Th.S LÊ NGỌC THẠCH

CHẠM TỚI GO TRONG 10 NGÀY

Mục lục

Đây là eBook của riêng bạn – đề nghị không chia sẻ cho ai khác nhé!

Lời nhắn

eBook "Chạm tới GO trong 10 ngày" này dự kiến phát hành vào tháng 3/2021. Bạn có thể đặt hàng ngay bây giờ với ưu đãi giảm 50% bằng 2 cách sau:

① Đăng ký khóa học ở đây https://thachln.github.io/courses/go1.

② Cài **App MinePI** cho điện thoại tại theo link:

https://minepi.com/thachln

Sử dụng invitation code: **thachln**

Thử dùng điện thoại để đào Pi Coin. Dự kiến eBook này được chia sẻ với giá **20** Pi Coin.

Phiên bản bạn đang nhận là bản nháp trong quá trình hoàn hiện.

Bạn được gởi riêng để tham khảo hoặc để góp ý. Vì thế bạn được toàn quyền sử dụng và **KHÔNG** chia sẻ với bất kỳ ai khác nhé, **KHÔNG** lưu trữ trên internet nói chung để hạn chế đến tay người không thật sự cần nó!

Về nội dung bạn thu lượm được từ eBook dưới dạng các bài tóm tắt, đánh giá, hoặc đề nghị bổ sung thì rất được **KHUYẾN KHÍCH** chia sẻ công khai.

Đặc biệt khuyến khích ban chia sẻ link:

https://thachln.github.io/courses/go1

Lê Ngọc Thạch

Quy ước

Một số nội dung trong tài liệu được trình bày với các định dạng khác nhau thì có ý nghĩa của nó, bạn đọc nên nắm thông tin này để tiện theo dõi.

Mã nguồn

Mã lệnh được viết và đóng khung với font chữ `Consolas`, có thanh màu vàng bên trái; và kết quả hiển thị trên màn hình được đóng trong khung màu đỏ bên dưới như sau:

```
package main

import (
    "fmt"
)

func main() {
    name := "Thạch"
    fmt.Println("Hello ",  name)
}
```

```
Hello Thạch
```

Lệnh thực thi trong hệ điều hành

Trường hợp các lệnh thực thi trong môi trường hệ điều hành (phân biệt với các lệnh, hoặc mã nguồn của chương trình thực thi trong môi trường lập trình) thì dấu hiệu có 2 thành màu vàng như sau:

```
Hello.exe "I can do"
```

Đường dẫn hiện hành

Đôi khi lệnh được hướng dẫn có cả tên ổ đĩa và thư mục và dấu mũi tên như bên dưới (phần chữ mờ). Phần này ý nói là chạy lệnh bên phải dấu mũi tên trong thư mục hiện hành D:\MyGo.

```
D:\MyGo> go build GoArgs.go
```

Cặp dấu nháy

Trong NNLT GO, dữ liệu **dạng kí tự** được bao đóng trong gặp **dấu nháy đơn**, dữ liệu **dạng chuỗi** được bao đóng trong **dấu nháy đôi**. Trên bàn phím máy tính thì dấu **nháy trái** và **phải** là giống nhau. Tuy nhiên trong phần mềm soạn thảo văn bản như Microsoft Word thì gặp dấu nháy đơn và đôi được thay thế bằng ‘’, “” để tăng tính thẩm mỹ. Các dấu nháy thẩm mỹ này khác với kí tự ' và " trên bàn phím (phím bên trái phím Enter).

Đôi khi bạn copy & paste mã nguồn vào các phần mềm như Microsoft Word thì các dấu nháy có thể bị “trang trí” lại như trên. Vì vậy khi copy mã ngồn từ Microsoft vào các công cụ lập trình thì hãy thay thế lại cho đúng.

Một qui ước khác liên quan đến dấu nháy đôi là khi dùng trong văn bản để bao đóng danh từ riêng, hoặc lệnh như hướng dẫn sau: *Bạn hãy thử gõ lệnh “dir” trong cửa của sổ TERMINAL để xem nội dung thư mục hiện hành*. Trong câu hướng dẫn này thì lệnh `dir` được gõ vào cửa sổ TERMINAL **KHÔNG** bao gồm cặp dấu nháy.

Cách viết trình tự bấm chọn menu

Khi cần trình bày thứ tự các nút bấm, hoặc các mục cần bấm trong các thao tác thì sẽ dùng dấu lớn hơn >. Ví dụ khi hướng dẫn bạn sử dụng phần mềm Visual Code vào menu Run, bấm vào mục “Run Without Debugging” thì sẽ viết gọn như sau:

Vào menu Run > Run Without Debugging.

Đường dẫn thư mục (Path)

Trong Windows thì dấu cách thư mục là dấu xuyệt trái (back slash). Ví dụ: D\ai2020\data.

Tuy nhiên ngôn ngữ GO và phần mềm lập trình Visual Code được thiết kế tương thích với các hệ điều hành khác như Macintosh, Linux. Các hệ điều hành thì thì dùng dấu xuyệt phải (right slash) để phân cách thư mục. Ví dụ: /mnt/d/MyGO.

Vì vậy khi trình bày đường dẫn thư mục trong câu văn thì đôi lúc dùng \, hoặc đôi lúc dùng / do dữ liệu được minh họa trên Windows hoặc Linux/Mac.

Nhưng trong mã nguồn thì đều thống nhất là dùng dấu xuyệt phải / như:

read.csv("D:/MyGO/HelloGO.go")

Các từ viết tắt, tiếng Anh thường xuyên được sử dụng trong sách

Viết tắt	Diễn giải
NNLT	Ngôn ngữ lập trình

Đây là eBook của riêng bạn – đề nghị không chia sẻ cho ai khác nhé!

Ngày 1: Giới thiệu

Đây là eBook của riêng bạn – đề nghị không chia sẻ cho ai khác nhé!

Bài 1 – Tại sao GO ra đời

Vào khoảng năm 2009, một nhóm chuyên gia của Google phát triển một dự án nội bộ tên là GO. GO được thiết kế để giúp các lập trình viên chuyên nghiệp tạo ra các phần mềm có tính ổn định, tin cậy và hiệu quả cao. Có thể xem GO là một hướng cải tiến của ngôn ngữ lập trình C cổ điển vốn rất mạnh nhưng kèm theo là rất phức tạp.

Bài 2: Ngôn ngữ lập trình GO

Trước khi đi vào ngôn ngữ lập trình, cụ thể là ngôn ngữ lập trình GOLANG (gọi ngắn gọn là GO) thì chúng nên biết vài khái niệm cơ bản về máy tính, về phần mềm. Đâu đó các khái niệm này có thể bạn đã học trong các lớp Tin học cơ bản, Nhập môn lập trình. Đây là cơ hội chúng ta ôn lại một chút.

Biến (Variable), Cấu trúc (Structure)

`Variable` là một cái tên dùng để chỉ một vùng nhớ trong máy tính. Để đơn giản, bạn hãy tưởng tượng cái máy vi tính giống như não người, trong đó có vùng nhớ (memory) để lưu thông tin tạm thời (lúc máy tính đang bật). Một `variable` được xem như một cái ô nhớ để đựng một giá trị nào đó.

Hình bên dưới là một thiết bị điện tử có trong máy tính của các bạn. Nó là một bản mạch gồm nhiều con chip có thể lưu trữ lại thông tin (bao gồm cả dữ liệu và lệnh) trong lúc máy tính có điện. Mọi người thường gọi ngắn gọn nó là thanh RAM.

Thanh RAM – nơi lưu "Trí nhớ" tạm thời của máy tính

Để các bạn hiểu hơn một chút về việc khai thác bộ nhớ của máy tính thì hãy tưởng tượng làm cách nào mà bạn bắt cái máy tính của bạn nhớ thông tin của một người bạn thân gồm các thông tin như sau:

Tên	Lê Ngọc Thạch
Chiều cao	165 cao
Cân nặng	70.5 kg
Giới tính	Nam
Ngày sinh	29/9/1977
Các chữ số yêu thích	1, 2, 5, 10, 20, 50, 100
Các môn thể thao yêu thích	Bóng bàn, bóng đá, Quần vợt

(Bạn có thể thay bằng thông tin của chính mình cho chính xác hơn nhé!)

Mỗi thông tin ở cột bên trái được gọi là một **biến** (variable). Bạn tưởng tượng là trong thanh RAM ở phần trước có rất nhiều ô nhỏ li ti. Mỗi ô nhỏ như vậy máy tính (*cụ*

thể các phần mềm mà chúng ta sẽ thực hành ở phần tiếp theo) được đặt cho một cái tên (name) – gọi là **tên biến** (variable name). Mỗi biến như vậy sẽ có một vùng nhớ khác nhau để chứa thông tin. Để đơn giản cho máy tính thì chúng ta nên sử dụng tên tiếng Anh để đặt cho tên biến.

Tên biến nên gồm các **kí tự chữ cái thường, chữ cái HOA**, **dấu gạch chân** (_) và có thể có kí số (ở giữa hoặc ở cuối tên biến). Để thống nhất cho các bạn khi thực hành thì tôi sử dụng quy trước theo thông lệ chung như sau:

- Tên biến bắt đầu bằng chữ thường.
- Kí tự Hoa và thường được hiểu là 2 kí tự khác nhau. *Ví dụ tên biến là fullName sẽ khác với tên biến là FullName. Tức là có hai vùng nhớ khác nhau để chứa thông tin của 2 biến này.*
- Tên biến phải ngắn gọn và gợi nghĩa.
- Khi tên biến gồm nhiều từ ghép lại (như Full name – 2 từ trong ví dụ trên) thì hãy viết Hoa kí tự của từ tiếp theo.

Để mô tả thông tin trong ví dụ trên thì chúng ta có thể tự quy định tên biến như bảng sau:

Thông tin	Tên biến
Tên	fullName
Chiều cao	height
Cân nặng	weight
Giới tính	sex
Ngày sinh	birthday
Các chữ số yêu thích	favorNumbers
Các môn thể thao yêu thích	favorSports

Trên đây là thông tin của một người, để mô tả thêm một người bạn nữa thì bạn phải làm sao?

Bạn có thể đặt thêm một loạt biến nữa như: fullName1, height1, …Tức là bạn thêm số thứ tự phía sau để có bộ biến mới cho người mới. Tuy nhiên cách này không hay. Giới khoa học máy tính đưa ra khái niệm **Structure** để giúp các bạn giải quyết nhu cầu này.

Structure (cấu trúc) là một khái niệm gom nhiều loại thông tin để mô tả một vật, một người hay nói chung là một đối tượng nào đó. Nói cụ thể hơn là Structure sẽ chứa trong nó nhiều biến. Chúng ta mô tả lại ví dụ trình bày thông tin cho người bạn "Thạch" của chúng ta ở trên dưới dạng một Structure như sau:

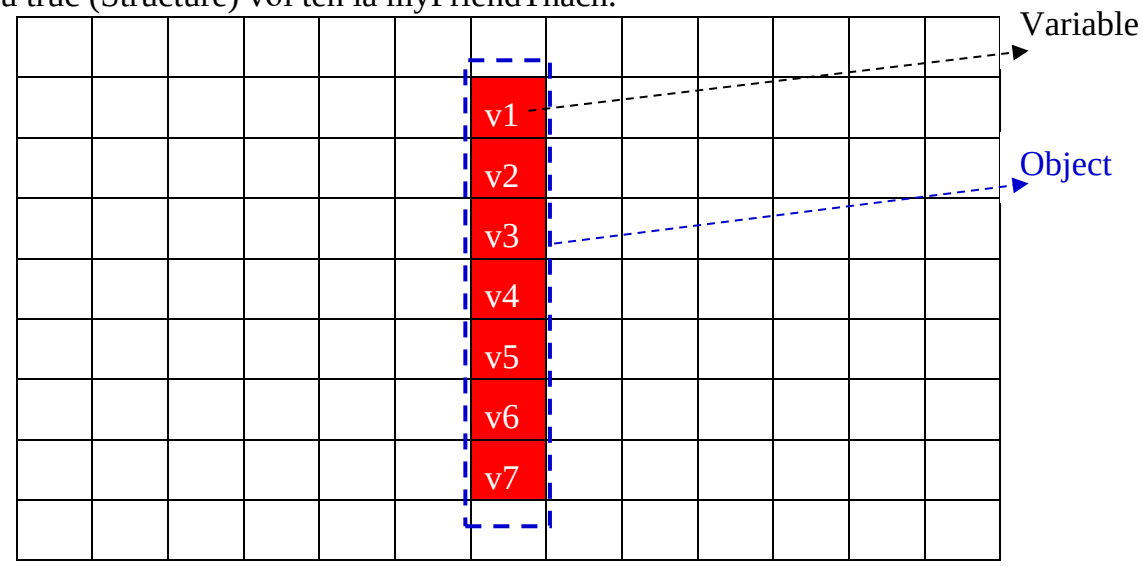
Structure: **myFriendThach**

fullName	Lê Ngọc Thạch
height	165 cao
weight	70.5 kg
sex	Nam
birthday	29/9/1977
favorNumbers	1, 2, 5, 10, 20, 50
favorSports	Bóng bàn, bóng đá, Quần vợt

Trong bảng trên xuất hiện từ **myFriendThach**, đây là một cái tên (name) được máy tính chỉ định (hoặc là **trỏ tới**) vùng nhớ của tất cả các thông tin về bạn Thạch.

Như vậy đến đây bạn biết được khái niệm biến (**variable**) là một cái tên (name) trỏ tới một vùng nhớ chứa thông tin cơ bản nào đó của bạn Thạch (như tên, cân nặng, v.v…). Toàn bộ các biến liên quan đến bạn Thạch được gom lại trong một vùng nhớ (đương nhiên là rộng hơn) gọi lại **Structure**.

Hình minh họa bên dưới gồm 7 ô nhớ tương ứng với 7 biến để mô tả thông tin về bạn Thạch (kí hiệu v1 đến v7 tương ứng với fullName…favorSports). Hình chữ nhật màu xanh được bao gởi đường đứt nét được gọi là một vùng nhớ cũng được đặt tên là một cấu trúc (Structure) với tên là myFriendThach.



Hình 1: Minh họa khái niệm biến (Variable) và cấu trúc (Structure)

Variable có nghĩa là gì?

🕮 Tra tự điển
Nếu tra tự điển Oxford thì variable có thể là danh từ, có thể là tính từ. ☞ Tính từ variable: *able to be changed or adapted* (có thể được thay đổi hoặc điều chỉnh)

> ☞ Danh từ variable: *an element, feature, or factor that is liable to vary or change* (một yếu tố, một nét đặc trưng, hoặc một nhân tố có khả năng **biến đổi** hoặc **thay đổi**).
>
> Cũng trong Oxford, variable được định nghĩa trong lĩnh vực Computing (điện toán) như sau: *a data item that may take on more than one value during the runtime of a program* (một phần tử dữ liệu có thể mang một hoặc hơn một giá trị trong suốt thời gian thực thi của chương trình).

Như vậy chữ variable có hai nghĩa mà các nhà khoa học máy tính và dịch giả Việt Nam đã dùng từ "biến" đã phản ánh đầy đủ rõ khái niệm "biến" trong máy tính.

Cụ thể là từ **vary** có hàm ý là có thể **biến đổi** thành đối tượng khác. Đối tượng khác ở đây có nghĩa là bản chất thông tin thay đổi hẳn. Chữ **change** có hàm ý là thay đổi giá trị của ô nhớ. Tức là bản chất, loại thông tin không thay đổi, mà chỉ thay đổi về nội dung, về giá trị của chúng.

Ví dụ:

Biến **height** đang có giá trị là 70.5 thì có thể được thay đổi thành một giá trị khác (tùy theo ngữ cảnh, thời gian như là đo lại tại một thời điểm khác) như là 71, 70 (chúng ta hiểu đơn vị là kg). Sự thay đổi này gọi là **change**.

Tuy nhiên, vì lý do nào đó trong ứng dụng phần mềm chúng ta muốn lưu trữ thông tin không phải là chiều cao nữa mà muốn lưu giá trị là một chức vụ cao nhất mà người đó đã từng làm. Tức là height sẽ được lưu giá trị là một **tên của chức vụ** (chứ không là một con số phản ánh chiều cao nữa). Lúc này biến height được biến đổi từ mục đích lưu con số phản ánh chiều cao thành một tên phản ánh chức vụ cao nhất. Cái này gọi là **vary** theo nghĩa trong tự điển Oxford.

Sau khi bạn hiểu được khái niệm Variable rồi thì câu hỏi tiếp theo là làm sao thiết lập giá trị cho biến. Cụ thể như thiết lập giá trị cho các ô nhớ từ v1 đến v2 trong hình 4.

Để làm được việc này thì bạn cần học thêm khái niệm gán (assign) trong phần tiếp theo.

Khai báo biến (variable declaration)

Trong ngôn ngữ lập trình GO, để khai báo một variable thì dùng cú pháp var như sau:

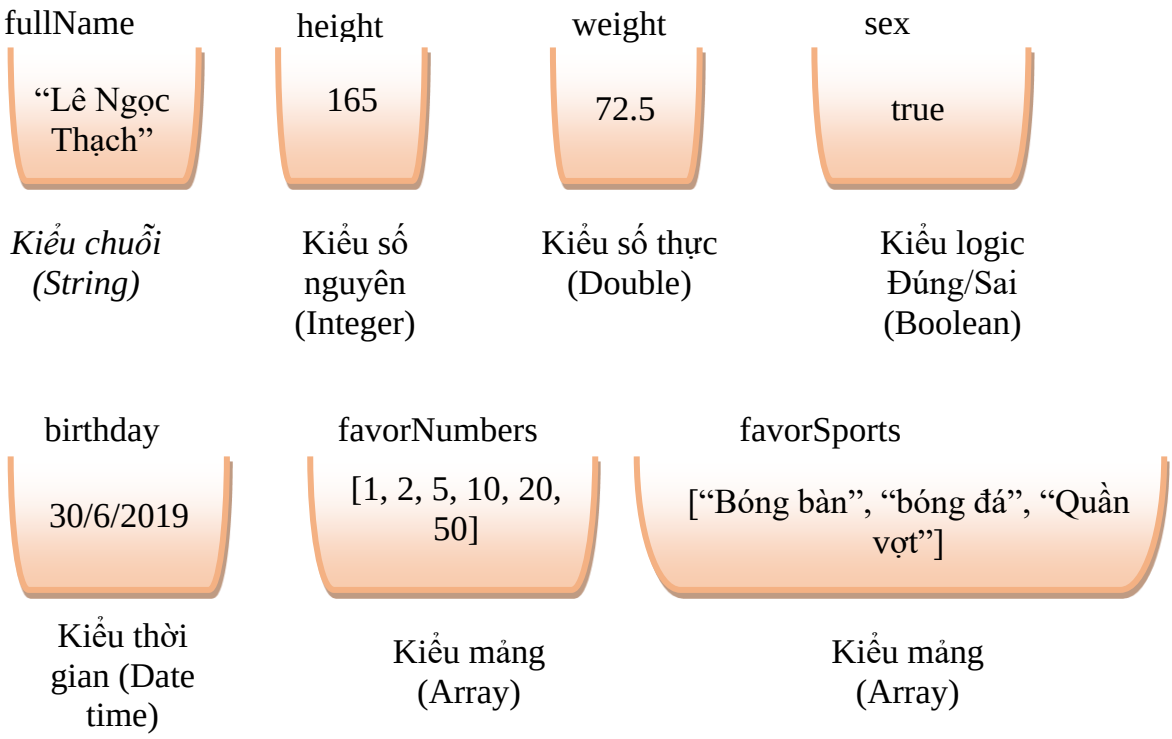
```
var <tên biến> <kiểu dữ liệu>
```

Ví dụ 2 dòng lệnh sau sẽ khai báo 2 vùng nhớ tương ứng cho fullName, height với kiểu dữ liệu tương ứng là `string` (chuỗi) và `int` (số nguyên)

```
var fullName string
var height int
```

Lệnh gán (assign)

Hình bên dưới minh họa các variable có tên level, score, name, birthday tương ứng với các ô nhớ (hãy xem như là một cái thùng) chứa bên trong nó các thông tin tương ứng.



Hình minh họa biến (variable)

Để thiết lập thông tin (hay còn gọi là dữ liệu) vào biến thì sử dụng phép gán (assign).

Cách 1 – Sử dụng cú pháp var và dấu =

Trong GO có thể vừa khai báo biến với từ khóa `var` và gán luôn giá trị cho biến với cú pháp là dấu = như sau:

```
var fullName string = "Lê Ngọc Thạch"
var height int = 165
```

Bạn cũng có thể không cần chỉ rõ kiểu giữ liệu, GO tự biết kiểu dữ liệu của biến với ví dụ sau:

```
    var fullName = "Lê Ngọc Thạch"
    var height = 165
```

Cách 2 – Sử dụng cú pháp :=

Trong GO, có thể dùng dấu bằng **:=** để thực hiện khai báo vùng nhớ và gán giá trị.

Ví dụ:

```
    fullName := "Lê Ngọc Thạch"
    height := 165
```

Khi khi đã khai báo biết thì GO dùng dấu bằng "=" thể thiết lập, hoặc thay đổi giá trị của biến.

Bài 3 – Chuẩn bị môi trường lập trình

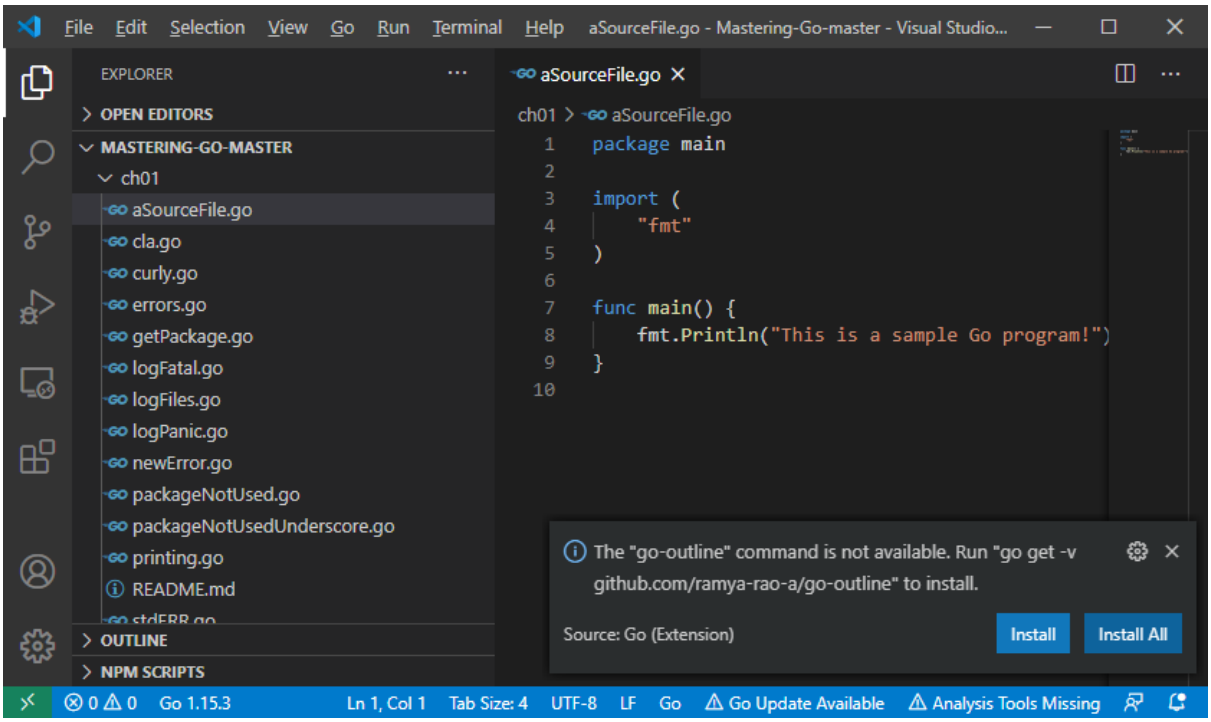
GO Core

Tải và cài gói phần mềm để biên dịch GO tại:

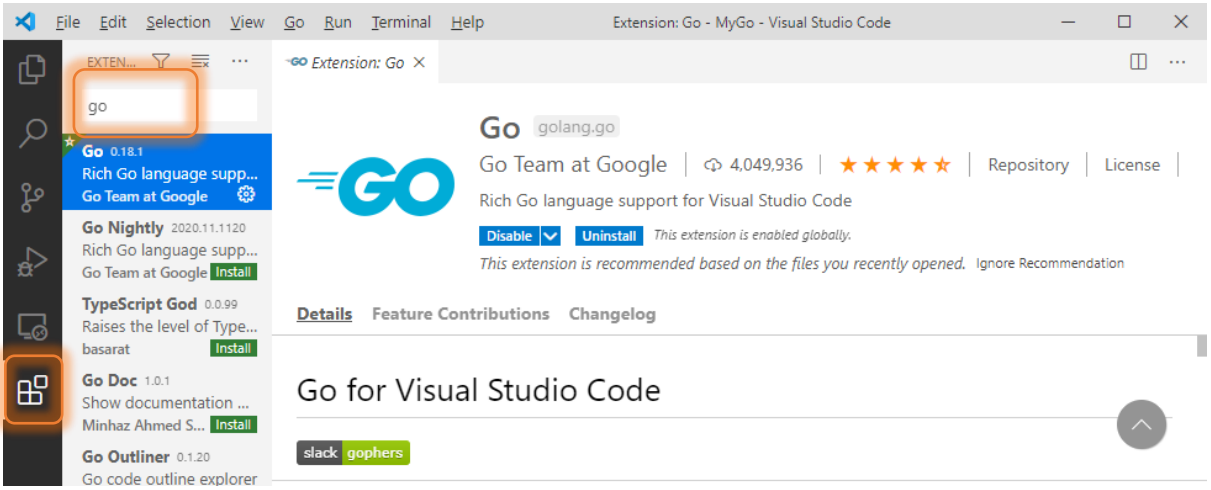
https://golang.org/dl/

Visual Code

Một trong các công cụ lập trình gọn nhẹ, phổ biến hiện tại là Visual Code. Visual Code có nhiều phần mở rộng giúp cho việc phát triển dự án bằng ngôn ngữ GO dễ dàng.



Cài extenstions



Phím tắt trong Visual Code

Hãy học thêm các phím tắt để sử dụng trong Visual Code tại link sau:

https://code.visualstudio.com/docs/languages/go

Bài 4 – Viết chương trình đơn giản với GO

Tạo thư mục D:\MyGo để chứa mã nguồn của các bài tập.

Khởi động Visual Code, nhấn tổ hợp phím Ctrl + K + O rồi chọn thư mục D:\MyGo.

Viết mã

Tạo file D:\MyGo\HelloGo.go với nội dung sau:

```
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Hello GO! Xin Chào GO nhé!")
}
```

Đoạn chương trình khai báo package là main ý muốn nói đoạn code phía sau được gọi để thực thi chương trình.

Đoạn chương trình trên sử dụng gói thư viện `fmt` bằng lệnh `import`.

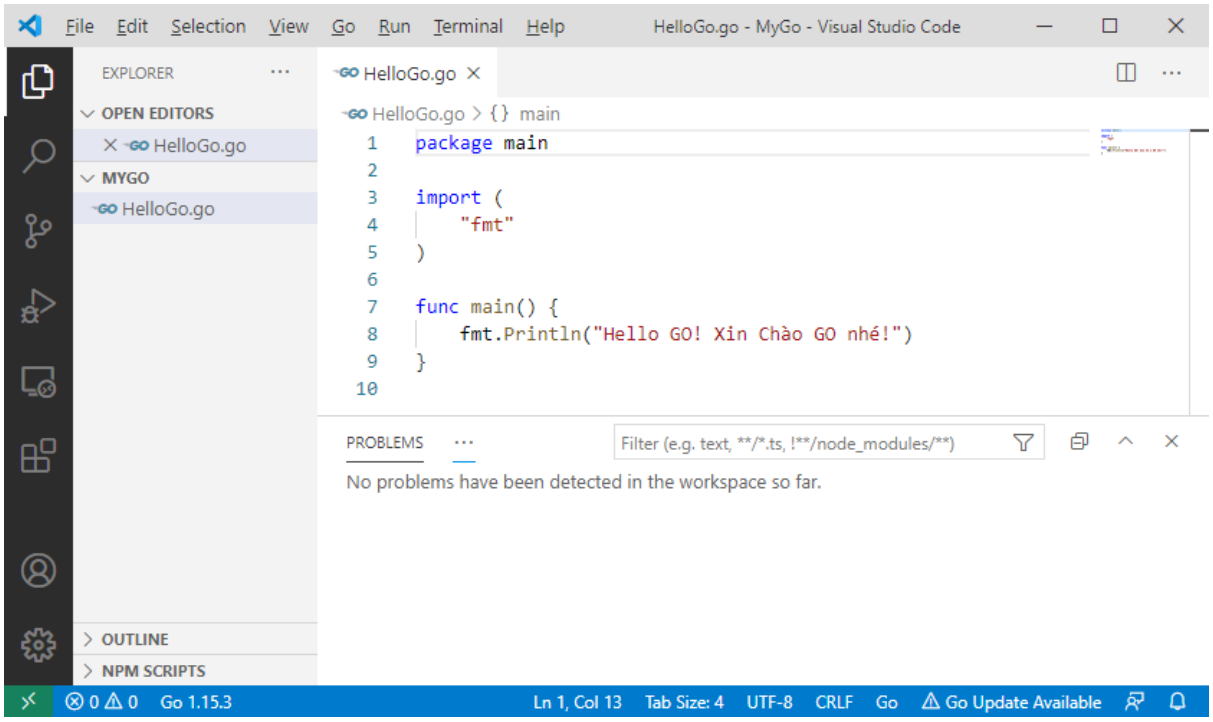
Khai báo hàm có tên là `main` là nơi bắt đầu của chương trình. Trong hàm `main` viết một lệnh đơn giản bằng cách gọi hàm Println trong gói thư viện `fmt` để hiển thị ra màn hình một câu chào (chuỗi đơn giản bao đóng bởi cặp dâu nháy đôi).

Biên dịch

Trong Visual Code nhấn phím Ctrl + Shift + ` (Thông thường phím ` là phím bên trái phím 1, phía trên phím Tab) để mở dấu nhắc lệnh.

Trường hợp thư mục hiện hành không phải là D:\MyGo thì bạn thực hiện hai lệnh sau:

```
D:
cd D:\MyGo
```



Lệnh "go build HelloGo.go" sẽ biên dịch file mã nguồn HelloGo.go thành file mã máy HelloGo.exe. Cách gõ nhanh như sau:

Bước 1: Gõ

go build H

Bước 2: Nhấn phím tab

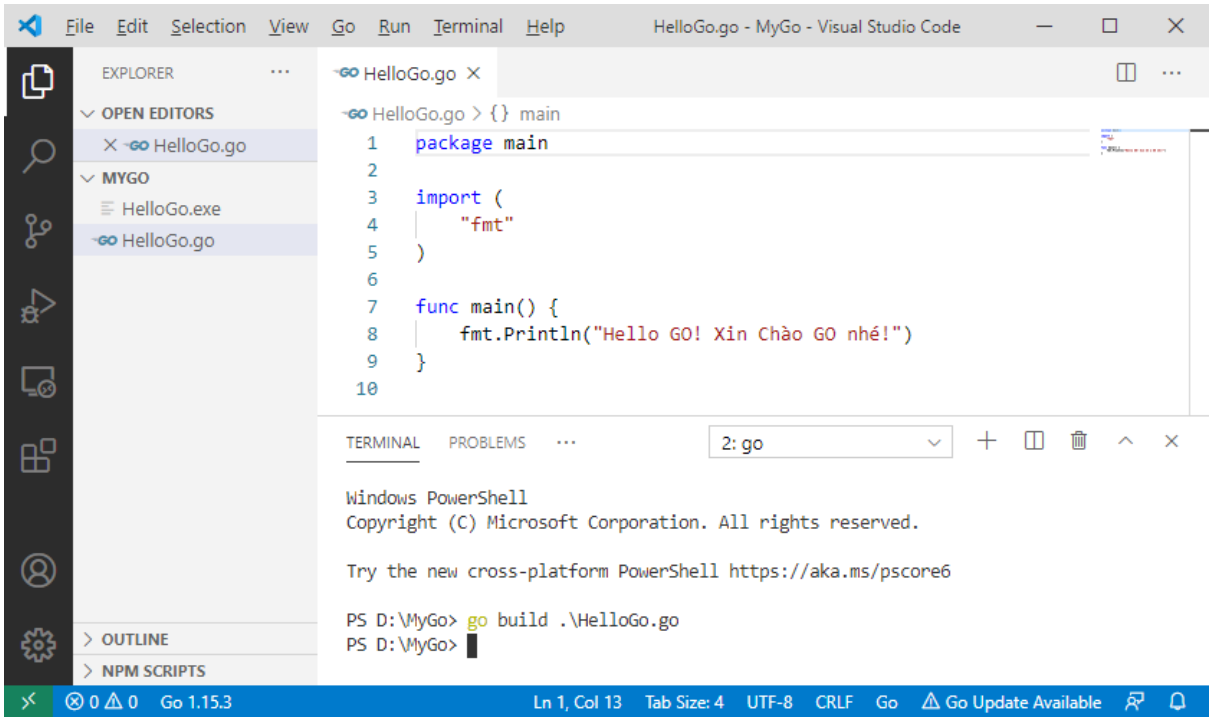
Cửa sổ lệnh sẽ tự động điền tiên file đầu đủ bắt đầu có chữ H. Trong trường hợp này là HelloGo.go. Kết quả lệnh đầy đủ là:

```
go build .\HelloGo.go
```

Kí hiệu dấu chấm có nghĩa là thư mục hiện hạnh. ".\HelloGo.go" có nghĩa là file HelloGo.go trong thư mục hiện hành.

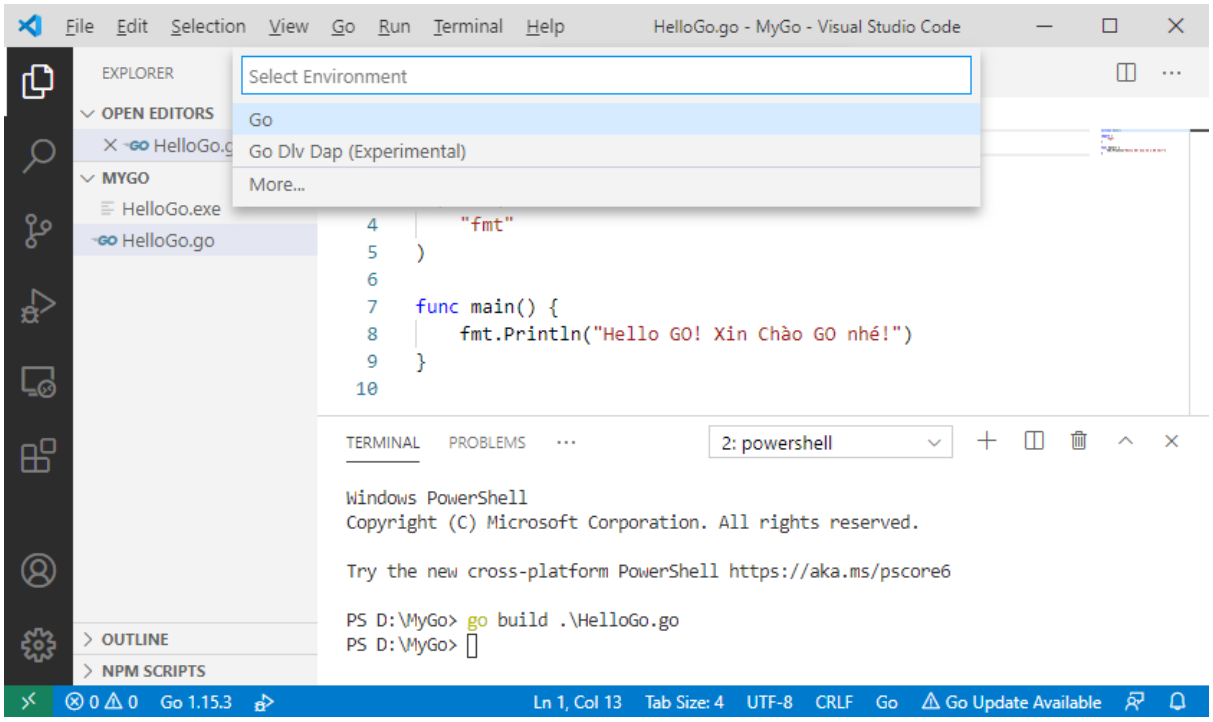
Tiếp theo bạn gõ lệnh HelloGo.exe để chạy thử. Cách gõ nhanh tương tư như sau:

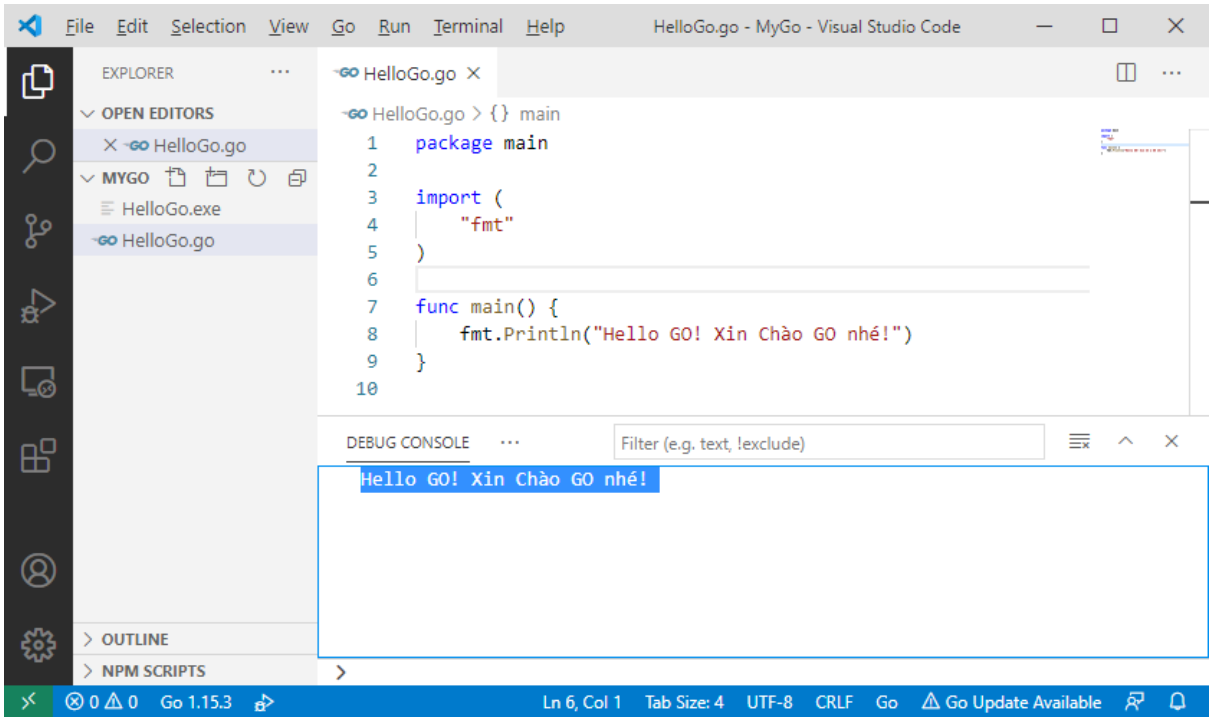
Bạn gõ chữ H rồi nhấn phím Tab, cửa sổ lệnh sẽ thông minh hiển thị sẵn cho mình lệnh .\HelloGo.exe. Xong nhấn Enter như hình bên dưới.



Chạy trực tiếp mã nguồn

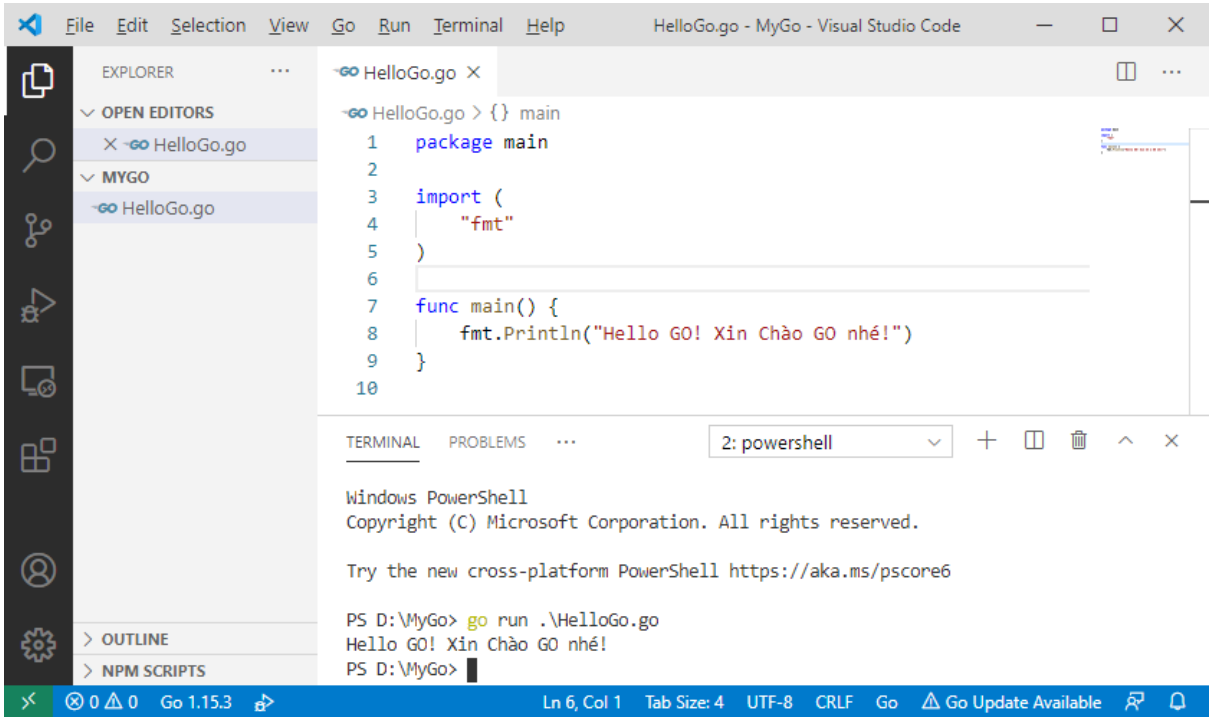
Để chạy chương trình mã không cần phải gõ lệnh biên dịch như ở trên thì bạn nhấn tổ hợp phím Ctrl + F5. Một hộp thoại nhỏ yêu cầu chọn môi trường (Select Environment), chọn mục Go. Sau đó xem kết quả trong cửa sổ TERMINAL như bên dưới:





Nếu bạn thích gõ lệnh thì mở cửa sổ lệnh bằng tổ hợp phím Ctrl + Shift + ` rồi gõ:

```
go run HelloGo.go
```



Phép gán (assign)

Cú pháp :=

Phần trước bạn đã biết cách viết một đoạn chương trình nhỏ để hiển thị ra màn hình một câu đơn giản. Thử cải tiến một chút để làm quen với khai báo biến `name` và phép gán với kí hiệu dấu bằng:

```
package main
```

```
import (
    "fmt"
)

func main() {
    name := "Thạch"
    fmt.Println("Hello ",  name)
}
```

Đoạn chương trình trên sử dụng cú pháp := để vừa khai báo biến vừa thiết lập giá trị cho nó. Tôi tạm gọi cú pháp := là **gán khai báo**.

Kết quả chương trình sẽ hiển thị ra chuỗi:

```
Hello Thạch
```

Cú pháp =

Chạy thử đoạn chương trình sau:

```
package main

import (
    "fmt"
)

func main() {
    name := "Thạch"
    fmt.Println("Hello ", name)

    name = "Lê Ngọc " + name
    fmt.Println("Họ và tên: ", name)
}
```

Bạn học thêm từ đoạn chương trình trên:

Sử dụng phép gán với cú pháp = để thay đổi giá trị của biến name bằng cách ghép nó với một chuỗi vào phía bên trái. Trong tài liệu này khi nói phép gán tức là dùng dấu =. Khi nói **phép gán khai báo** thì dùng hai chấm bằng : = nhé!

Các toán tử cơ bản

Các phép toán số học (Arithmetic Operator)

Phép toán	Ý nghĩa	Ví dụ
+		
-		
*		
/	Chia lấy phần nguyên	
%	Chi lấy phần dư	

++		
--		

Các toán tử so sánh (Relational Operator)

Phép toán	Ý nghĩa	Ví dụ
==		
!=		
>		
>=		
<		
<=		

Các toán tử logic (Logical Operators)

Phép toán	Ý nghĩa	Ví dụ
&&		
||		
!		

Hàm (function)

Khảo soát chương trình có hàm tính toán tuổi như sau:

```
package main

import (
    "fmt"
)

func calAge(birthYear int) int {
    return 2020 - birthYear
}

func main() {

    myAge := calAge(1977)

    fmt.Println(myAge)
}
```

Vài điểm chú ý về khai báo hàm:

- Từ khóa: func
- Sau từ khóa func là tên hàm. Ví dụ: calAge

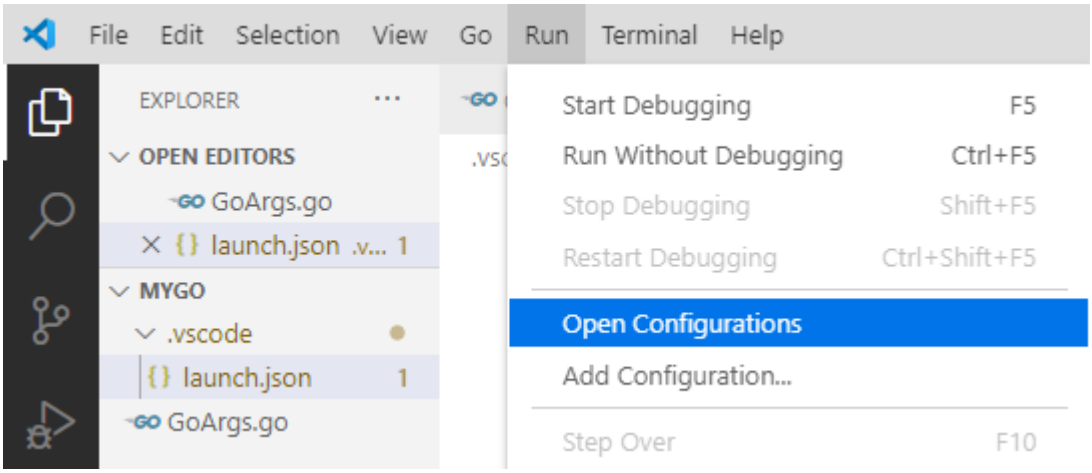
- Tiếp theo tên hàm là cặp dấu ngoặc, trong cặp dấu ngoặc là tham số. Kiểu dữ liệu của tham số được viết bên phải của tên tham số. Ví dụ: birthYear int

- Sau dấu ngoặc) kết thúc phần tham số là kiểu dữ liệu trả về của hàm. Trong ví dụ này là kiểu `int`.

- Nội dung của hàm được viết trong cặp dấu ngoặc nhọn {}

Chạy chương trình có tham số dòng lệnh trong Visual Code

Để chạy code GO trong Visual Code và truyền các tham số từ dòng lệnh thì bạn cần cấu hình một chút. Cụ thể là bạn vào menu Run > Open Configurations



Lần đầu tiên bạn vào menu này thì Visual Code sẽ cài đặt thêm một vài thứ. Bạn chỉ cần ngồi theo dõi là được.

Sau đó Visual Code sẽ tự tạo file launch.json trong thư mục làm việc của bạn. Bạn tìm đến dòng bên dưới để thêm các tham số:

```
"args": []
```

Bạn điền tham số vào giữa cặp dấu ngoặc, các tham số bao đóng bởi cặp dấu nháy đôi và cách nhau bởi dấu phẩy. Ví dụ

```
"args": ["Hải", "2000"]
```

Lưa file và quay lại file mã nguồn để chạy lại.

Lấy tham số từ dòng lệnh

Khảo sát mã nguồn của file D:\MyGo\GoArgs.go như sau:

```
package main

import (
    "fmt"
    "os"
)
```

```
func main() {
    fmt.Println("Number of arguments:", len(os.Args))

    fmt.Println("The first argument:", os.Args[0])
}
```

Biên dịch chương trình bằng lệnh

```
D:\MyGo> go build GoArgs.go
```

Chú ý phần bên trái "D:\MyGo>" ý nói đường dẫn thư mục hiện hành chứ không phải là nội dung của lệnh

Thực thi chương trình bằng cách gõ lệnh GoArgs.exe:

```
D:\MyGo> GoArgs.exe
```

```
Number of arguments: 1
The first argument: D:\MyGo\GoArgs.exe
PS D:\MyGo>
```

Kiến thức học được:

- Sử dụng thuộc tính `Args` trong thư viện `os` để lấy ra danh sách các tham số trên dòng lệnh.
- Phần tử đầu tiên của os.Args là đường dẫn của chương trình đang chạy.

Hãy thử chay luôn mã nguồn mà không cần biên dịch bằng lệnh:

```
go run .\GoArgs.go
```

Vòng lặp (loops)

Thử chạy đoạn code sau để khám phá cú pháp vòng lặp `for`. Ngoài ra học thêm cách sử dụng dấu phẩy để ngăn cách tham số trong lệnh `fmt.Println()`:

```
package main

import (
    "fmt"
)

func main() {
    for i := 0; i < 10; i++ {
        fmt.Println("i x 2 = ", i*2)
    }
}
```

Bài 5 – Biểu diễn thông tin đơn giản với GO

Câu hỏi chung cho những ai mới học lập trình là làm sao biểu diễn được các thông tin để máy tính hiểu được. Cụ thể trong ngữ cảnh eBook này là làm sao biểu diễn các thông tin cơ bản với ngôn ngữ GO.

Trong bài này chúng ta sẽ học các loại thông tin cơ bản, thường dùng sau:

- String
- Numeric
- Go arrays
- Go slices
- Go maps
- Go pointer
- Times & dates

Kiểu chuỗi (string)

Trong bài 4, bạn đã làm quen với một chương trình đơn giản là hiển thị một câu ra màn hình. Câu này được bao đóng trong gặp dấu nháy đôi như:

"Đây là một chuỗi các kí tự"

Lấy ra một kí tự của chuỗi

Khảo sát đoạn code sau:

```
package main

import (
    "fmt"
)

func main() {
    st := "I can do it"

    fmt.Println(st[0])
    fmt.Printf("%c", st[0])

}
```

```
73
I
```

Như vậy cú pháp st[0] sẽ lấy ra kí tự đầu tiên của chuỗi nhưng giá trị trả lại là một số nguyên. Giá trị này chính là giá trị mã kí tự.

Để hiển thị ra màn hình dạng kí tự của mã thì dùng hàm `fmt.Printf` với định dạng là `%c`.

Kiểu dữ liệu số (Numeric data types)

Số nguyên (Interger)

GO hỗ trợ 4 kiểu dữ liệu số nguyên không dấu tương ứng với số byte đi kèm như: `int8`, `int16`, `int32`, `int64`

Và 4 kiểu dữ liệu số nguyên có dấu: `uint8`, `uint16`, `uint32`, `uint64`. (uint là viết tắt của unsigned integer, số nguyên không dấu)

Thêm vào đó có 2 kiểu dữ liệu không ghi rõ số byte được sử dụng: `int` và `uint`. Kích thước (số byte) cho kiểu `int` và `uint` này tùy thuộc vào kiến trúc phần cứng của máy tính và hệ điều hành và phần mềm dùng để lập trình của bạn.

Số thực (floating-point numbers)

GO hỗ trợ 2 kiểu dữ liệu để biểu diễn số thực: `float32` và `float64`.

Viết chương trình Fibonacci

Đến đây bạn đã biết các kí hiệu để biểu diễn các thông tin cơ bản dạng số, và vòng lặp. Bây giờ hãy thực hành một chút bằng cách viết một chương trình hiển thị ra dãy Fibonacci.

Qui tắc của dãy số Finobaci đơn giản được áp dụng trong bài này như sau:

1 2 3 5 8 13 21 34 55 89 144…

Cho 2 số đầu tiên là 1 và 2. Số tiếp theo được tính bằng cách cộng 2 số liền kề phía trước.

Áp dụng các kiến thức đã học để viết chương trình Fibonacci bên dưới:

- Sử dụng cú pháp **gán khai báo** := để khai báo biến và gán dữ liệu luôn mà không cần nói rõ kiểu dữ liệu. Cụ thể là khai báo hai biến cho 2 số bên liền kề là n và m với n là 0; m là 1.
- Sử dụng vòng lặp `for` 10 bước với biến đếm là nCount
- Sử dụng lệnh Print trong thư viện `fmt` với 2 tham số: số fibonacci tiếp theo, và dấu cách.

```
package main

import (
    "fmt"
)

func main() {
```

```
    n := 0
    m := 1
    p := n + m

    for nCount := 0; nCount <= 10; nCount++ {

        fmt.Print(p, " ")
        n = m
        m = p
        p = n + m
    }
}
```

Kết quả:

```
1 2 3 5 8 13 21 34 55 89 144
```

Mảng (arrays)

Bạn đã làm quen với kiểu dữ liệu số, vòng lặp và viết được chương trình Fibonacci. Bây giờ mở rộng kiến thức với kiểu mảng nhé!

Mảng 1 chiều

Khám phá đoạn chương trình sau để biết cách khai báo mảng 1 chiều, ghi rõ số phần tử là 4 và liệt kê giá trị 4 phần từ là 1, 2, 3, 4.

```
package main

import "fmt"

func main() {
    arrayA := [4]int{1, 2, 3, 4}

    fmt.Println(arrayA, " len =", len(arrayA))
}
```

Kết quả:

```
[1 2 3 4] len = 4
```

Bạn tự đoán ý nghĩa cảu hàm `len`(array) nhé, `len` là viết tắt của length (độ dài).

Duyệt mảng 1 chiều bằng cú pháp `range`

```
package main

import "fmt"

func main() {
    arrayX := [5]int{1, 2, 3, 4, 5}
    for _, number := range arrayX {
        fmt.Print(number, " ")
    }
```

```
}
```

Lấy một phần của mảng tại từ một vị trí

Khảo sát đoạn code sau để biết cách dùng cú pháp [i:] và [i:j] của mảng

```
package main

import "fmt"

func main() {
    arrayX := [5]int{1, 2, 3, 4, 5}

    fmt.Println(arrayX[2:])
    fmt.Println(arrayX[2:4])
}
```

```
[3 4 5]
[3 4]
```

Kinh nghiệm học được:

- a[i:] sẽ trả lại mảng con tính từ phần tử tại vị trí i. Trong ví dụ trên phần tử có vị trí 2 (chú ý vị trí bắt đầu từ 0) là 3.
- a[i:j]: sẽ trả lại mảng con tính từ phần tử tại vị trí j cho đến phần tử ở vị trí **trước** j. Chú ý trước j tức là không bao gồm phần tử tại vị trí j.

Mảng 2 chiều (2 dimensions-array)

Khám phá đoạn code sau để hình dung cách khai báo và thiết lập mảng 2 chiều, hay còn gọi là ma trận (matrix). Code minh họa là ma trận gồm 3 dòng và 2 cột.

```
package main

import "fmt"

func main() {
    twoD := [3][2]int{
        {1, 2},
        {3, 4},
        {5, 6}
    }

    fmt.Println(twoD, " len =", len(twoD))
}
```

Slice (chưa biết gọi tiếng Việt là gì)

Ý tưởng chính của slides là:

- Được sử dụng như là array nhưng không cố định độ dài
- Kích thước của slice được mở rộng tự động
- Khi slice được sử dụng như là tham số của một hàm thì nó được truyền kiểu tham chiếu (passed by reference). Tức là các thay đổi slide bên trong hàm thì sau khi kết thúc hàm thì giá thị slide được thay đổi theo. Nếu bạn chưa quen khái niệm hàm, tham số dạng tham chiếu thì không sao, tạm thời chưa quan tâm đến nó nhé!

Quan sát đoạn code sau để thấy sự khác biệt khi khai báo và thiết lập giá trị giữa slice và array.

```
package main

import "fmt"

func main() {

    arrayA := [4]int{1, 2, 3, 4}
    fmt.Println(arrayA, " len =", len(arrayA))

    sliceA := []int{5, 6, 7, 8}
    fmt.Println(sliceA, " len =", len(sliceA))
}
```

Tạo slide với hàm make

Đoạn code sau sẽ tạo slice gồm 5 phần tử, mỗi phần tử mặc định có giá trị là 0.

```
package main

import "fmt"

func main() {

    intSlice := make([]int, 5)

    fmt.Println("Số phần tử của slice ", len(intSlice))
}
```

Duyệt các phần tử của slice hoặc array

```
package main

import "fmt"

func main() {

    intSlice := make([]int, 5)
```

```
    fmt.Println("Số phần tử của slice ", len(intSlice))

    for i := 0; i < len(intSlice); i++ {
        fmt.Println(intSlice[i])
    }
}
```

Thêm phần tử vào slice

```
package main

import "fmt"

func main() {

    intSlice := []int{1, 2, 3, 4, 5}

    intSlice = append(intSlice, 6)

    fmt.Println("intSlice = ", intSlice)
}
```

Truy xuất các phần từ của slice

Để truy cập 1 phần tử của slide hoặc array thì sử dụng cú pháp `[i]` với i là số thứ tự của phần tử, bắt đầu từ 0.

Khảo sát đoạn code sau để khám phá cú pháp `[i: j]`:

```
package main

import "fmt"

func main() {

    intSlice := []int{5, 6, 7, 8, 9, 10}

    fmt.Println("Lấy các phần tử từ vị trí 1 đến 3 = ", intSlice[1:3])
    fmt.Println("Lấy các phần tử từ vị trí 0 đến 3 = ", intSlice[0:3])
}
```

Dung lượng và Kích thước của slice

Slice có 2 thuộc tính quan trọng là capacity và length.

Hàm `len(slice)` cho biết số phần tử thật đang có của slice.

Hàm cap(slice) sẽ cho biết khả năng lưu trữ của slice. Bạn hình dung là GO chuẩn bị sẵn bộ nhớ để có thể lưu trữ các phần tử mới.

Hãy chạy và quan sát kết quả đoạn code sau để khám phá kết quả của hàm cap(slice) sau khi được thêm 1 phần tử nhé!

```
package main

import "fmt"

func main() {
    aSlice := []int{1, 2, 3}
    fmt.Println("aSlice: ", aSlice, "; len= ", len(aSlice), " ;cap=", c
ap(aSlice))

    // Thêm 1 phần tử
    aSlice = append(aSlice, 4)
    fmt.Println("aSlice: ", aSlice, "; len= ", len(aSlice), " ;cap=", c
ap(aSlice))
}
```

Kết quả:

```
aSlice: [1 2 3] ; len= 3 ;cap= 3
aSlice: [1 2 3 4] ; len= 4 ;cap= 6
```

Bạn có rút ra được điều gì không?

Slice 2 chiều

Tương tự như mảng 2 chiều thì slide 2 chiều được minh họa trong ví dụ sau:

```
package main

import "fmt"

func main() {
    aSlice := [][]int{
        {1, 2, 3},
        {4, 5},
    }
    fmt.Println("aSlice: ", aSlice, "; len= ", len(aSlice))
}
```

```
aSlice: [[1 2 3] [4 5]] ; len= 2
```

Chú ý kích thước của dòng 1 và dòng 2 là khác nhau. Bạn tự rút ra nhận xét nhé.

Thử so sách kế quả với đoạn code sau minh họa mảng 2 chiều gồm 2 dòng và 3 cột như sau:

```
package main

import "fmt"

func main() {
    aSlice := [2][3]int{
        {1, 2, 3},
        {4, 5},
    }
```

```
    fmt.Println("aSlice: ", aSlice, "; len= ", len(aSlice))
}
```

```
aSlice: [[1 2 3] [4 5 0]] ; len= 2
```

Tạm dừng việc làm quen kiểu dữ liệu `slice` ở đây. Còn nhiều điều thú vị về slide sẽ được giải thích trong tài liệu nâng cao nhé!

Maps

Kiểu Maps dùng để biểu diễn một bảng dữ liệu gồm có 2 cột Key và Value.

Key	Value
1	"Một"
2	"Hai"
…	

Khảo sát đoạn code sau để khám phá kiểu Maps

```
package main

import "fmt"

func main() {

    numberMap := map[int]string{
        1: "Một",
        2: "Hai",
    }

    for key, value := range numberMap {
        fmt.Println(key, value)
    }

    fmt.Println(numberMap[1])
}
```

```
2 Hai
1 Một
Một
```

Thời gian (Times & dates)

Khám phát đoạn code sau:

```
package main

import (
    "fmt"
    "time"
)

func main() {
```

```
    fmt.Println("Epoch time:", time.Now().Unix())
    t := time.Now()
    fmt.Println(t, t.Format(time.RFC3339))
    fmt.Println(t.Weekday(), t.Day(), t.Month(), t.Year())
    time.Sleep(time.Second)
    t1 := time.Now()
    fmt.Println("Time difference:", t1.Sub(t))
}
```

```
Epoch time: 1605330249
2020-11-14 12:04:09.5610308 +0700 +07 m=+0.001000301 2020-11-14T12:04:09
+07:00 Saturday 14 November 2020
Time difference: 1.0013609s
```

Đây là eBook của riêng bạn – đề nghị không chia sẻ cho ai khác nhé!

Bài tập

Chương trình ①: Viết chương trình tên là GoNameYear nhận 2 tham số từ dòng lệnh. Tham số thứ nhất là Tên, tham số thứ hai là Năm sinh. Ví dụ:

```
GoNameYear.exe Hải 2000
```

Chương trình sẽ hiển thị

```
Chào Hải 20 tuổi
```

Chương trình ②: Mở rộng chương trình trên bằng cách hiển thị thêm một thông báo dạng như sau:

Chào Hải, từ năm bạn sinh ra (năm 2000) đến bây giờ có các năm chia hết cho 4 gồm: 2000, 4004, …

(Phần … là thông tin bạn phải liệt kê đầy đủ).

Ngày 2: Biểu diễn thông tin phức hợp

Bài 6 – Biểu diễn thông tin phức hợp với GO

Cấu trúc (Structure)

Khám phá đoạn code sau:

```
package main

import (
    "fmt"
)

type aStructure struct {
    person string
    height int
    weight int
}

func main() {
    p1 := aStructure{"Thạch", 165, 72}

    fmt.Println(p1)
}
```

```
{Thạch 165 72}
```

Kết hợp Slice và Structure

Khám phá đoạn chương trình sau:

```
package main

import (
    "fmt"
)

type aStructure struct {
    person string
    height int
    weight int
}

func main() {
    pSlice := [2]aStructure{}

    pSlice[0] = aStructure{"Thạch", 165, 72}
    pSlice[1] = aStructure{"Ngọc", 170, 77}

    fmt.Println(pSlice)
}
```

```
[{Thạch 165 72} {Ngọc 170 77}]
```

Con trỏ đến cấu trúc

Tuples (Bộ dữ liệu)

Trong ngày 2, bạn đã làm quen với khái niệm hàm (function), lúc đó chỉ là hàm `calAge` đơn giản nhận một tham số là năm sinh và trả làm một số nguyên là số tuổi.

```
func calAge(birthYear int) int {
    return 2020 - birthYear
}
```

Nhu cầu thực tế có thể phức tạp hơn như yêu cầu hàm trả nhiều thông tin hơn. Khảo sát đoạn chương trình sau:

```
package main

import (
    "fmt"
)

func calAge(birthYear int) (int, string) {
    return 2020 - birthYear, "Đinh Tỵ"
}

func main() {

    myAge, moonAge := calAge(1977)

    fmt.Println(myAge, " ", moonAge)
}
```

```
43 Đinh Tỵ
```

Bạn để ý lúc này hàm `calAge` không phải trả lại một số nguyên (tuổi) nữa mà có thêm một chuỗi cho biết tuổi theo 12 con giáp. Cú pháp của kết quả trả về của hàm được bao đóng trong cặp dấu ngoặc như thế này:

(int, string)

Các viết lệnh return trong hàm cũng có chút khác biệt là

return value1, value2

Bộ giá trị value1, value2 (có thể có nhiều giá trị hơn nữa) gọi lạ `tuple` (bộ)

Minh họa hàm strconv.Atoi

Hàm Ato trong thư viện `strconv` sẽ chuyển một chuỗi các kí tự thành số. Bạn có thể tra cứu thư viện này tại " https://golang.org/pkg/strconv/".

Bạn sẽ thấy ràng hàm atoi sẽ trả lại một bộ gồm 2 giá trị với cú pháp sử dụng như sau:

```
n, err = strconv.Atoi(string)
```

Trong đó tham số string là biến có kiểu string hoặc là literal string bao đóng với cặp nháy đôi.

n và err lần lượt là 2 biến kết xuất: giá trị số bạn cần nhận, thông báo lỗi (nếu có)

Khảo sát ví dụ sau:

```
package main

import (
    "fmt"
    "strconv"
)

func main() {

    n, err := strconv.Atoi("123A")
    fmt.Println("Lỗi: ", err)
    fmt.Println("n: ", n)
}
```

```
Lỗi: strconv.Atoi: parsing "123A": invalid syntax
n: 0
```

Hãy sử lại tham số "123A" thành "123" thì kết quả sẽ như sau:

```
Lỗi: <nil>
n: 123
```

Vài nhận xét:

- Khi dữ liệu hợp lệ thì err sẽ bằng nil. Đây là giá trị đặc biệt có nghĩa là "không có gì cả". Các ngôn ngữ lập trình khác như C,C++, Java, Python gọi là Null.
- Khi dữ liệu không hợp lệ thì err sẽ chứa thông báo cụ thể. Tức là khác nil

Đọc thêm và thực hành

Chuỗi (String)

Có thể xem String là thông tin phức hợp vì nó được tạo thành từ các kí tự (char).

Trong Bài 5, bạn đã làm quen với kiểu chuỗi với vài thao tác đơn giản. Phần này sẽ giúp các bạn mở rộng thêm kiến thức của mình trong việc khai thác kiểu dữ liệu chuỗi.

Chuyển đổi kiểu chuỗi thành số

Một tình huống đặt ra cho các bạn là khi viết chương trình cần nhận tham số đầu vào từ dòng lệnh có ý nghĩa là số như ví dụ sau: Bạn cần viết chương trình tính toán năm sinh dương lịch để hiển thị ra năm âm lịch theo con giáp. Ví dụ năm 1984 là năm Giáp Tý. Cách chạy chương trình bằng lệnh như sau:

```
amlich 1984
```

Chương trình sẽ hiển thị ra chữ:

```
Giáp Tý
```

Như vậy bạn cần áp dụng kiến thức của Bài 4 để biết cách lấy tham số từ dòng lệnh bằng cách truy xuất mảng `os.Args`. Tuy nhiên khi lấy tham số được truyền từ dòng lệnh như os.Args[1] thì kết quả là một String.

Để chuyển từ string sang kiểu số thì dùng thư viện `strconv` (viết tắt của string conversion). Bạn tập xem tài liệu tại:

https://godoc.org/strconv

Hãy tra cứu thêm tài liệu tại trang " http://buaphep.net/2020/02/06/cach-chuyen-doi-nam-duong-lich-sang-nam-am-lich/" để hoàn thành chương trình Âm Lịch ở trên.