ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN
LẬP TRÌNH NÂNG CAO (CO2039)

TÌM HIỂU VỀ NGÔN NGỮ LẬP TRÌNH GOLANG

GV hướng dẫn: TS. Trương Tuấn Anh
SV thực hiện: Huỳnh Tấn Lộc - 2010391

Thành phố Hồ Chí Minh, Tháng 5 năm 2022

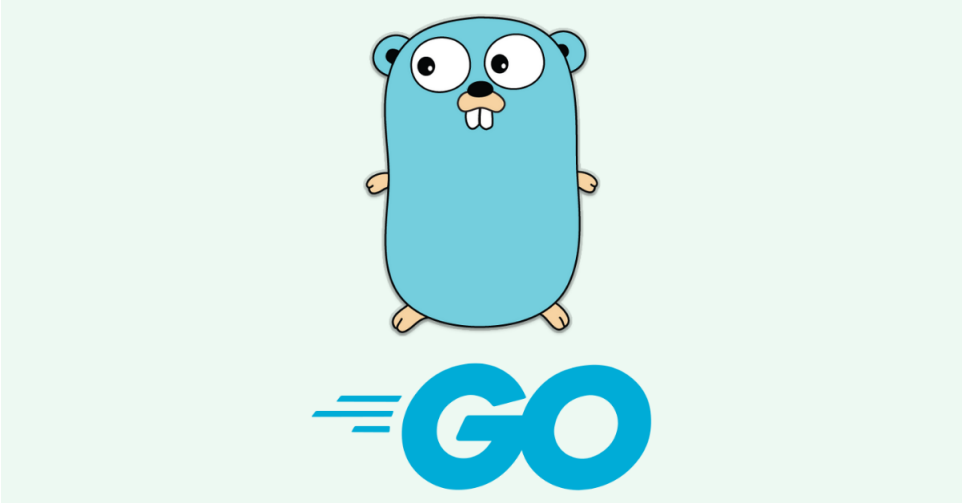
Mục lục

Danh mục hình ảnh

1 Giới thiệu chung về Golang

1.1 Golang là gì?

Ngôn ngữ Go hay còn gọi là **Golang** là một ngôn ngữ lập trình mới được phát triển tại Google vào năm 2007 bởi Robert Griesemer, Rob Pike và Ken Thompson. Nó được kỳ vọng sẽ giúp ngành công nghiệp phần mềm khai thác tối đa nền tảng đa lõi của bộ vi xử lý và hoạt động đa nhiệm tốt hơn.



Hình 1: *Ngôn ngữ lập trình Golang*

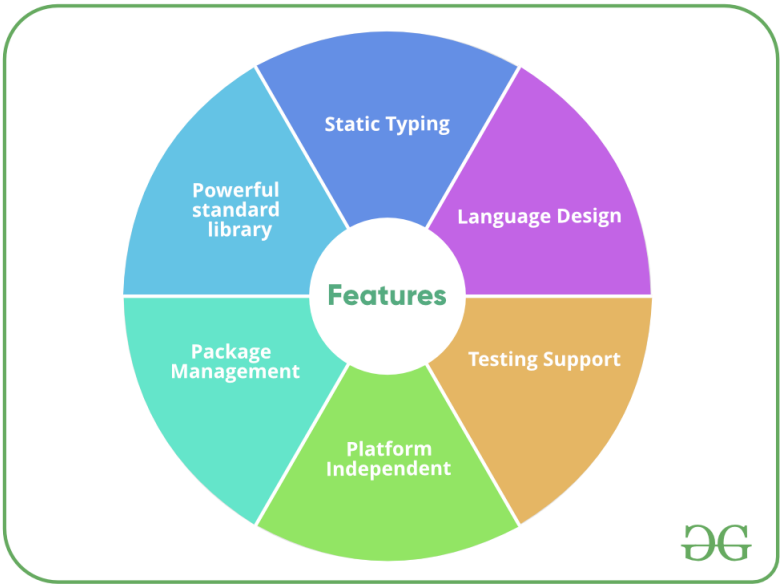
Golang thuộc kiểu **ngôn ngữ lập trình biên dịch** (compiled language), điều này có nghĩa tập tin chứa mã nguồn (source code) viết bằng Golang không thể chạy được trực tiếp bởi máy tính. Thay vào đó chúng ta cần biên dịch (compile) tập tin này sử dụng phần mềm biên dịch mã nguồn (hay compiler) để tạo ra một tập tin khác chứa mã máy. Sau đó máy tính mới có thể chạy tập tin đã được biên soạn này.

Go là một **ngôn ngữ lập trình mã nguồn mở** (open-source). Ta có thể đóng góp cho nó bằng cách tạo các đề xuất mới, sửa lỗi, làm cho nó nhanh hơn. Hiện nay, có hàng triệu lập trình viên trên thế giới đang sử dụng Go và ngày càng có nhiều hơn. Sự thành công của Go đã vượt qua sự mong đợi của người tạo ra nó.

1.2 Đặc điểm chung của ngôn ngữ lập trình Golang

Go có rất nhiều đặc điểm, nhưng có những điểm chung có thể liệt kê như sau:

- Go là ngôn ngữ lập trình biên dịch (compiled language), các chương trình hay phần mềm viết bởi Go cần phải được biên dịch (compile) trước khi có thể chạy được.
- Cú pháp đơn giản: Cú pháp của Go cũng tương tự như C/C++ nhưng được thiết kế cho đơn giản hơn, dễ học, dễ nhớ hơn. Ta không cần phải đặt dấu ";" sau mỗi câu lệnh, điều này sẽ làm giảm tỷ lệ xảy ra lỗi trong compile-time.
- Case sensitive: Go là ngôn ngữ phân biệt rõ ràng chữ hoa và chữ thường. Ví dụ: *Programming* trong Go sẽ khác biệt hoàn toàn so với *programming*.
- Go là một **ngôn ngữ lập trình kiểu tĩnh** (static type), các biến và dữ liệu trong suốt chương trình Go sẽ không thay đổi.
- Go sở hữu một hệ thống các **thư viện chuẩn** (standard library) cực khổng lồ, điều này sẽ hỗ trợ lập trình viên tận dụng những nguồn tài nguyên đã được lập trình sẵn, từ đó sẽ giúp giảm đáng kể thời gian lập trình ra một hệ thống.

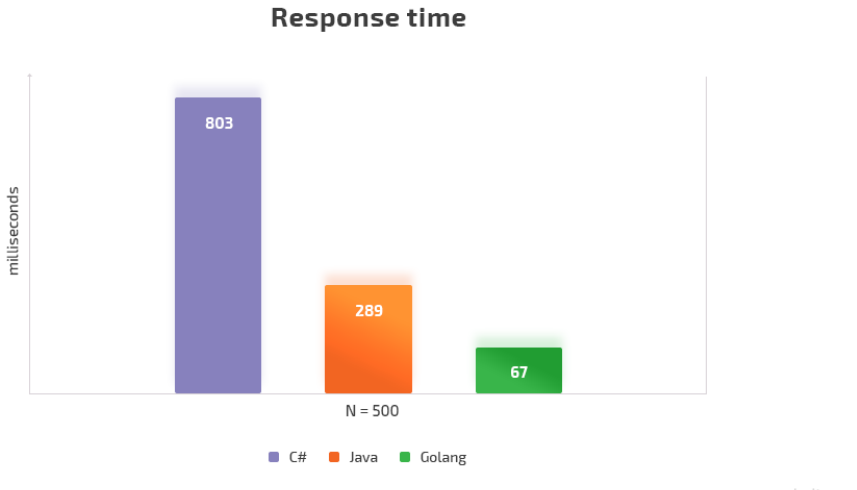


Hình 2: *Đặc điểm chung của ngôn ngữ Go*

- Ngôn ngữ Go thực thi không phụ thuộc vào nền tảng trên máy tính, ví dụ như kernel của hệ điều hành, vì vậy sẽ hạn chế việc hao tổn tài nguyên của máy chủ, đồng thời thúc đẩy tốc độ xử lý nhanh, ổn định hơn gấp nhiều lần so với các ngôn ngữ phụ thuộc nền tảng.
- Go quản lý source code dự trên các package, điều này sẽ làm cho ngôn ngữ quản lý, chia sẻ các biến dữ liệu được tối ưu nhất, phù hợp với việc thiết kế nên những phần mềm, hệ thống có chứa nhiều thành phần khác nhau.
- Go hỗ trợ rất nhiều cho lập trình viên trong việc lập trình, cụ thể là tối ưu các dòng lệnh không có nghĩa, làm giảm hiệu suất của chương trình. Ngoài ra, Go còn có cách trình dọn rác (garbage collection) hỗ trợ người lập trình không bị tràn bộ nhớ bởi những dữ liệu cấp phát chưa được giải phóng.

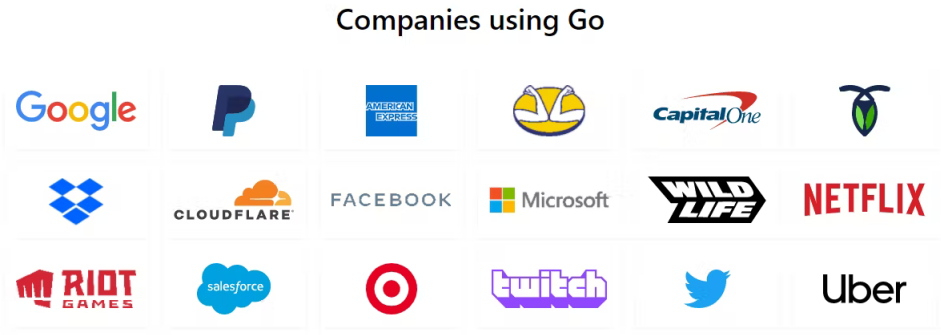
1.3 Lợi ích của việc dùng Golang

Đặc điểm nổi bật nhất của Go mà các ngôn ngữ khác khó đánh bại đó là tốc độ biên dịch và hỗ trợ xử lý đa luồng (multi-threading). Chính đặc điểm này đã làm nên tên tuổi của Go mặc dù đây là ngôn ngữ có tuổi đời khá trẻ. Go thường được dùng để xây dựng nên các phần mềm, ứng dụng chạy chạy trên web.



Hình 3: *Thời gian phản hồi của một số ngôn ngữ lập trình trong Microservices*

Đây là ngôn ngữ có các thư viện giúp kết nối việc xử lý đến các cơ sở dữ liệu nổi tiếng như MySQL, PostgreSQL,... giúp ta có thể lập trình nên những ứng dụng mạnh mẽ hơn rất nhiều.



Hình 4: *Một số công ty lớn sử dụng Golang*

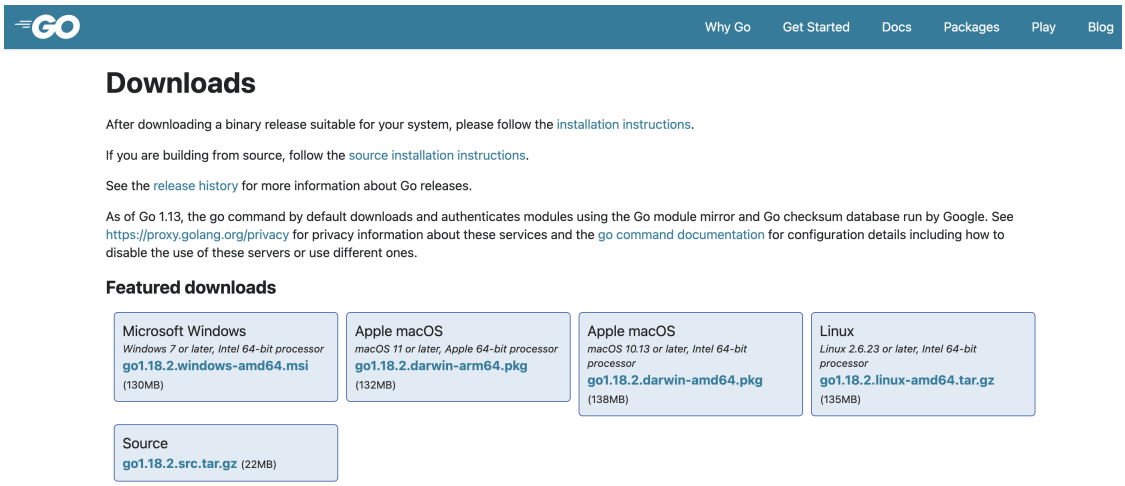
Có thể nói trong 15 năm kể từ khi được phát triển, Go đã trở thành một trong những ngôn ngữ được quan tâm nhất hiện nay. Nó được sử dụng để xây dựng các chương trình của rất nhiều công ty lớn trên thế giới như Google, Facebook, IBM, Adobe, Uber, Twitch, Intel, Dropbox, SoundCloud, SendGrid,... và chắc chắn rằng trong tương lai, nó sẽ được sử dụng nhiều hơn nữa bởi những tính năng tuyệt vời mà ngôn ngữ này mang lại.

2 Cài đặt và chạy chương trình đầu tiên

2.1 Cài đặt Golang

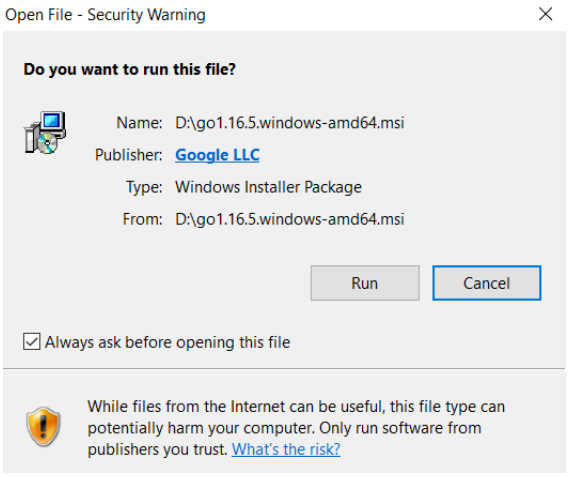
Để cài đặt Golang, ta chỉ cần thực hiện các bước sau:

- **Bước 1:** Mở trang web go.dev/dl và chọn phiên bản phù hợp với máy chủ.



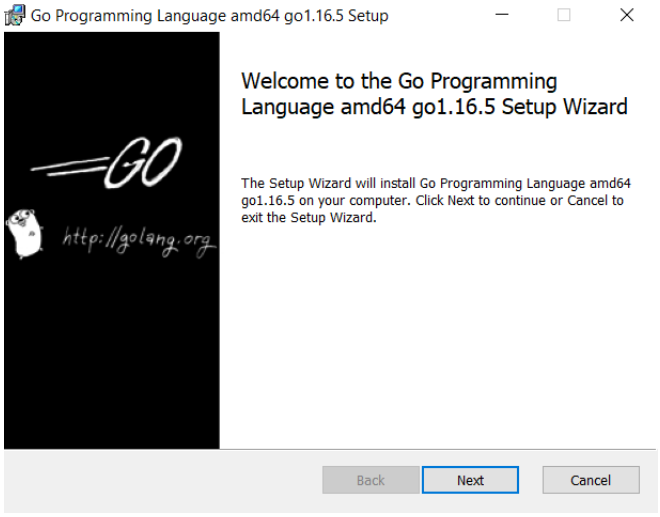
Hình 5: *Giao diện trang web để tải Golang*

- **Bước 2:** Sau khi tải về, cửa số install sẽ có dạng như hình dưới đây, ta nhấn **Run**.



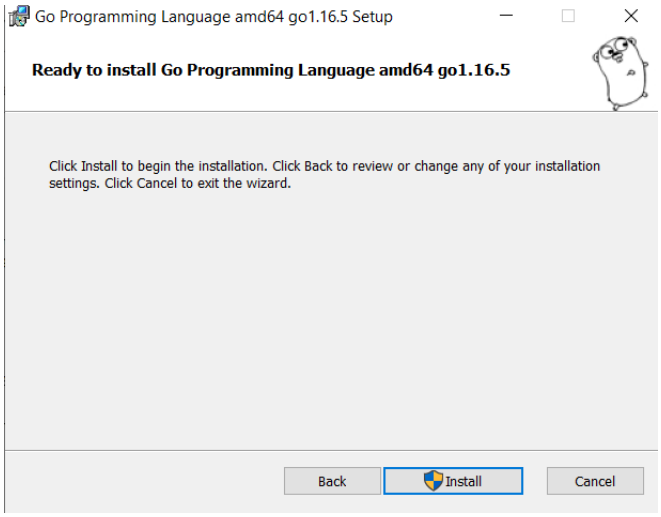
Hình 6: *Cửa số install của Golang*

- **Bước 3:** Sau khi thực hiện bước 2, các cửa sổ hiện lên liên tiếp, theo như mặc định, để nhanh chóng ta có thể chọn **next** option.



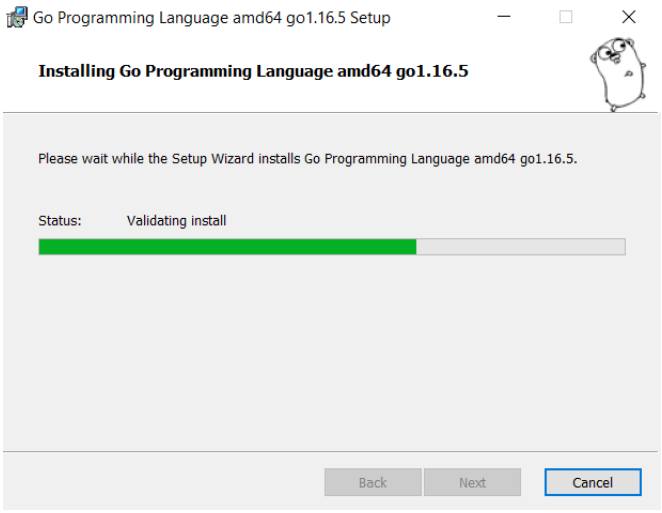
Hình 7: *Cửa số setup của Golang*

- **Bước 4:** Sau một lúc chọn next option, cửa sổ xác nhận cài đặt xuất hiện, ta chọn **Install** để cài đặt.



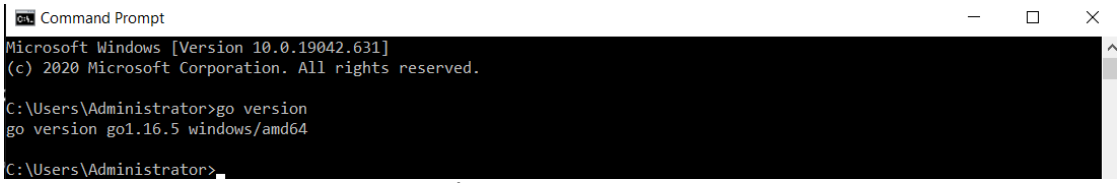
Hình 8: *Cửa số xác nhận cài đặt của Golang*

- **Bước 5:** Chờ quá trình cài đặt. Sau quá trình này, ta chọn **finish** để hoàn tất quá trình cài đặt Golang vào máy.



Hình 9: *Quá trình cài đặt Golang*

- **Bước 6:** Mở terminal và kiểm tra xem phiên bản Go trong máy, bằng cách sử dụng câu lệnh bên dưới.



Hình 10: *Kiểm tra phiên bản của Go trên máy*

Vậy là ta đã hoàn tất quá trình cài đặt trên máy Windows, với các hệ điều hành khác, ta cũng có thể làm một cách tương tự.

2.2 Chạy thử chương trình đầu tiên

Tập tin thuộc Go được tạo với phần extension .go. Chúng ta có thể chạy chương trình Go sử dụng cú pháp.

```
go run < filename >
```

Để tạo ra chương trình đầu tiên, ta phải tạo file, ví dụ **program.go** với nội dung bên dưới, sau đó lưu lại.

```
package main

import "fmt"

// this is a comment

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

Sau đó, ta mở terminal lên, di chuyển thư mục hiện tại về vị trí chứa file **program.go** và gõ câu lệnh sau để chạy chương trình trên.

```
go run program.go
```

Sau khi chạy chương trình, kết quả xuất ra terminal sẽ như sau.

```
Hello World
```

Vậy là ta đã chạy thử chương trình đầu tiên thành công.

2.3 Cấu trúc cơ bản của chương trình Golang

Sau khi chạy xong chương trình đầu tiên, ta có thể dễ dàng nhận ra cấu trúc của một chương trình Golang gồm 3 phần chính:

- **package**:
  Mọi chương trình Golang đều phải bắt đầu bằng việc khai báo **package**. Package được sử dụng để nhóm một hoặc nhiều tập tin có liên quan lại với nhau, hướng tới xử lí một tập hợp các vấn đề có liên quan đến thứ mà package đó hướng tới. Đây là cách tổ chức và sử dụng lại source code của Go. Packages giúp phân chia mã nguồn thành nhiều phần, do đó việc bảo trì ứng dụng cũng thuận tiện hơn.

```
package < package_name >
```

  Với tập tin dùng để chạy chương trình thì tên **package** được quy định cần phải đặt là **main** đồng thời chúng ta cũng cần định nghĩa một hàm với tên cũng là **main**. Trình biên dịch Go compiler sẽ sử dụng hàm main này để chạy chương trình đầu tiên.

```
package main

func main() {
    // Program starts in main first
    // Write code here
}
```

- **import**:
  Các package đôi khi cần phải import các package khác để tận dụng lại source code đã thiết kế sẵn. Để import thêm package khác, ta sử dụng câu lệnh sau.

```
import "fmt"
```

  Khi muốn import nhiều package cùng lúc, ta khai báo như sau.

```
import (
        "fmt",
        "math",
        "string",
)
```

  Lưu ý rằng các package khi import phải đặt trong cặp dấu " " mới hợp lệ.

- **func** (function hay hàm):
  Chức năng của hàm là nhóm một khối các câu lệnh lại với nhau, thường nhằm thực hiện một tác vụ nhất định. Ta có thể gọi khối các câu lệnh đó thực hiện nhiều lần chỉ bằng cách gọi tên hàm.

```
package main

import "fmt"

func myFunc(){
    fmt.Println("myFunc is executing")
    fmt.Println("Hello Golang")
}

func main(){
    fmt.Println("Hello World, I am Golang")
    myFunc()
```

```
13      myFunc ()
14  }
```

Chương trình sẽ in ra màn hình:

```
Hello  World , I  am  Golang
myFunc  is  executing
Hello  Golang
myFunc  is  executing
Hello  Golang
```

Ta có thể rút ra một số nhận xét về func như sau: Khi bắt đầu, mọi chương trình sẽ thực thi hàm **main()** đầu tiên. Việc viết hàm sẽ giúp tận dụng lại code, tiết kiệm thời gian khi lập trình các câu lệnh thực hiện một công việc lặp đi lặp lại nhiều lần. Đồng thời, hàm sẽ giúp cho người lập trình quản lý source code hiệu quả hơn.

3 Một số khái niệm cơ bản

3.1 Kiểu dữ liệu (Types)

Kiểu dữ liệu (Types) là một tập hợp các giá trị liên quan, mô tả các hoạt động có thể được thực hiện trên chúng và xác định cách chúng được lưu trữ trên bộ nhớ.

Go là **ngôn ngữ lập trình kiểu tĩnh** (statically typed programming language). Điều này có nghĩa là các biến luôn có một kiểu dữ liệu cụ thể, và nó không thể được thay đổi trong suốt chương trình. Mặc dù lập trình kiểu tĩnh có thể gây ra một số phiền phức và rườm rà, lập trình viên phải dành nhiều thời gian hơn để sửa lỗi chương trình. Tuy nhiên, nó giúp chúng ta suy luận về những gì chương trình của chúng ta đang làm và phát hiện ra nhiều lỗi phổ biến.

Trong Go có 3 kiểu dữ liệu chính như sau:

- **Numeric** (Kiểu số):
  - **Integer**:
    Đây là các giá trị số nguyên giống hoàn toàn như trong toán, tuy nhiên số nguyên trong máy tính có giới hạn.
    Các kiểu số nguyên trong Go là **uint8**, **uint16**, **uint32**, **uint64**, **int8**, **int16**, **int32**, **int64**, **int**.
    * uint8: 0 – 255
    * uint16: 0 – 65535
    * uint32: 0 – 4294967295
    * uint64: 0 – 18446744073709551615
    * int8: -128 – 127
    * int16: -32768 – 32767
    * int32: -2147483648 – 2147483647
    * int64: -9223372036854775808 – 9223372036854775807

    Ta có ví dụ về cách sử dụng kiểu số nguyên như sau:

```
1  package  main
2
3  import  "fmt"
4
5  func  main () {
6    var  num1  int16  =  20132
7    var  num2  int16  =  23244
8      fmt . Println ("Tong  2  so  num1  va  num2  la:",  num1  +  num2 )
9
10   var  num3  int32  =  20132
```

```
  var num4 int32 = 23244
    fmt.Println("Tong 2 so num3 va num4 la:", num3 + num4)

  var num5 int = 20132
  var num6 int = 23244
    fmt.Println("Tong 2 so num5 va num6 la:", num5 + num6)
}
```

Kết quả của chương trình trên là:

```
Tong 2 so num1 va num2 la: -22160
Tong 2 so num3 va num4 la: 43376
Tong 2 so num5 va num6 la: 43376
```

– **Floating Point**:
Đây là các giá trị số có phần thập phân, ví dụ 1.234, 123.4... Việc lưu trữ cũng như thao tác với số thực trong máy tính khá phức tạp. Ở đây chúng ta có một số lưu ý như sau:

* Số thực không bao giờ chính xác một cách tuyệt đối, rất khó để biểu diễn chính xác một số thực. Ví dụ như phép trừ 1.01 – 0.99 sẽ cho ra kết quả là 0.020000000000000018 chứ không phải là 0.02 như ta vẫn nghĩ.
* Cũng giống như số nguyên, số thực trong máy tính cũng có nhiều giới hạn khác nhau.

Trong Go có 2 kiểu số thực là **float32** và **float64**. Thông thường để biểu diễn số thực, ta chỉ cần dùng float64 là đủ.
Ví dụ về cách sử dụng kiểu số thực:

```
package main

import (
    "fmt"
)

func main() {
    a, b := 5.67, 8.97
    fmt.Printf("Kieu du lieu cua a la \%T va cua b la \%T\n", a, b)
    sum := a + b
    sub := a - b
    fmt.Println("Tong a va b:", sum)
    fmt.Println("Hieu a va b:", sub)
}
```

Kết quả của chương trình trên là:

```
Kieu du lieu cua a la float64 va cua b la float64
Tong a va b: 14.64
Hieu a va b:  -3.3000000000000007
```

– **Complex**:
Trong Go, có 2 kiểu số phức là **complex64** và **complex128**.

* **Complex64**: Số phức có phần thực float32 và phần ảo
* **Complex128**: Số phức có phần thực float64 và phần ảo

Số phức được tạo bằng một hàm Complex để xây dựng **phần thực** và **phần ảo**:

```
func complex(r, i FloatType) ComplexType
```

hoặc số phức cũng có thể được tạo bằng cú pháp viết tắt:

```
c := 6 + 7i
```

Phần thực và phần ảo như tham số và kết quả trả về là kiểu số phức. Khi khai báo số phức, chúng ta nên khai báo phần thực và phần ảo ở cùng kiểu dữ liệu. Ví dụ phần thực và phần

ảo đều là float32 thì kết quả trả về sẽ có kiểu complex64, phần thực và phần ảo đều là float64 thì kết quả trả về sẽ có kiểu complex128.
Ví dụ về cách sử dụng số phức:

```
package main

import (
  "fmt"
)

func main() {
  c1 := complex(5, 7)
  c2 := 8 + 27i
  cadd := c1 + c2
  fmt.Println("Tong 2 so phuc c1 va c2:", cadd)
  cmul := c1 * c2
  fmt.Println("Tich 2 so phuc c1 va c2:", cmul)
}
```

Kết quả của chương trình trên là:

```
Tong 2 so phuc c1 va c2: (13 + 34i)
Tich 2 so phuc c1 va c2: (-149 + 191i)
```

- **String** (Kiểu chuỗi):
Trong Go, chuỗi là một tập hợp các byte, bây giờ chúng ta giả sử chuỗi là một tập hợp các kí tự. Ví dụ:

```
package main

import (
    "fmt"
)

func main() {
    first := "Naveen"
    last := "Ramanathan"
    name := first +" "+ last
    fmt.Println("My name is",name)
}
```

Kết quả của chương trình trên là:

```
My name is Naveen Ramanathan
```

Trong chương trình trên, các biến **first**, **last** được gán lần lượt bằng các chuỗi "Naveen", "Ramanathan". Các chuỗi được nối với nhau bởi dấu cộng, khoảng trắng cũng là một chuỗi, biến name được gán bằng việc cộng các chuỗi vừa khai báo ở trên và chuỗi khoảng trắng. Có một số thao tác khác với chuỗi, chúng ta sẽ tìm hiểu trong một hướng dẫn khác.

- **Bool** (Kiểu luận lý):
Bool là kiểu dữ liệu chỉ nhận 2 giá trị hoặc true hoặc false (hoặc đúng hoặc sai). Ví dụ:

```
package main

import "fmt"

func main() {
  a := true //a duoc gan bang true
  b := false // b duoc gan bang false
  fmt.Println("a:", a, "b:", b)
  c := a && b // c duoc gan bang a&&b
  fmt.Println("c:", c)
  d := a || b // d duoc gan bang a||b
  fmt.Println("d:", d)
```

```
}
```

Kết quả của chương trình trên là:

```
a: true b: false
c: false
d: true
```

3.2 Biến (Variables)

Biến là một đại điểm lưu trữ, với kiểu dữ liệu cụ thể và tên liên quan. Biến có thể được khai báo bằng cách sử dụng cú pháp:

```
var < variable_name > <type>
```

Một khi đã khai báo biến và kiểu dữ liệu của nó, chúng ta có thể gán nó bất kì giá trị nào thuộc kiểu dữ liệu ấy. Chúng ta cũng có thể gán giá trị ban đầu cho một biết khi khai báo bằng cú pháp:

```
var < variable_name > <type> = <value>
```

Nếu ta khai báo biến với giá trị khởi tạo của nó, Go có thể chỉ ra kiểu giá trị của biến từ kiểu giá trị mà chúng ta gán cho biến. Vì vậy chúng ta không cần thiết phải khai báo kiểu dữ liệu theo dưới đây:

```
var < variable_name > = <value>
```

Chúng ta cũng có thể khai báo nhiều biến cùng một lúc bằng cú pháp

```
var < variable_name1 >, < variable_name2 >  = <value1>, <value2>
```

```
package main
import "fmt"

func main() {
    // declaring a integer variable x
    var x int
    x=3 // assigning x the value 3
    fmt.Println("x:", x) // prints 3

    // declaring a integer variable y with value 20 in a single statement and prints it
    var y int=20
    fmt.Println("y:", y)

    // declaring a variable z with value 50 and prints it
    //Here type int is not explicitly mentioned
    var z=50
    fmt.Println("z:", z)

    // Multiple variables are assigned in single line- i with an integer and j with a string
    var i, j = 100,"hello"
    fmt.Println("i and j:", i,j)
}
```

Kết quả của chương trình trên là:

```
x: 3
y: 20
z: 50
i and j: 100 hello
```

Go cũng cung cấp một cách dễ dàng hơn để khai báo biến với giá trị ban đầu bằng cách loại bỏ đi từ khó var sử dụng cú pháp

```
< variable_name > := < value >
```

Chú ý rằng khi sử dụng := thay vì =, chúng ta không thể sử dụng := để gán giá trị cho một biến đã được khai báo trước đó. := chỉ được dùng để khai báo và gán biến ngay lúc bán đầu.

```
package main
import ("fmt")

func main() {
  a := 20
  fmt.Println(a)

  //gives error since a is already declared
  a := 30
  fmt.Println(a)
}
```

Kết quả của chương trình trên là:

```
./assign.go:7:4: no new variables on left side of :=
```

Các biến được khởi tạo mà không có giá trị ban đầu sẽ mặc định có giá trị 0 cho các kiểu số, sai cho kiểu Boolean và chuỗi rỗng cho kiểu chuỗi.

3.3 Hằng (Constants)

Hằng (constants) khai báo một giá trị không thể thay đổi được trong suốt chương trình. Hằng trong lập trình ngôn ngữ Go được khai báo bằng cách sử dụng từ khóa "const".

Ví dụ về **const** trong Go:

```
package main
import ("fmt")

func main() {
  const b =10
  fmt.Println(b)
  b = 30
  fmt.Println(b)
}
```

Kết quả của chương trình trên là:

```
.constant.go:7:4: cannot assign to b
```

4 Vòng lặp (Loop)

Vòng lặp được sử dụng để chạy một đoạn code lặp đi lặp lại nhiều lần. Thông thường, các câu lệnh được xử lý tuần tự nối tiếp nhau. Một câu lệnh vòng lặp cho phép chúng ta thực thi một câu lệnh hoặc một nhóm các câu lệnh nhiều lần.

for là vòng lặp duy nhất được hỗ trợ bởi Go. Go không hỗ trợ vòng lặp while và do while thường hiện diện trong các ngôn ngữ khác như là C, C++.

Cú pháp của for trong Go như sau:

```
for initialisation_expression; evaluation_expression; iteration_expression{
  // statement
}
```

- **initialisation_expression** được thực thi đầu tiên và một lần duy nhất
- Tiếp theo điều kiện **evaluation_expression** sẽ được đánh giá và nếu nó đúng thì phần thân bên trong vòng for sẽ được thực thi
- Sau đó **iteration_expression** sẽ được thực thi và **evaluation_expression** sẽ được đánh giá lại. Quá trình trên diễn ra đến khi điều kiện **evaluation_expression** sai.

Ví dụ về cách sử dụng for:

```
package main

import (
    "fmt"
)

func main() {
    for i := 1; i <= 10; i++ {
        fmt.Printf(" %d",i)
    }
}
```

Kết quả của chương trình trên là:

```
1 2 3 4 5 6 7 8 9 10
```

Khi chạy vòng for, người lập trình muốn dừng lặp ngay lập tức chỉ cần sử dụng câu lệnh **break**:

```
package main

import (
    "fmt"
)

func main() {
    for i := 1; i <= 10; i++ {
        if i > 5 {
            break //loop is terminated if i > 5
        }
        fmt.Printf("%d ", i)
    }
    fmt.Printf("\nline after for loop")
}
```

Kết quả của chương trình trên là:

```
1 2 3 4 5
line after for loop
```

Vòng lặp lồng nhau là một vòng lặp **for** có một vòng lặp **for** khác bên trong nó:

```
package main

import (
    "fmt"
)

func main() {
    n := 5
    for i := 0; i < n; i++ {
        for j := 0; j <= i; j++ {
            fmt.Print("*")
        }
        fmt.Println()
    }
}
```

Kết quả của chương trình trên là:

```
*
**
***
****
*****
```

5 Cấu trúc rẽ nhánh

5.1 Câu lệnh if else

Cấu trúc câu điều kiện If else cũng khá giống với các ngôn ngữ lập trình khác. Với cú pháp

```
if  condition {
//  statements_1
} else {
//  statements_2
}
```

```
package main
import "fmt"

func main() {
    var x = 50
    if x < 10 {
        // Executes if x < 10
        fmt.Println("x is less than 10")
    }
}
```

Vì giá trị của x lớn hơn 10 nên các lệnh bên trong khối điều kiện if sẽ không được thực hiện. Tương tự ta có các ví dụ

```
package main
import "fmt"

func main() {
    var x = 50
    if x < 10 {
        // Executes if x is less than 10
        fmt.Println("x is less than 10")
    } else {
        // Executes if x >= 10
        fmt.Println("x is greater than or equals 10")
    }
}
```

Kết quả của chương trình trên là:

```
x is greater than or equals 10
```

Golang cũng cho phép các lệnh if-else lồng nhau (nested if-else)

```
package main
import "fmt"

func main() {
    var x = 100
    if x < 10 {
        // Executes if x is less than 10
        fmt.Println("x is less than 10")
    } else if x >= 10 && x <= 90 {
        // Executes if x >= 10 and x<=90
        fmt.Println("x is between 10 and 90")
    } else {
```

```
        // Executes if both above cases fail i.e x>90
        fmt.Println("x is greater than 90")
    }
}
```

Kết quả của chương trình trên là:

```
x is greater than 90
```

5.2 Switch

Switch là một câu lệnh điều kiện khác. Các câu lệnh Switch đánh giá một biểu thức và kết quả được so sánh với một tập hợp các giá trị có sẵn (cases). Khi tìm thấy một kết quả phù hợp, các câu lệnh ở trong case đó sẽ được thực thi. Nếu không tìm thấy kết quả phù hợp, sẽ không có gì được thực hiện. Chúng ta có thể thêm một trường hợp mặc định với các câu lệnh sẽ được thực thi nếu không tìm thấy kết quả nào phù hợp. Cú pháp của switch là

```
switch expression {
    case value_1 :
         statements_1
    case value_2 :
         statements_2
    case value_n :
         statements_n
    default :
         statements_default
    }
```

Ở đây giá trị của expression được so sánh với các giá trị trong mỗi case. Sau khi tìm thấy giá trị khớp, các câu lệnh liên quan đến case đó sẽ được thực thi. Nếu không tìm thấy kết quả phù hợp, các câu lệnh trong phần default sẽ được thực thi.

```
package main
import "fmt"

func main() {
    a,b := 2,1
    switch a+b {
    case 1:
        fmt.Println("Sum is 1")
    case 2:
        fmt.Println("Sum is 2")
    case 3:
        fmt.Println("Sum is 3")
    default :
        fmt.Println(" Printing default ")
    }
}
```

Kết quả của chương trình trên là:

```
Sum is 3
```

Chúng ta có thể có nhiều giá trị trong một case bằng cách tách chúng với nhau bằng dấu phẩy.

6 Hàm

Ta có thể hiểu Function là một khuôn chứa các dòng code thực hiện một công việc nhất định. Function tiếp nhận đầu vào, sau đó thực hiện một vài tính toán và trả về kết quả đầu ra.

Cú pháp khai báo hàm trong Golang là:

```
func function_name(parameter_1 type, parameter_n type) return_type {
```

```
//statements
}
```

Một chương trình cơ bản về function như sau:

```
package main

import (
    "fmt"
)

func calculateBill(price, no int) int {
    var totalPrice = price * no
    return totalPrice
}
func main() {
    price, no := 90, 6
    totalPrice := calculateBill(price, no)
    fmt.Println("Total price is", totalPrice)
}
```

Kết quả của chương trình trên như sau:

```
Total price is 540
```

Các parameter và kiểu dữ liệu trả về là tùy chọn. Và chúng ta cũng có thể trả về đa giá trị từ một hàm.

```
package main
import "fmt"

//calc is the function name which accepts two integers num1 and num2
//(int, int) says that the function returns two values, both of integer type.
func calc(num1 int, num2 int)(int, int) {
    sum := num1 + num2
    diff := num1 - num2
    return sum, diff
}

func main() {
    x,y := 15,10

    //calls the function calc with x and y an d gets sum, diff as output
    sum, diff := calc(x,y)
    fmt.Println("Sum",sum)
    fmt.Println("Diff",diff)
}
```

Kết quả của hàm trên là:

```
Sum 25
Diff 5
```

Go cho phép ta có **định dạng trống**. "_" được hiểu là định danh trống trong Go. Nó có thể sử dụng cho bất kì giá trị nào của bất kì loại nào. Chúng ta cùng tìm hiểu việc sử dụng định danh trống này.

Chương trình bên dưới chỉ có area được trả về từ Function rectProps

```
package main

import (
    "fmt"
)

func rectProps(length, width float64) (float64, float64) {
    var area = length * width
    var perimeter = (length + width) * 2
```

```
    return area, perimeter
}
func main() {
    area, _ := rectProps(10.8, 5.6) // perimeter is discarded
    fmt.Printf("Area %f ", area)
}
```

7 Cấu trúc dữ liệu cơ bản

7.1 Mảng

Mảng biểu diễn cho một dãy các phần tử cùng kiểu được đặt tên có một kích thước cố định. Chúng ta không thể có một mảng chứa cả số nguyên và ký tự trong đó. Chúng ta cũng không thể thay đổi kích thước của một mảng sau khi đã xác định size của nó.

Cú pháp khai báo là:

```
var arrayname [size] type
```

Mỗi phần tử của mảng có thể được gán giá trị sử dụng cú pháp

```
arrayname [index] = value
```

Index của mảng bắt đầu từ 0 cho tới size-1. Thêm nữa, chúng ta có thể không khai báo giá trị size khi khai báo mảng với giá trị thay thế size viết thành ... và sau đó trình biên dịch sẽ tìm kích thước của mảng bằng số giá trị gán vào. Độ dài của mảng được tìm bằng cú pháp

```
len( arrayname )
```

```
package main
import "fmt"

func main() {
    var numbers [3] string // Declaring a string array of size 3 and adding elements
    numbers [0] = "One"
    numbers [1] = "Two"
    numbers [2] = "Three"
    fmt.Println(numbers [1]) // prints Two
    fmt.Println(len(numbers)) // prints 3
    fmt.Println(numbers) // prints [One Two Three]

    directions := [...] int {1,2,3,4,5} // creating an integer array and the size of the array is defined by the number of elements
    fmt.Println(directions) // prints [1 2 3 4 5]
    fmt.Println(len(directions)) // prints 5

    // Executing the below commented statement prints invalid array index 5 (out of bounds for 5- element array)
    //fmt.Println(directions [5])
}
```

Kết quả của chương trình trên là:

```
Two
3
[One Two Three]
[1 2 3 4 5]
5
```

7.2 Slice

Một slice là một phần hoặc phân đoạn của một mảng. Hoặc có thể coi nó là một dạng view hoặc một phần view của một mảng cơ bản mà nó trỏ đến. Chúng ta có thể truy cập các phần tử của một slice bằng cách sử dụng tên và index của slice giống như chúng ta làm trong một mảng. Tuy chúng ta không thể thay đổi độ dài của một mảng, nhưng bạn có thể thay đổi kích thước của một slice.

Nội dung của một slice thực ra là các con trỏ đến các phần tử của một mảng. Có nghĩa là nếu chúng ta thay đổi bất kỳ phần tử nào trong một slice, thì nội dung mảng bên dưới cũng sẽ bị ảnh hưởng.

```
var slice_name [] type = array_name[start:end]
```

Cú pháp này sẽ tạo ra ra một slice tên slie_name từ một mảng tên array_name với các giá trị từ index start đến end-1

```
package main
import "fmt"

func main() {
    // declaring array
    a := [5] string {"one", "two", "three", "four", "five"}
    fmt.Println("Array after creation:",a)

    var b [] string = a[1:4] //created a slice named b
    fmt.Println("Slice after creation:",b)

    b[0]="changed" // changed the slice data
    fmt.Println("Slice after modifying:",b)
    fmt.Println("Array after slice modification:",a)
}
```

Kết quả của chương trình trên là:

```
Array after creation: [one two three four five]
Slice after creation: [two three four]
Slice after modifying: [changed three four]
Array after slice modification: [one changed three four five]
```

7.3 Structures

Structure là một kiểu dữ liệu do người dùng định nghĩa mà bản thân nó chứa thêm một phần tử cùng loại hoặc khác kiểu.

Cú pháp để khai báo một struct là

```
type structname struct {
   variable_1 variable_1_type
   variable_2 variable_2_type
   variable_n variable_n_type
}
```

```
package main
import "fmt"

//declared the structure named emp
type emp struct {
        name string
        address string
        age int
}

//function which accepts variable of emp type and prints name property
func display(e emp) {
          fmt.Println(e.name)
}
```

```

func main() {
// declares a variable, empdata1, of the type emp
var empdata1 emp
//assign values to members of empdata1
empdata1.name = "John"
empdata1.address = "Street-1, London"
empdata1.age = 30

//declares and assign values to variable empdata2 of type emp
empdata2 := emp{"Raj", "Building-1, Paris", 25}

//prints the member name of empdata1 and empdata2 using display function
display(empdata1)
display(empdata2)
}
```

Kết quả của chương trình trên là:

```
John
Raj
```

7.4 Methods

Cú pháp của một method là:

```
func (variable variabletype) methodName(parameter1 paramether1type) {
}
```

```
package main
import "fmt"

//declared the structure named emp
type emp struct {
    name string
    address string
    age int
}

//Declaring a function with receiver of the type emp
func(e emp) display() {
    fmt.Println(e.name)
}

func main() {
    //declaring a variable of type emp
    var empdata1 emp

    //Assign values to members
    empdata1.name = "John"
    empdata1.address = "Street-1, Lodon"
    empdata1.age = 30

    //declaring a variable of type emp and assign values to members
    empdata2 := emp {
        "Raj", "Building-1, Paris", 25}

    //Invoking the method using the receiver of the type emp
   // syntax is variable.methodname()
    empdata1.display()
    empdata2.display()
}
```

Go không phải là một ngôn ngữ hướng đối tượng và nó không có khái niệm về lớp. Các method cung cấp cho chúng ta cảm giác về những gì chúng ta làm trong các chương trình hướng đối tượng, nơi các hàm của một class được gọi bằng cú pháp objectname. functionname()

8 Packages

Các packages được sử dụng để tổ chức code. Trong một dự án lớn, việc viết mã trong một tệp duy nhất là không khả thi. Ngôn ngữ lập trình Go cho phép chúng ta tổ chức code theo các package khác nhau. Điều này làm tăng khả năng đọc code và khả năng tái sử dụng. Một chương trình Go có thể thực thi phải chứa một gói có tên là main và việc thực thi chương trình bắt đầu từ hàm có tên là main. Chúng ta có thể thêm package vào chương trình với cú pháp

```
import package_name
```

Step 1) Tạo file tên package_example.go và thêm vào đoạn code dưới đây

```
package main
import "fmt"
//the package to be created
import "calculation"

func main() {
  x,y := 15,10
  //the package will have function Do_add()
sum := calculation.Do_add(x,y)
fmt.Println("Sum",sum)
}
```

Ở chương trình phía trên ta thấy fmt là một package mà Go cung cấp cho chúng ta chủ yếu cho hoạt động I/O. Ngoài ra chúng ta cũng có thể thấy package tên calculation.

Step 2) Đầu tiên chúng ta phải tạo package calculation bên trong tập tin cùng tên nằm bên trong tập tin scr của go. Đối với Window chúng ta có thể tìm được path bằng cách type lênh echo %GOROOT% trong terminal. Và đây là tập tin chúng ta được chuyển đến C:/Go/

Step 3) Chuyển đến tập tin scr. Vì Go yêu cầu tên của package và tên của tập tin ở thư mục scr phải giống nhau, chúng ta tạo một thư mục tên calculation.

Step 4) Tạo một tệp tin tên calc.go (Tên có thể tùy chọn, tuy nhiên tên package trong code phải giống với tên package chúng ta cần. Ở đây nó nên là calculation) trong thư mục calculation và thêm vào đoạn code bên dưới.

```
package calculation

func Do_add(num1 int, num2 int)(int) {
    sum := num1 + num2
    return sum
}
```

Step 5) Chạy lệnh go install từ thư mục calculation

Step 6) Giờ quay trở lại package_example.go và chạy lệnh go run package_examle.go. Kết quả sẽ là 25.

Lưu ý rằng tên của hàm Do_add bắt đầu bằng một chữ cái viết hoa. Điều này là do trong Go nếu tên hàm bắt đầu bằng chữ in hoa thì có nghĩa là các chương trình khác có thể nhìn thấy (truy cập) nó thì các chương trình khác không thể truy cập nó. Nếu tên hàm là do_add, thì chúng ta sẽ gặp lỗi

```
cannot refer to unexported name calculation.calc..
```

9 Defer và stacking defers

Các câu lệnh defer được sử dụng để trì hoãn việc thực hiện một lệnh gọi hàm cho đến khi hàm chứa câu lệnh defer hoàn thành việc thực thi.

```
package main
import "fmt"

func sample() {
```

```
    fmt.Println("Inside the sample()")
}
func main() {
    //sample() will be invoked only after executing the statements of main()
    defer sample()
    fmt.Println("Inside the main()")
}
```

Kết quả của chương trình trên là:

```
Inside the main()
Inside the sample()
```

Stacking defer là sử dụng nhiều câu lệnh defer. Giả sử chúng ta có nhiều câu lệnh defer bên trong một hàm. Go đặt tất cả các lệnh gọi hàm defer trong một stack và khi hàm bao quanh thực thi xong, các hàm trong stack được thực thi theo thứ tự **Last In First Out** (LIFO).

```
package main
import "fmt"

func display(a int) {
    fmt.Println(a)
}
func main() {
    defer display(1)
    defer display(2)
    defer display(3)
    fmt.Println(4)
}
```

Kết quả của chương trình trên là:

```
4
3
2
1
```

10 Pointers

Trước khi tìm hiểu con trỏ, trước tiên chúng ta sẽ thảo luận về toán tử '&'. Toán tử '&' được sử dụng để lấy địa chỉ của một biến. Nó có nghĩa là '& a' sẽ in địa chỉ bộ nhớ của biến a.

```
package main
import "fmt"

func main() {
  a := 20
  fmt.Println("Address:",&a)
  fmt.Println("Value:",a)
}
```

Kết quả của chương trình trên là:

```
Address: 0xc000078008
Value: 20
```

Một biến con trỏ lưu trữ địa chỉ bộ nhớ của một biến khác. Bạn có thể khai báo một con trỏ bằng cú pháp

```
var variable_name *type
```

```
package main
import "fmt"
```

```
3
4  func main() {
5    //Create an integer variable a with value 20
6    a := 20
7
8    //Create a pointer variable b and assigned the address of a
9    var b *int = &a
10
11   //print address of a(&a) and value of a
12   fmt.Println("Address of a:",&a)
13   fmt.Println("Value of a:",a)
14
15   //print b which contains the memory address of a i.e. &a
16   fmt.Println("Address of pointer b:",b)
17
18   //*b prints the value in memory address which b contains i.e. the value of a
19   fmt.Println("Value of pointer b",*b)
20
21   //increment the value of variable a using the variable b
22   *b = *b+1
23
24   //prints the new value using a and *b
25   fmt.Println("Value of pointer b",*b)
26   fmt.Println("Value of a:",a)}
```

Kết quả của chương trình trên là:

```
Address of a: 0x416020
Value of a: 20
Address of pointer b: 0x416020
Value of pointer b 20
Value of pointer b 21
Value of a: 21
```

11 Concurrency

Go hỗ trợ thực thi đồng thời các tác vụ. Sự đồng thời hóa đạt được trong Go bằng cách sử dụng Goroutines và Channels.

11.1 Goroutines

Goroutine là một hàm có thể chạy đồng thời với các hàm khác. Thông thường, khi một hàm được gọi, điều khiển sẽ được chuyển vào hàm được gọi và khi điều khiển hoàn thành việc thực thi hàm được gọi của nó, nó sẽ trở lại hàm đang gọi. Sau đó, hàm gọi tiếp tục thực thi. Hàm gọi đợi hàm được gọi hoàn tất việc thực thi trước khi nó tiếp tục với phần còn lại của các câu lệnh.

Nhưng trong trường hợp của goroutin thì khác, hàm gọi sẽ không đợi quá trình thực thi của hàm được gọi hoàn tất. Nó sẽ tiếp tục thực thi với các câu lệnh tiếp theo. Bạn có thể có nhiều go routines trong một chương trình.

Ngoài ra, chương trình main cũng sẽ thoát khi nó hoàn thành việc thực thi các câu lệnh của nó và nó sẽ không đợi hoàn thành các goroutines được gọi.

Goroutine được gọi bằng cách sử dụng từ khóa go, theo sau là một lệnh gọi hàm.

Ví dụ

```
go add(x,y)
```

```
package main
import "fmt"

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

```
9
10 func main() {
11   //invoking the goroutine display()
12   go display()
13   //The main() continues without waiting for display()
14   for i:=0; i<5; i++ {
15     fmt.Println("In main")
16   }
17 }
```

Kết quả của chương trình trên là:

```
1 In main
2 In main
3 In main
4 In main
5 In main
```

Ở đây, chương trình main đã hoàn thành việc thực thi ngay cả trước khi chương trình goroutine bắt đầu. display () là một goroutine được gọi bằng cú pháp

```
1 go function_name(parameter list)
```

Bây giờ chúng ta sửa đổi chương trình để in các câu lệnh từ display (). Chúng ta thêm thời gian trễ 2 giây trong vòng lặp for của main() và thời gian trễ 1 giây trong vòng lặp for của display().

```
1  package main
2  import "fmt"
3  import "time"
4
5  func display() {
6    for i:=0; i<5; i++ {
7      time.Sleep(1 * time.Second)
8      fmt.Println("In display")
9    }
10 }
11
12 func main() {
13   //invoking the goroutine display()
14   go display()
15   for i:=0; i<5; i++ {
16     time.Sleep(2 * time.Second)
17     fmt.Println("In main")
18   }
19 }
```

Kết quả của chương trình trên là:

```
1  In display
2  In main
3  In display
4  In display
5  In main
6  In display
7  In display
8  In main
9  In main
10 In main
```

11.2 Channels

Channel là một cách để các hàm giao tiếp với nhau. Nó có thể được coi là một phương tiện nơi một routine đặt dữ liệu và sau đó được truy cập bởi một routine khác trong máy chủ Golang.

Một kênh có thể được khai báo bằng cú pháp.

```
channel_variable := make(chan datatype)
```

Ví dụ

```
ch := make(chan int)
```

Chúng ta có thể gửi dữ liệu đến một routine bằng cú pháp

```
channel_variable <- variable_name
```

Ví dụ

```
ch <- x
```

Chúng ta có thể nhận dữ liệu từ một kênh bằng cú pháp

```
variable_name := <- channel_variable
```

Ví dụ

```
y := <- ch
```

Trong các ví dụ ngôn ngữ Go ở trên về goroutine, chúng ta đã thấy chương trình main không đợi goroutine. Nhưng đó không phải là trường hợp khi các channel có mặt. Giả sử nếu một goroutine đẩy dữ liệu đến channel, hàm main () sẽ đợi ở câu lệnh nhận dữ liệu trên channel cho đến khi nó nhận được dữ liệu.

```
package main
import "fmt"
import "time"

func display() {
  time.Sleep(5 * time.Second)
  fmt.Println("Inside display()")
}

func main() {
  go display()
  fmt.Println("Inside main()")
}
```

Kết quả của chương trình trên là:

```
Inside main()
```

Giờ chỉnh sửa code chúng ta có

```
package main
import "fmt"
import "time"

func display(ch chan int) {
  time.Sleep(5 * time.Second)
  fmt.Println("Inside display()")
  ch <- 1234
}

func main() {
  ch := make(chan int)
  go display(ch)
  x := <-ch
  fmt.Println("Inside main()")
  fmt.Println("Printing x in main() after taking from channel:",x)
}
```

Kết quả của chương trình trên là:

```
Inside display()
Inside main()
Printing x in main() after taking from channel: 1234
```

Sender đẩy dữ liệu đến kênh có thể thông báo cho receiver rằng không có thêm dữ liệu nào được thêm vào channel bằng cách đóng channel. Điều này chủ yếu được sử dụng khi chúng ta sử dụng một vòng lặp để đẩy dữ liệu đến một channel. Có thể đóng channel bằng cách sử dụng

```
close(channel_name)
```

Và ở phía receiver, có thể kiểm tra xem channel có bị đóng hay không bằng cách sử dụng một biến bổ sung trong khi tìm nạp dữ liệu từ channel bằng cách sử dụng

```
variable_name, status := <- channel_variable
```

Nếu status là True có nghĩa là bạn đã nhận dữ liệu từ kênh. Nếu là false, có nghĩa là bạn đang cố gắng đọc từ một channel đã đóng.

12 Select

Select có thể được xem như một lệnh switch hoạt động trên các channel. Ở đây các case sẽ là một hoạt động của channel. Thông thường, mỗi câu lệnh case sẽ được đọc thử từ kênh. Khi bất kỳ case nào sẵn sàng (channel được đọc), thì câu lệnh liên quan đến trường hợp đó được thực thi. Nếu nhiều case cùng sẵn sàng, nó sẽ chọn một case ngẫu nhiên. Chúng ta cũng có thể có một case mặc định được thực thi nếu không có case nào sẵn sàng.

Hãy nhìn đoạn chương trình dưới đây

```
package main
import "fmt"
import "time"

//push data to channel with a 4 second delay
func data1(ch chan string) {
    time.Sleep(4 * time.Second)
    ch <- "from data1()"
}

//push data to channel with a 2 second delay
func data2(ch chan string) {
    time.Sleep(2 * time.Second)
    ch <- "from data2()"
}

func main() {
    //creating channel variables for transporting string values
    chan1 := make(chan string)
    chan2 := make(chan string)

    //invoking the subroutines with channel variables
    go data1(chan1)
    go data2(chan2)

    //Both case statements wait for data in the chan1 or chan2.
    //chan2 gets data first since the delay is only 2 sec in data2().
    //So the second case will execute and exits the select block
    select {
    case x := <-chan1:
        fmt.Println(x)
    case y := <-chan2:
        fmt.Println(y)
    }
}
```

Kết quả của chương trình trên là:

```
from data2()
```

```
package main
import "fmt"
import "time"

//push data to channel with a 4 second delay
func data1(ch chan string) {
    time.Sleep(4 * time.Second)
    ch <- "from data1()"
}

//push data to channel with a 2 second delay
func data2(ch chan string) {
    time.Sleep(2 * time.Second)
    ch <- "from data2()"
}

func main() {
    //creating channel variables for transporting string values
    chan1 := make(chan string)
    chan2 := make(chan string)

    //invoking the subroutines with channel variables
    go data1(chan1)
    go data2(chan2)

    //Both case statements check for data in chan1 or chan2.
    //But data is not available (both routines have a delay of 2 and 4 sec)
    //So the default block will be executed without waiting for data in channels.
    select {
    case x := <-chan1:
        fmt.Println(x)
    case y := <-chan2:
        fmt.Println(y)
    default:
      fmt.Println("Default case executed")
    }
}
```

Kết quả của chương trình trên là:

```
Default case executed
```

13 Mutex

Mutex là hình thức ngắn gọn cho mutual exclusion. Mutex được sử dụng khi chúng ta không muốn cho phép một tài nguyên được truy cập bởi nhiều subroutines cùng một lúc. Mutex có 2 methods Lock và Unlock. Mutex được chứa trong package sync. Các câu lệnh phải được thực thi mutually exclusively có thể được đặt bên trong mutex.Lock () và mutex.Unlock ().

```
package main
import "fmt"
import "time"
import "strconv"
import "math/rand"
//declare count variable, which is accessed by all the routine instances
var count = 0

//copies count to temp, do some processing(increment) and store back to count
//random delay is added between reading and writing of count variable
func process(n int) {
  //loop incrementing the count by 10
  for i := 0; i < 10; i++ {
```

```
    time.Sleep(time.Duration(rand.Int31n(2)) * time.Second)
    temp := count
    temp++
    time.Sleep(time.Duration(rand.Int31n(2)) * time.Second)
    count = temp
  }
  fmt.Println("Count after i="+strconv.Itoa(n)+" Count:", strconv.Itoa(count))
}

func main() {
  //loop calling the process() 3 times
  for i := 1; i < 4; i++ {
    go process(i)
  }

  //delay to wait for the routines to complete
  time.Sleep(25 * time.Second)
  fmt.Println("Final Count:", count)
}
```

Kết quả của chương trình trên là:

```
Count after i=1 Count: 11
Count after i=3 Count: 12
Count after i=2 Count: 13
Final Count: 13
```

Khi chúng ta chạy lại với mutex

```
package main
import "fmt"
import "time"
import "sync"
import "strconv"
import "math/rand"

//declare a mutex instance
var mu sync.Mutex

//declare count variable, which is accessed by all the routine instances
var count = 0

//copies count to temp, do some processing(increment) and store back to count
//random delay is added between reading and writing of count variable
func process(n int) {
  //loop incrementing the count by 10
  for i := 0; i < 10; i++ {
    time.Sleep(time.Duration(rand.Int31n(2)) * time.Second)
    //lock starts here
    mu.Lock()
    temp := count
    temp++
    time.Sleep(time.Duration(rand.Int31n(2)) * time.Second)
    count = temp
    //lock ends here
    mu.Unlock()
  }
  fmt.Println("Count after i="+strconv.Itoa(n)+" Count:", strconv.Itoa(count))
}

func main() {
  //loop calling the process() 3 times
  for i := 1; i < 4; i++ {
    go process(i)
  }

  //delay to wait for the routines to complete
  time.Sleep(25 * time.Second)
  fmt.Println("Final Count:", count)
}
```

Và giờ kết quả của chương trình sẽ là:

```
Count after i=3 Count: 21
Count after i=2 Count: 28
Count after i=1 Count: 30
Final Count: 30
```

14 Tổng kết

14.1 Ưu điểm của Go

Từ những phần trên, ta có thể tổng kết lại một số ưu điểm của Go như sau:

- Đây là ngôn ngữ mã nguồn mở, nó miễn phí cho mọi người, kể cả sử dụng với mục đích thương mại.
- Ngôn ngữ dễ học, dễ nhớ, dễ tiếp cận cho người mới tiếp cận, tìm hiểu.
- Điểm nổi bật của ngôn ngữ này là nó chạy với tốc độ đáng kinh ngạc để biên dịch mã máy.
- Sự đơn giản của Go mang lại khó thể thay thế bởi ngôn ngữ khác.
- Go cung cấp những tính năng ngắn gọn, thực tế và nhưng dịch vụ hiệu quả.
- Những hàm trong Go có tính linh hoạt và mang tính đồng bộ cao hơn hẳn những ngôn ngữ lập trình khác.
- Tích hợp mạnh mẽ với ngôn ngữ C++ tạo nên sức mạnh vượt trội.

14.2 Nhược điểm của Go

Tuy nhiên, ngôn ngữ này cũng có những ngược điểm nhất định:

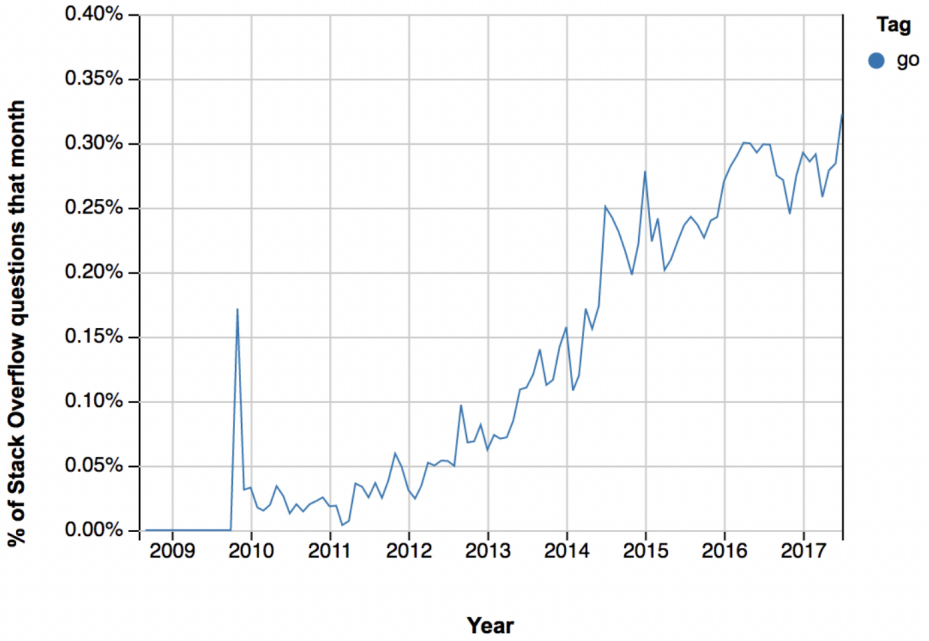
- Đây là ngôn ngữ vẫn còn mới, vẫn đang được phát triển.
- Không có sự quản lý bộ nhớ thủ công nên dễ gây ra một số lỗi về bộ nhớ trong lúc thực thi chương trình.
- Phát triển web với Go sẽ ưu tiên sự đơn giản hơn là hiệu suất. Do đó tạo một trang web với Go rất nhanh, tuy nhiên hiệu suất không cao bằng những ngôn ngữ đặc thù về web.
- Tính bất biến của Go là chưa đủ cho các tác vụ cần thiết.
- Mặc dù Go cung cấp nhiều tính năng khác biệt và quan trọng, thế nhưng Go không thể xem như một ngôn ngữ hệ thống.
- Xử lý lỗi chưa thật sự hoàn hảo.
- Quy tắc nghiêm ngặt. Đôi khi là một lợi thế, đôi khi là một bất lợi. Ví dụ: Có thể cảm thấy hơi nặng nề khi bạn có các cấu trúc luôn thay đổi. Tuy nhiên, hầu hết các quy tắc này có thể được ghi đè bằng cách sử dụng các khả năng phản chiếu của Go.

14.3 Triển vọng của Go trong tương lai

Có ít nhất nửa triệu lập trình viên trong cộng đồng Go. Các công ty đáng chú ý nhất là Google, Docker, Dropbox, Heroku, Medium, Lyft, Uber và các công ty khác đang sử dụng ngôn ngữ Go để lập trình cho một số hệ thống của họ.

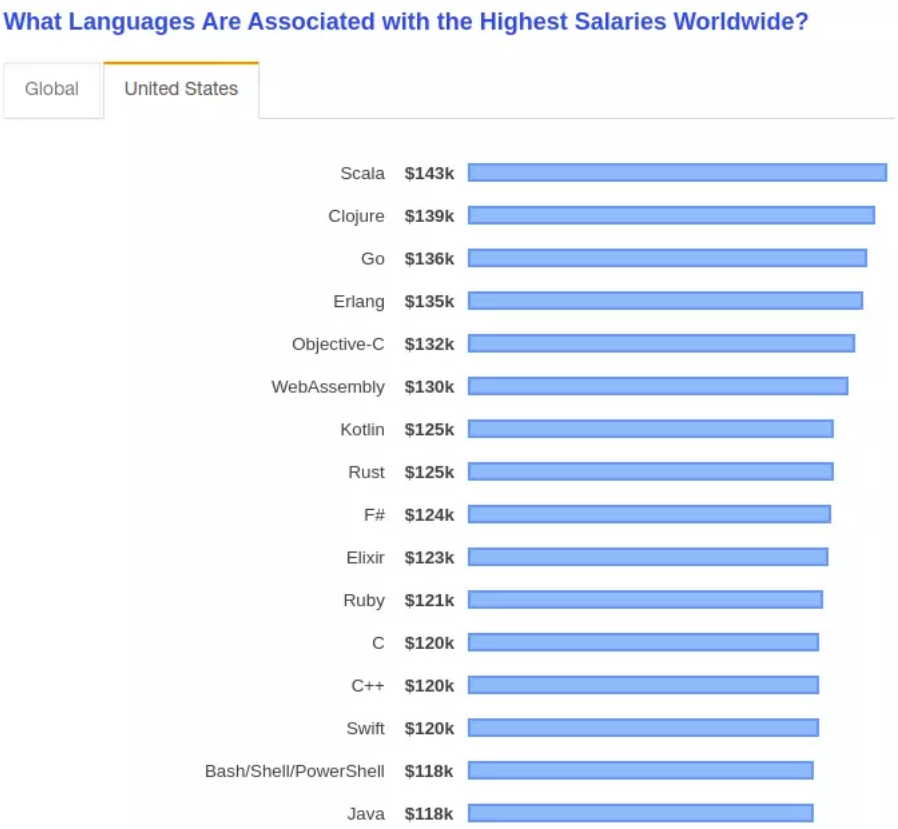
Trên các nền tảng hỏi đáp công nghệ nổi tiếng như Stackoverflow, tỷ trọng câu hỏi liên quan đến Go đang ngày được tăng lên.

Khi được các công ty công nghệ lớn yêu thích thì tương ứng mức lương của lập trình viên về Golang ngày càng tăng lên nhanh chóng.



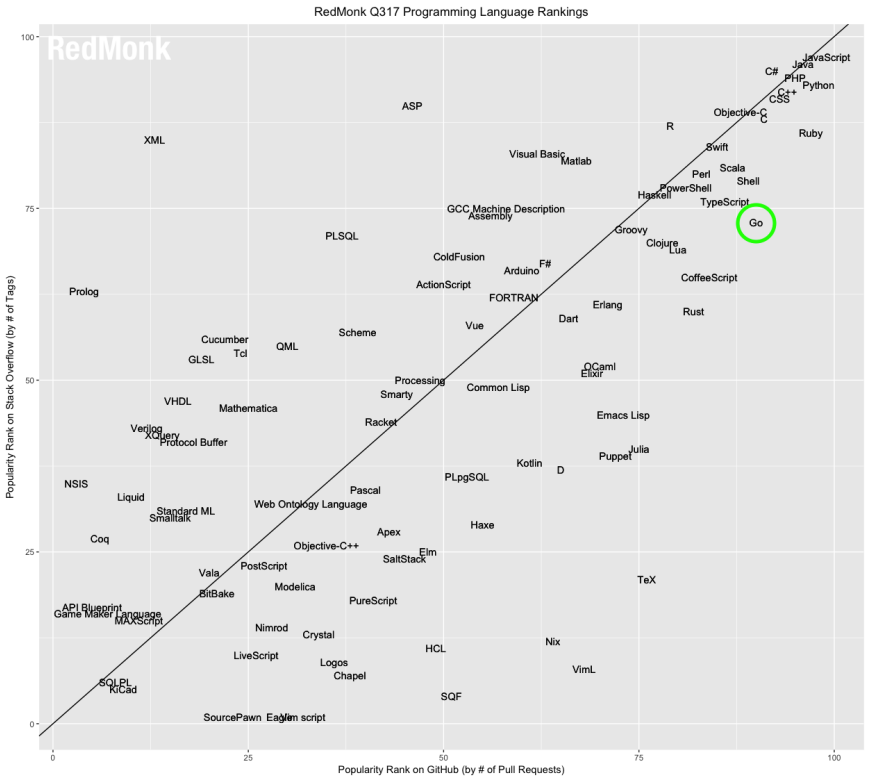
Hình 11: *Tỷ trọng số câu hỏi trên nền tảng Stackoverflow của Golang*

Theo nghiên cứu của HackerRank, Go có tương quan với mức tăng lương 33% so với mức trung bình. Không chỉ vậy, cứ 10 nhà quản lý tuyển dụng thì có 1 người mong muốn ứng viên có kinh nghiệm về Go. Các nhà phát triển có động lực tài chính để học nó.



Hình 12: *Mức lương của lập trình viên Golang đang ngày càng tăng cao*

Có thể nhận thấy Go đang dần trở thành một ứng viên sáng giá cho ngôn ngữ lập trình mới được sử dụng nhiều nhất. Mặc dù chỉ mới 15 tuổi, Golang đã ghi dấu ấn của mình. Và với sự thống trị của điện toán đám mây đang lan rộng, tương lai của Go đang trở nên tươi sáng hơn bao giờ hết.



Hình 13: *Độ phổ biến của Golang trên các nền tảng công nghệ nổi tiếng*

Tài liệu

[1] Tổng quan về Golang? Golang là gì? Tại sao nên dùng Golang? Trích nguồn: https://nordiccoder.com/blog/golang-la-gi/

[2] Go Vs Rust: Cuộc chiến ngôn ngữ lập trình. Trích nguồn: https://www.techgropse.com/blog/go-vs-rust/

[3] The Go Programming Language. Trích nguồn: https://go.dev/

[4] Golang là gì và tại sao bạn nên học Go? Trích nguồn: https://topdev.vn/blog/golang-la-gi-va-tai-sao-ban-nen-hoc-go/

[5] Go tutorial. Trích nguồn: https://www.tutorialspoint.com/go/index.htm

[6] Golang Tutorial: Learn Go Programming Language for Beginners. Trích nguồn: https://www.guru99.com/google-go-tutorial.html

[7] Golang cơ bản. Trích nguồn: https://www.codehub.com.vn/Golang-Co-Ban

[8] Golang tutorial series. Trích nguồn: https://golangbot.com/learn-golang-series/