai báo con trỏ (pointer)

- Con trỏ (pointer) được khai báo theo cú pháp sau

```
var var_name *var-type
```

Ví dụ :

```
var iPointer *int     /* pointer to an integer */
var fPointer *float32 /* pointer to a float */
```

b. Cách sử dụng con trỏ

- Con trỏ được dùng để thay đổi giá trị tại một vùng nhớ mà nó trỏ tới.
- Con trỏ thường được sử dụng trong việc xây dựng các cấu trúc dữ liệu phức tạp như Linked List, Stack, Queue,…Các cấu trúc này không có sẵn trong Golang nếu cần người dùng phải tự định nghĩa.
- Để sử dụng con trỏ ta có thể tuân theo các nguyên tắc sau :
  i. Khởi tạo con trỏ
  ii. Gán địa chỉ của vùng nhớ cần trỏ đến cho pointer
  iii. Truy cập, thay đổi giá trị của vùng nhớ mà con trỏ đến bằng cách thay đổi giá trị của con trỏ.

```
package main

import "fmt"

func main() {
    var a int = 20 /* actual variable declaration */
    var ip *int    /* pointer variable declaration */
    ip = &a        /* store address of a in pointer variable*/

    fmt.Printf("Address of a variable: %x\n", &a)



    /* address stored in pointer variable */
    fmt.Printf("Address stored in ip variable: %x\n", ip)
    /* access the value using the pointer */
    fmt.Printf("Value of *ip variable: %d\n", *ip)
}
```

Output:

```
Address of a variable: c0000aa058
Address stored in ip variable: c0000aa058
Value of *ip variable: 20
```

c. Con trỏ nil (Nil Pointer)

- Con trỏ nil (Nil pointer) là con trỏ được định nghĩa nhưng không có địa chỉ trỏ tới.
- Con trỏ nil (Nil pointer) được xác định ở địa chỉ 0 trong một số thư viện của Golang.

```go
package main

import "fmt"

func main() {
    var ptr *int

    fmt.Printf("The value of ptr is : %x\n", ptr)
}
```

Output:

```
The value of ptr is : 0
```

CHƯƠNG 3. ĐÁNH GIÁ ĐIỂM MẠNH, ĐIỂM YẾU VÀ ỨNG DỤNG CỦA GOLANG

3.1 Điểm mạnh của Golang

Ngôn ngữ lập trình Golang ra đời nhằm nâng cao năng suất của những phần mềm, đặc biệt là ở những lĩnh vực như **multi-core processing** hay **network**.

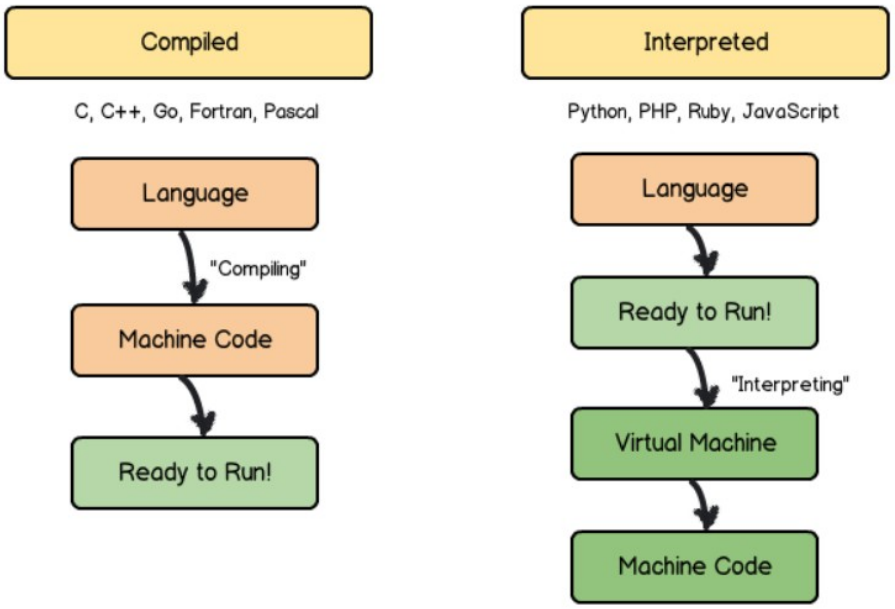
Một số ưu điểm nổi bật của ngôn ngữ Golang:

- **Go kế thừa được những ưu điểm của nhiều ngôn ngữ lập trình khác nhau**
  - Go thừa hưởng những ưu điểm từ C/C++ như cấu trúc rõ ràng, chặt chẽ, là ngôn ngữ compiled nên tốc độ xử lý nhanh hơn những ngôn ngữ intepretered
  - Bên cạnh đó, Go cũng thừa hưởng một số ưu điểm từ các ngôn ngữ intepretered như Python, Perl ở đặc điểm code ngắn gọn dễ dàng, kiểu dữ liệu có thể tự suy diễn mà không cần khai báo trước, giúp code linh hoạt, dễ đọc.
  - Ngoài ra, Go còn được phát triển thêm một số tính năng mới như **Goroutines** giúp xử lý các hoạt động song song (parallel) hoặc đồng thời (concurrency) hiệu quả như Java.
- **Go là ngôn ngữ có cú pháp đơn giản, ngắn gọn, dễ hiểu**
  - Cú pháp câu lệnh của Golang tương đối giống với C/C++, yêu cầu phải tuân thủ chặt chẽ quy tắc các câu lệnh để đảm bảo tính rõ ràng, chính xác khi lập trình. Tuy nhiên, cú pháp câu lệnh đã được đơn giản hóa để code ngắn gọn, dễ đọc.
  - Cú pháp trong Golang đơn giản hơn C/C++ ở vài đặc điểm như không cần **dấu chấm phẩy** cuối mỗi câu lệnh; không cần khai báo kiểu dữ liệu ngay từ đầu mà có thể tự suy diễn (inferred) từ giá trị đầu vào, có thể khai báo biến bằng một số cú pháp đơn giản bằng toán tử **:= (ví dụ x := 0).**

- **Go là một ngôn ngữ lập trình biên dịch (compiled language)**

- Ở các ngôn ngữ biên dịch **(Compiled)** như C, C++, Go, Pascal thì code sẽ được chuyển (Compiled) sang mã máy ở dạng các bit nhị phân (binaries) và sẵn sàng để chạy ngay và luôn.

- Với các ngôn ngữ thông dịch **(Interpreted)** như Python, PHP, Ruby, Javascript thì code khi thực thi sẽ được chuyển sang byte code, với mã byte code thì trình thông dich (virtual machines) mới hiểu được, sau đó trình thông dịch này sẽ tiếp tục chuyển sang dạng nhị phân (binaries) để mà vi xử lý máy tính có thể hiểu được.



Hình 12. Interpreted and Compiled

- Do đó, Go là ngôn ngữ biên dịch (Compiled) nên code sẽ được chuyển sang dạng (binaries) để thực thi chứ không cần thông qua trình thông dịch nữa, nên nó sẽ tăng được hiệu suất chương trình.
- Trên thực tế, Go là ngôn ngữ có hiệu suất (performance) cao gần bằng so với các ngôn ngữ lập trình như C, C++.

- **Go là một ngôn ngữ có kiểu dữ liệu tĩnh (statically-typed language) nhưng vẫn có hỗ trợ tính năng tự suy diễn kiểu dữ liệu (inferred type)**

- Golang là một ngôn ngữ sử dụng biến tĩnh, do đó complier trong quá trình biên dịch mã phải đáp ứng việc chuyển đổi kiểu và kiểm tra sự tương thích của biến, hỗ trợ khai báo động như mảng động, con trỏ. Một đặc điểm nổi bật của Go so với C/C++ là khả năng tự suy diễn kiểu dữ liệu dựa vào giá trị trong trường hợp kiểu dữ liệu chưa được định nghĩa, điều này có thể giúp code gọn gang, dễ hiểu hơn.

- Kiểu dữ liệu của biến sau khi được xác định sẽ không được thay đổi trong suốt quá trình thực thi chương trình, điều này giúp tránh lỗi khi chương trình được thực thi, nếu gán cho biến sai kiểu thì chương trình sẽ báo lỗi ngay lúc compile nhằm đảm bảo chương trình thực hiện chính xác, tiết kiệm thời gian thực thi.

- **Go hỗ trợ việc lập trình đồng bộ (Concurrency) nhờ vào Goroutines**

- Các ngôn ngữ như Python hay Java là các ngôn ngữ sinh ra vào thời kỳ của môi trường đơn luồng (single threaded environment) nhưng hầu như các ngôn ngữ này đều hỗ trợ tính đa luồng (multi-threading). Nhưng có một số vấn đề đã phát sinh như concurrent execution, threading-locking, race conditions và deadlocks đã làm cho việc tạo ra các ứng dụng đa luồng trên các ngôn ngữ này cực kỳ khó khăn.
- Do đó, Go được thiết kế và phát triển với ý tưởng nhằm hỗ trợ việc thực thi đồng thời (concurrency), **Go có goroutines thay cho threads**.
- Goroutine có ngăn xếp phân khúc có thể mở rộng (growable segmented stacks). Điều này có nghĩa là nó sẽ sử dụng nhiều bộ nhớ RAM hơn nếu điều đó là cần thiết.
- Goroutines có thời gian khởi động nhanh hơn là threads.
- Goroutines có các channel và giữa các channel này có thể giao tiếp với nhau.

- **Go là một ngôn ngữ có khả năng tương thích cực kỳ cao**

- Một trong những điều kiện cần thiết ở một ngôn ngữ lập trình là sự tương thích của nó đối với những hệ thống khác nhau.
- Ngôn ngữ Golang có thể liên kết với thư viện C bên ngoài lẫn hệ thống native bên trong. Ví dụ trong Docker, Go interface với các chức năng Linux low-level, cgroups và namespace để hoạt động với container.

- **Go có hỗ trợ tính năng garbage collection**

- Đây là một tính năng vô cùng mạnh của Golang, giúp các lập trình viên có thể sử dụng bộ nhớ một cách an toàn, tự động giải phóng bộ nhớ sau khi sử dụng.

- **Go là một ngôn ngữ đa hỗ trợ với sự có mặt của các toolchain**

- Go có mặt trong nhiều hệ điều hành như macOS, Window, Linux, thậm chí có thể được xem như một container trong Docker.

- Toolchain của Go luôn có sẵn trong thư viện của những hệ điều hành trên giúp cho mọi người có thể dễ dàng cài đặt, làm việc với Golang.

3.2 Điểm yếu của Golang

Một số điểm yếu của ngôn ngữ Golang:

- **Go không có hỗ trợ Generics**
  - Generics có thể được hiểu là giống kiểu template của C++ như **type T** khi định nghĩa lớp, hàm, … Generics giúp tổng quát hóa chung cho mọi kiểu dữ liệu.
  - Việc không hỗ trợ Generics nên Golang không cho phép nạp chồng hàm, lớp với các kiểu dữ liệu đầu vào khác nhau ví dụ List<Int>, List<Double>,…
  - Do đó gần như chúng ta phải viết lặp lại những câu lệnh tương tự chỉ với kiểu dữ liệu trả về khác nhau. Ví dụ chúng ta chỉ muốn viết hàm tính tổng là sum nhưng có hai yêu cầu về kiểu dữ liệu trả về như double sum(x, y) và int sum(x,y), lúc này chúng ta phải viết lại hai hàm riêng biệt chứ không thể tái sử dụng mã bằng template được.
- **Go gần như lượt bỏ hầu hết các tính chất quan trọng của lập trình hướng đối tượng (OOP)**
  - Có thể khẳng định Golang không phải là một ngôn ngữ phù hợp cho lập trình hướng đối tượng (OOP).
  - Các tính chất OOP bị lượt bỏ dẫn đến việc không thể áp dụng các mẫu thiết kế, chia module cho chương trình được viết bằng Golang. Do đó đoạn code trong các chương trình lớn trở nên phức tạp, khó quản lý.
- **Kích thước của các chương trình Golang quá lớn**
  - Đánh đổi với cú pháp ngắn gọn là một câu lệnh Golang là sự kết hợp câu lệnh nhỏ lại với nhau. Do đó kích thước của một câu lệnh Golang là lớn.
  - Khi được biên dịch sang mã máy, dĩ nhiên dù chỉ một câu lệnh nhưng nếu thực hiện cùng lúc nhiều công việc sẽ gây tiêu tốn tài nguyên, tốc độ xử lý chậm.
- **Golang thiếu một số thư viện cần thiết**
  - Golang rất nhẹ và nhỏ gọn tuy nhiên chính sự tối giản đó nó đã làm mất đi một số tính năng cần thiết của một ngôn ngữ lập trình.

  - Thư viện của Golang cũng không rộng rãi như các ngôn ngữ khác. Điều đó làm hạn chế khả năng làm việc của người lập trình.
- **Golang chưa có bất kỳ framework nào hỗ trợ**
  - So với các ngôn ngữ khác như Javascript thì có framework là React, Angular, Ruby thì có Ruby on Rails, Python thì có Django,… thì Golang hiện tại chưa có bất cứ framework nào hỗ trợ, điều này có lẽ do Golang ra đời khá muộn, cộng đồng và công nghệ chưa đủ mạnh để tạo ra một framework đủ mạnh cho Golang.

3.3. Khi nào nên sử dụng Golang ?

- **Nhận xét:**
  - Golang có sự cân bằng giữa hai yếu tố là **hiệu suất chương trình (performance)** và **thời gian lập trình (coding time).** Với ngôn ngữ C/C++, chúng ta cần ít thời gian để build chương trình do đó hiệu suất chương trình cao hơn nhưng đánh đổi lại người lập trình phải mất nhiều thời gian để coding vì mã nguồn dài và phức tạp do cú pháp chặt chẽ, hơn nữa lại thiếu nhiều hàm, thư viện hỗ trợ. Còn đối với các ngôn ngữ như JavaScript hay Python có cú pháp ngắn gọn hay hỗ trợ khai báo dữ liệu động sẽ giúp người lập trình có thể coding nhanh hơn nhưng lại mất nhiều thời gian hơn khi build chương trình. Golang ra đời dựa trên việc học hỏi, kết hợp những ưu điểm của cả hai do đó chương trình Golang có hiệu năng tương đương với các ngôn ngữ như C/C++ nhưng lại có lợi thế về cú pháp đơn giản, mã nguồn dễ đọc, dễ lập trình của các ngôn ngữ như Python, Javascript.
  - Golang có sự đơn giản về cú pháp tuy nhiên chính vì sự đơn giản đó cũng đã gây ra cho Golang một số nhược điểm vì quá ưu tiên sự đơn giản trong cú pháp mà Golang đã lượt bỏ đi một số thư viện, tính năng cần thiết, cũng như do gộp quá nhiều câu lệnh nhỏ trong một câu lệnh lớn dẫn đến kích thước của chương trình Golang vô cùng lớn.
  - **Kết luận:**
  - Đối với những dự án lớn, những dự án với lượng code nhiều và quy mô phức tạp, cũng như khi được đảm bảo về mặt vật chất như máy tính có hiệu năng đủ

mạnh, phần cứng máy tính có thể hỗ trợ tính toán và build các chương trình lớn, lúc này chúng ta nên sử dụng Go để đơn giản hóa mã nguồn, dễ lập trình, tiết kiệm thời gian lập trình (coding time)

- Còn đối với những chương trình nhỏ và đơn giản thì chúng ta không nên sử dụng Go vì những câu lệnh của Go tuy đơn giản nhưng có kích thước lớn sẽ gây lãng phí tài nguyên.

3.4 Ứng dụng của Golang

Một số ứng dụng phổ biến của Golang là:

- **Phân phối các network service (dịch vụ mạng):** Các chương trình ứng dụng mạng (network application) chủ yếu là dựa vào concurrency và các tính năng native concurrency của Go, các **goroutines** và các **channel,** rất phù hợp cho các tác vụ đó. Do đó, có nhiều dự án Go dành cho mạng, các chức năng distributed (phân phối) và dịch vụ đám mây: API, web server, minimal frameworks cho các web application và các loại tương tự.
- **Sự phát triển của cloud-native:** Các tính năng concurrency và network của Go và tính linh hoạt cao của nó làm cho nó phù hợp với việc xây dựng các ứng dụng cloud-native. Trên thực tế, Go đã được sử dụng để xây dựng một trong những nền tảng phát triển ứng dụng dựa trên cloud-native, ứng dụng hệ thống containerization Docker.
- **Thay thế cơ sở hạ tầng hiện có:** Phần lớn các phần mềm phụ thuộc vào cơ sở hạ tầng Internet đã lạc hậu. Việc viết lại những thứ như vậy bằng Go mang lại nhiều lợi ích, như giữ an toàn bộ nhớ tốt hơn, triển khai trên nhiều nền tảng dễ dàng hơn và một code base “sạch” để hỗ trợ bảo trì trong tương lai.

KẾT LUẬN

Trong bài tập lớn này, chúng ta đã cùng nhau tìm hiểu tổng quát về ngôn ngữ Golang, biết được cách lập trình với ngôn ngữ Golang là như thế nào cũng như biết thêm về những điểm mạnh, điểm yếu của ngôn ngữ rồi từ đó trả lời cho câu hỏi khi nào nên sử dụng ngôn ngữ Golang.

Đây là một ngôn ngữ trẻ, còn nhiều tiềm năng phát triển trong tương lai. Dù đã đạt được một số thành công nhất định nhưng để thật sự trở thành một ngôn ngữ hoàn thiện, Golang cần phải khắc phục, hoàn thiện thêm những điểm yếu về kiến trúc ngôn ngữ và nâng cấp thêm nhiều thư viện, tính năng mới hỗ trợ cho người dùng trong quá trình lập trình. Với những điểm mạnh và đặc điểm nổi bật của mình, nhiều khả năng trong tương lai Golang sẽ thật sự trở thành một trong những ngôn ngữ được nhiều lập trình viên ưa chuộng và sử dụng nhiều hơn.

Đối với bản thân em, đây là một trải nghiệm để em tìm hiểu thêm về một ngôn ngữ mới và lạ như Golang - ngôn ngữ có sự kết hợp những ưu điểm của nhiều ngôn ngữ lập trình khác nhau. Trong tương lai, em sẽ cố gắng học hỏi thêm và nghiên cứu cách sử dụng ngôn ngữ Golang vào một vài dự án thực tế.

TÀI LIỆU THAM KHẢO

[1] **Offical Website Golang, go.dev,** truy cập từ The Go Programming Language

[2] **Golang là gì và tại sao bạn nên học Go ?,** *tác giả Võ Xuân Phong*, truy cập từ https://topdev.vn/blog/golang-la-gi-va-tai-sao-ban-nen-hoc-go/

[3] **Go Tutorial**, *website Tutorials Point*, truy cập từ Go Tutorial (tutorialspoint.com)

[4] **Go Tutorial**, *website Tutorials Point*, truy cập từ Go Tutorial (w3schools.com)

[5] **The Go Programming Language**, *Brian Kernighan*, 2015.

[6] **Golang – Geeksforgeeks**, *website geeksforgeeks*, truy cập từ Golang - GeeksforGeeks

[7] **Một số tính năng nổi bật của Ngôn ngữ lập trình Golang**, *TechMaster*, truy cập từ: https://techmaster.vn/posts/35409/mot-so-tinh-nang-noi-bat-cua-ngon-ngu-lap-trinh-golang

[8] **Giới thiệu ngôn ngữ lập trình GoLang**, *Tran Gia Nhuan*, truy cập từ: https://viblo.asia/p/gioi-thieu-ngon-ngu-lap-trinh-golang-1VgZv9VrKAw

[9] **Tại sao bạn nên sử dụng Golang ?,** *Blog Teky.edu.vn*, truy cập từ https://teky.edu.vn/blog/ngon-ngu-golang/

[10] **Go (Ngôn ngữ lập trình),** *Wikipedia*, truy cập từ: Go (programming language) - Wikipedia