



Th.S LÊ NGỌC THẠCH

Lời nhắn

eBook "**Chạm tới GO trong 10 ngày**" này dự kiến phát hành vào tháng 12/2021. Bạn có thể đặt hàng ngay bây giờ với ưu đãi giảm 50% chỉ **199K**, tiết kiệm 200K. Thanh toán nhanh theo 2 cách:

① MoMo	② Chuyển khoản
<p>☎ 0908550642 👤 Lê Ngọc Thạch</p> <p>📄 Nội dung tin nhắn: GO2021 email sdt Ví dụ: abc@gmail.com 0908550642 Email và sdt của người nhận eBook.</p> <p>Trường hợp tặng bạn bè thì ghi thông tin email và sdt của bạn.</p> <p>Quét mã QR thanh toán 199K.</p> 	<p>Lê Ngọc Thạch, Ngân Hàng Tiên Phong, CN HCM Số tài khoản: 00002888001 Nội dung tin nhắn: GO2021 email sdt Vd tin nhắn: GO2021 abc@gmail.com 0908456321 Quét mã QR để thanh toán cho:</p>  <p>Quét mã vạch này để giao dịch</p>

Ngoài ra, bạn có thể đọc ngay bản nháp hiện tại với giá 0đ theo cách sau:

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

<https://minepi.com/thachln>

Sử dụng invitation code: **thachln**

Liên lạc với tác giả qua <https://facebook.com/ThachLN> để cung cấp account MinePI, SĐT và Email nhận nhận eBook với thông tin mã hóa đính kèm.

Lê Ngọc Thạch

CHẠM TỚI GO TRONG 10 NGÀY

Mục lục

Mục lục	2
Quy ước	9
Ngày 1: Giới thiệu	11
Bài 1 – Tại sao GO ra đời.....	12
Bài 2: Ngôn ngữ lập trình GO	13
Biến (Variable), Cấu trúc (Structure).....	13
Variable có nghĩa là gì?	16
Khai báo biến (variable declaration)	17
Lệnh gán (assign)	17
Bài 3 – Chuẩn bị môi trường lập trình	20
GO Core	20
Visual Code	20
Bài 4 – Viết chương trình đơn giản với GO.....	23
Viết mã	23
Biên dịch	23
Chạy trực tiếp mã nguồn.....	25
Phép gán (assign)	27
Các toán tử cơ bản.....	28
Hàm (function)	28
Chạy chương trình có tham số dòng lệnh trong Visual Code.....	29
Lấy tham số từ dòng lệnh.....	30
Vòng lặp (loops).....	31
Sử dụng Logging.....	32
Bài 5 – Biểu diễn thông tin đơn giản với GO	33
Kiểu chuỗi (string)	33

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

Kiểu dữ liệu số (Numeric data types)	34
Viết chương trình Fibonacci	39
Mảng (arrays)	40
Slice (chưa biết gọi tiếng Việt là gì)	42
Maps	45
Thời gian (Times & dates)	47
Tra cứu định dạng	48
Bài tập.....	50
Ngày 2: Biểu diễn thông tin phức hợp	51
Bài 1 – Biểu diễn thông tin phức hợp với GO	52
Cấu trúc (Structure).....	52
Kết hợp Slice và Structure	52
Con trỏ (Pointer)	53
Tuples (Bộ dữ liệu)	55
Đọc thêm và thực hành.....	58
Chuỗi (String)	58
Regular expressions and pattern matching.....	59
Bài 2 – Viết hàm cho cấu trúc	61
Phân tích hàm calculateBMI cho struct Employee	61
Tổ chức thành thư viện (module).....	62
Bài 3 – Dữ liệu dạng JSON.....	65
Đọc dữ liệu JSON	65
Ngày 3: Cấu trúc điều khiển.....	65
Bài 7 – Cấu trúc rẽ nhánh.....	66
Lệnh if	66
Switch.....	66
Bài 8 – Vòng lặp.....	69
Vòng lặp (loops).....	69
Vòng lặp for nâng cao	71
Ngày 4: Làm việc với dữ liệu trên đĩa cứng.....	75
Bài 1 – Làm việc với thư mục và file.....	76
Lấy thông tin về file/thư mục.....	76

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

Lấy nội dung thư mục	77
Lấy nội dung file	77
Lưu file.....	78
Lưu và đọc file mã hóa.....	79
Bài 2 – Làm việc với file CSV	81
Bài 3 – Đọc file CSV.....	83
Bài 4 – Ghi file CSV	85
Bài 5 – Đọc file vào cấu trúc (struct)	86
Cài đặt thư viện	86
Đọc đoạn dữ liệu binary vào mảng các struct.....	86
Đọc file CSV vào mảng các struct.....	87
Bài 6 – Đọc file văn bản.....	91
Bài 7 – Đọc file Excel	93
Cài đặt	93
Đọc file Excel.....	93
Ghi dữ liệu ra file Excel.....	94
Ngày 5: Tổ chức dự án GOLANG	95
Bài 1 – Tổ chức mã nguồn	96
Bước 1: Tạo file go.mod để mô tả tên của module.....	96
Bước 2: Tạo thư mục và file chứa hàm dùng chung.....	97
Bước 3: Viết chương trình chính	97
Bài 2 – Tinh chỉnh mã nguồn	100
Phiên bản 0.0.2.....	100
Phiên bản 0.0.3	101
Phiên bản 0.0.4	102
Do it yourself:.....	104
Ngày 6: Sử dụng cơ sở dữ liệu PostgreSQL.....	104
Bài 1 – Làm quen với CSDL.....	105
Bài 2 – Sử dụng PostgreSQL portable	106
Tải gói binary	106
Tạo file khởi động PostgreSQL server	107
Khởi động PostgreSQL server	108

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

Tương tác với PostgreSQL Server qua dòng lệnh	108
Tương tác với PostgreSQL Server qua web site	111
Tương tác với PostgreSQL Server qua web site	115
Bài 3 – Thực hành với PostgreSQL	116
Tạo một CSDL “ECP”	116
Nhập dữ liệu	121
Bài 4 – GOLANG và PGSQL	123
Cài thư viện	123
Ví dụ	123
Giải thích code từ ví dụ	124
Ngày 7: Lập trình đồng thời và song song với GO	127
Bài 1 – Khái niệm Concurrency và Parallelism	128
Bài 2 – Khái niệm Concurrency và Parallelism	129
Bài 3 – Lập trình Concurrency	130
Bài 4: Lập trình Parallelism	131
Ngày 8: GOLANG và C/C++	131
Bài 1 - Lập trình C trong GO	132
Ngày 9: Các chủ đề mở rộng/nâng cao	132
Bài 1 – Viết hàm với tham số linh động	133
Variadic functions	133
Bài 2 - Crawl dữ liệu với GOLANG	135
Request đơn giản	135
Thiết lập timeout cho request	135
Thiết lập header	136
Download URL	137
Use substring	137
Bài 3 - Lập trình CUDA với GOLANG	139
Bài 4 - Phát triển Web Application với Beego	140
Cài đặt GO	140
Cài đặt Beego	140
Tạo dự án	140
Chạy ứng dụng	141

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

Truy cập ứng dụng	142
Chỉnh sửa code	142
Tạo API	143
Bài 5 - Phát triển Web Backend với Gin-Gonic.....	144
Cài đặt	144
Viết Backend đơn giản.....	144
Triển khai lên server Ubuntu với Nginx	146
Nâng cao.....	146
Cài đặt các thư viện hỗ trợ web	146
Bài 6 - Sử dụng GOLANG trong WSL2.....	148
Ngày 10: Các chủ đề mở rộng/nâng cao	148
Ngày 11: Testing với GO	148
Biên dịch OpenCV từ mã nguồn	149
Cài đặt Anaconda:.....	149
Cài đặt mkl-service	149
Kiểm tra thông tin thiết bị GPU	149
Biên dịch	150
Ngày 12 - Blockchain.....	151
Bài 1: Ôn tập kiến thức cơ bản.....	152
Làm quen lại với kiểu Slice của byte	152
Thư viện bytes.....	153
Thư viện mã hóa.....	153
Mã hóa base58	154
Khóa công khai và khóa bí mật.....	155
Bài 2 – Tạo cấu trúc chuỗi khối	157
Tạo dự án.....	157
Viết mã nguồn main.go	157
Bài 3 – Minh họa thuật toán đồng thuận ProofOfWork.....	162
Định nghĩa bổ sung Block.....	162
Hàm tạo Block cũ.....	162
Hàm tạo Block cải tiến.....	162
Cài đặt ProofOfWork (PoW)	163

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

Bài 4 – Lưu trữ Blockchain.....	166
Sử dụng database dạng Key-Value	166
Đóng gói OpenSSL	168
Chuẩn bị công cụ.....	168
Cài đặt tool Visual Studio 2019	168
Clone mã nguồn dự án OmiseGo eWallet	169
Lập trình wxWidget	171
Phụ lục	171
Phụ lục 1: Quá trình tiến hóa của các mô hình phần mềm.....	171
Phần mềm trên máy cá nhân	172
Máy vi tính cá nhân (personal computer)	172
Giao diện console	173
Giao diện đồ họa (GUI – Graphics User Interface)	175
Phần mềm trên mạng nội bộ.....	177
Mạng nội bộ (LAN - Local Network).....	177
Phần mềm trên nền tảng mạng Internet.....	179
Mạng Internet	179
Phần mềm trên mạng Internet	179
Phụ lục 2 – Cấu trúc của phần mềm.....	181
Công thức I + P + O	182
Thu nhận thông tin	182
Xử lý thông tin	182
Xuất kết quả.	182
Ví dụ 1:.....	182
Phụ lục 3	183
Lập trình giao diện với goki	184
Cài đặt GCC for Windows	184
Cài đặt thư viện goki	185
Viết ứng dụng.....	185
Biên dịch và chạy ứng dụng.....	186
GOLANG và QT	187
Cài đặt phần mềm QT	187

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

Cài đặt thư viện	192
Trải nghiệm lập trình.....	193
Phụ lục 4	195
GOLANG và Google Sheet.....	195

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

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 cặp **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 cặp 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).

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

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 nguồn 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 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 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. Đầu 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

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

(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

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

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:

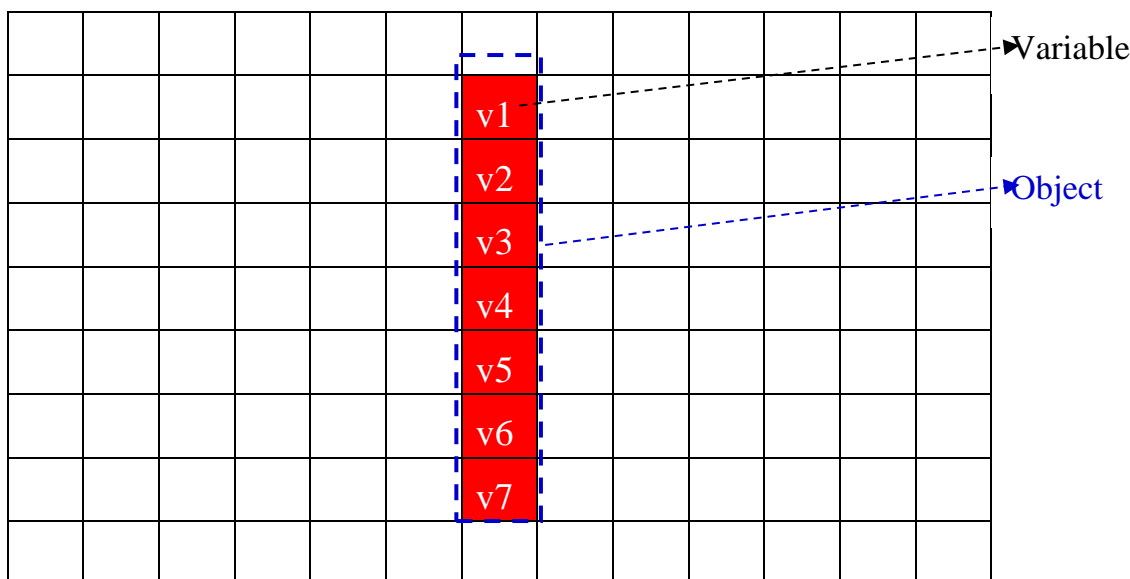
Structure: myFriendThach	
fullName	Lê Ngọc Thạch
height	165 cao
weight	70.5 kg
sex	Nam
birthday	29/9/1977
favorNumbers	1, 2, 5, 10, 20, 50
favorSports	Bóng bàn, bóng đá, Quần vợt

Trong bảng trên xuất hiện từ **myFriendThach**, đây là một cái tên (name) được máy tính chỉ định (hoặc là **trở 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) trở 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.

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



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:

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

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 từ 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>
```

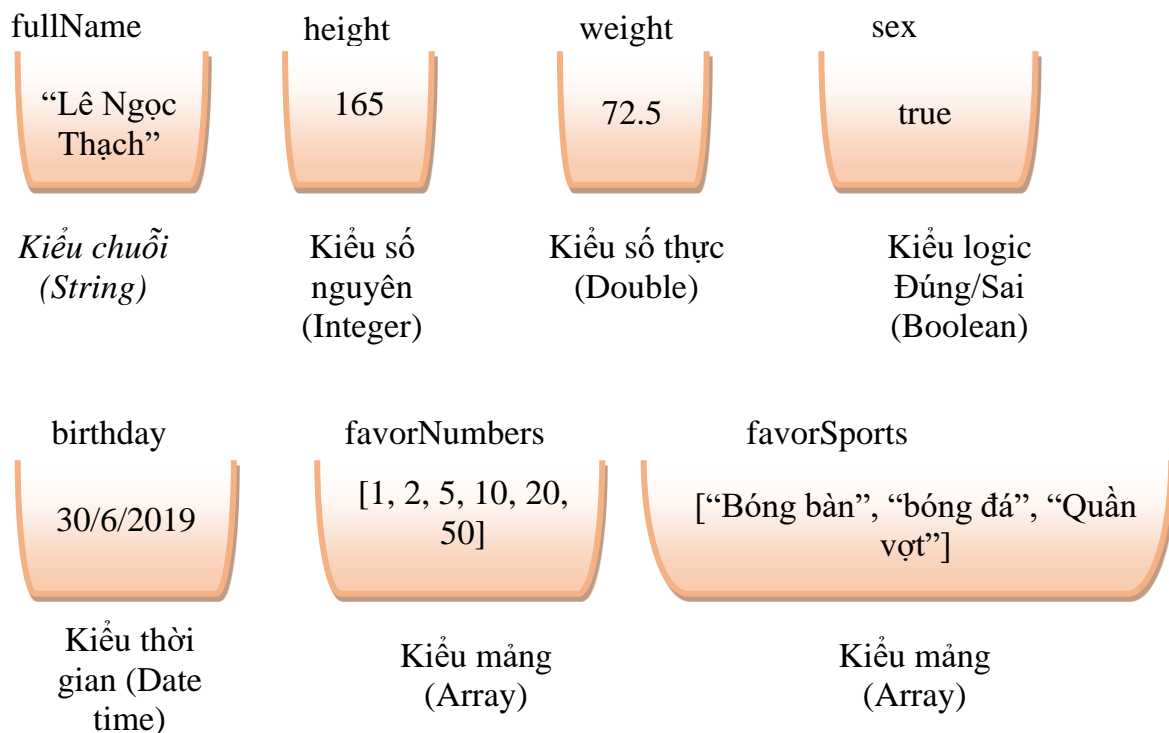
Ví dụ 2 dòng lệnh sau sẽ khai báo 2 vùng nhớ tương ứng cho fullName, height với kiểu dữ liệu tương ứng là string (chuỗi) và int (số nguyên)

```
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.

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



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 đã khai báo biến thì GO dùng dấu bằng `=` để thiết lập, hoặc thay đổi giá trị của biến.

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

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

Bài 3 – Chuẩn bị môi trường lập trình

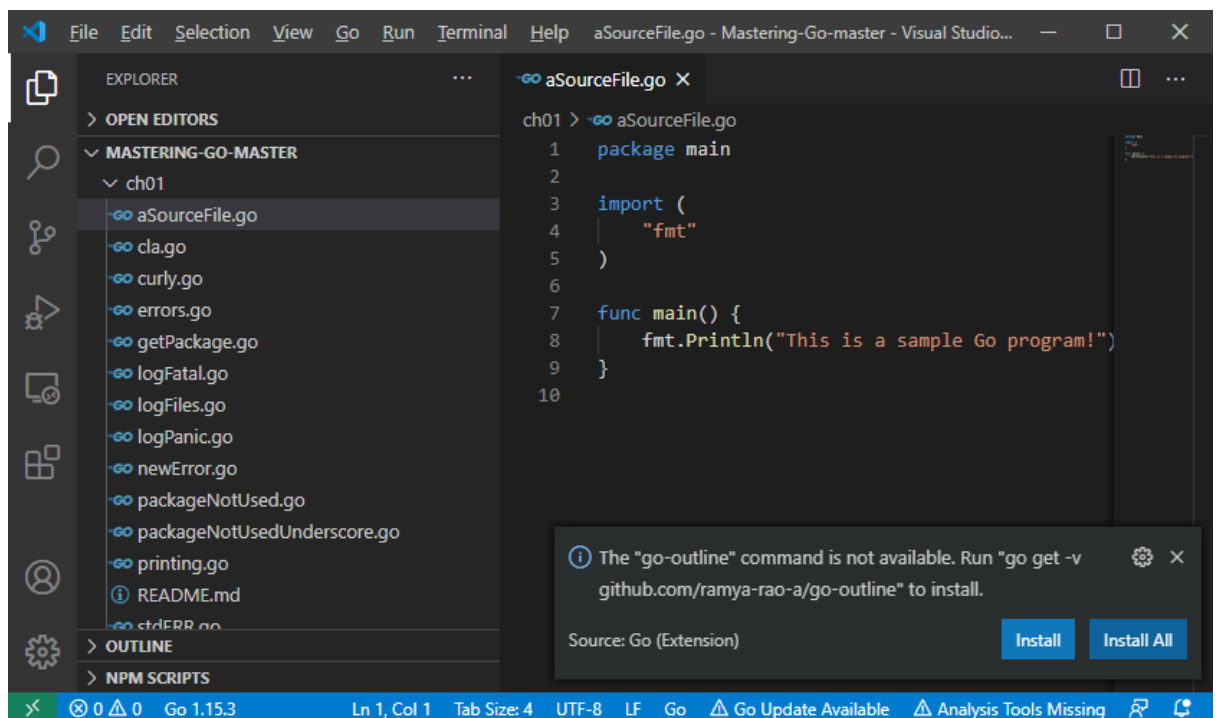
GO Core

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

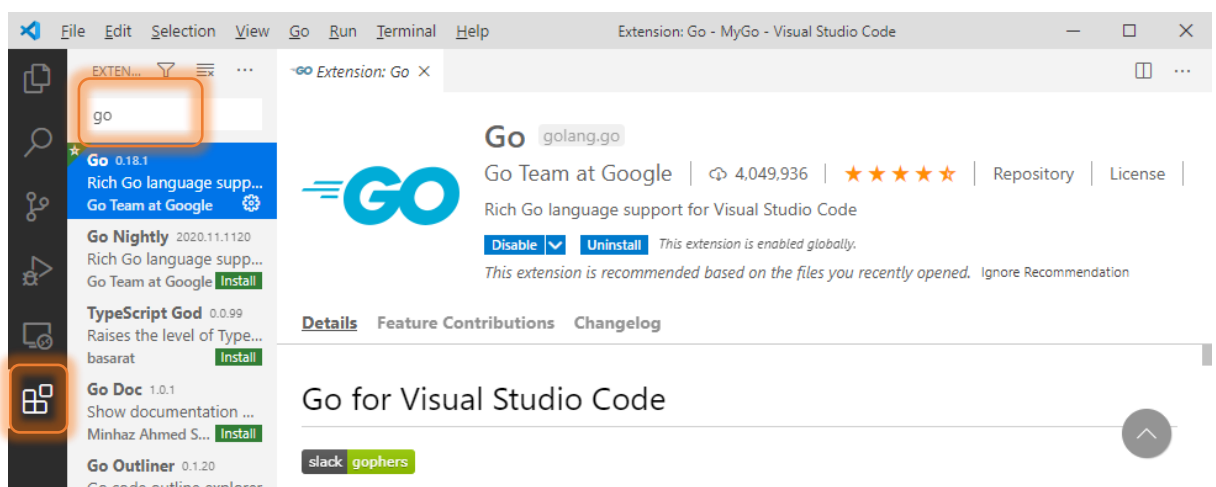
<https://golang.org/dl>

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 extensions cơ bản



Phím tắt trong Visual Code

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

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

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

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

Cài extentions nâng cao

Gọi là nâng cao thôi chứ thật ra cũng không phải cao gì đâu. Chỉ là các công cụ này có nhiều chức năng hay mà nếu bạn khai thác tốt thì giúp cải thiện đáng kể năng suất lập trình.

#	Từ khóa	Ghi chú	Link
1	docs-markdown		Trang chủ

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

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 dấu nháy đôi).

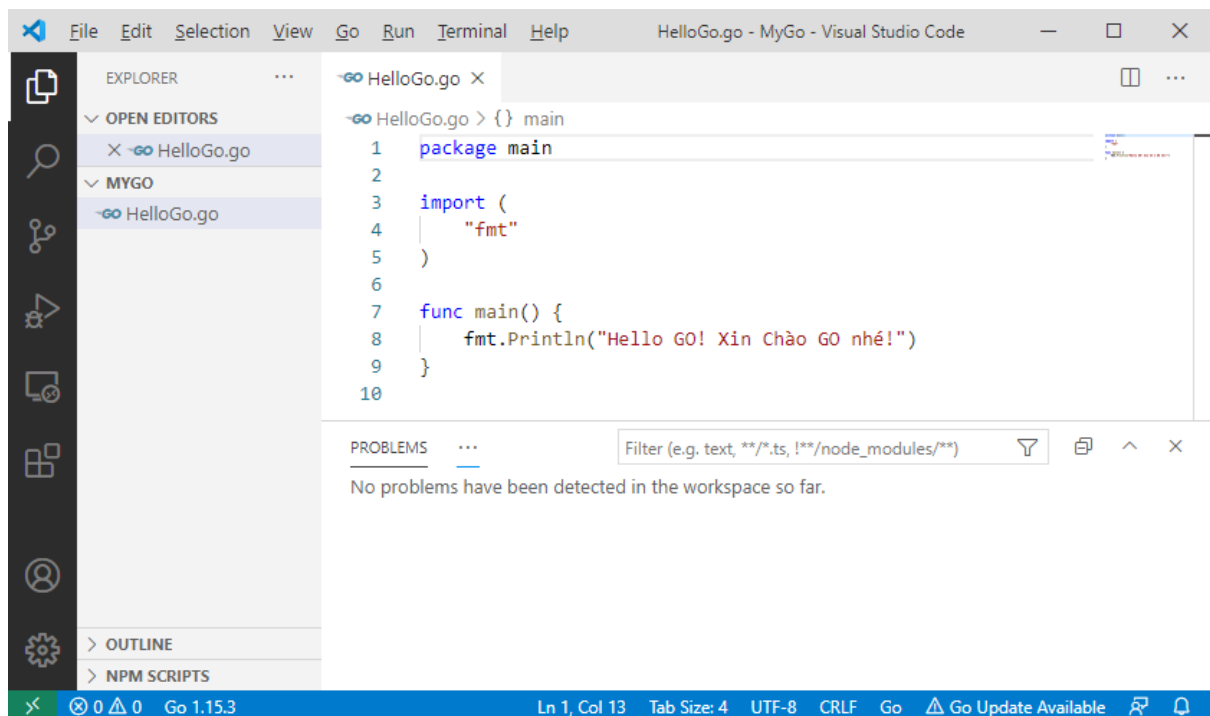
Biên dịch

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
```

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



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 tiền file đầu đủ bắt đầu có chữ H. Trong trường hợp này là HelloGo.go. Kết quả lệnh đầy đủ là:

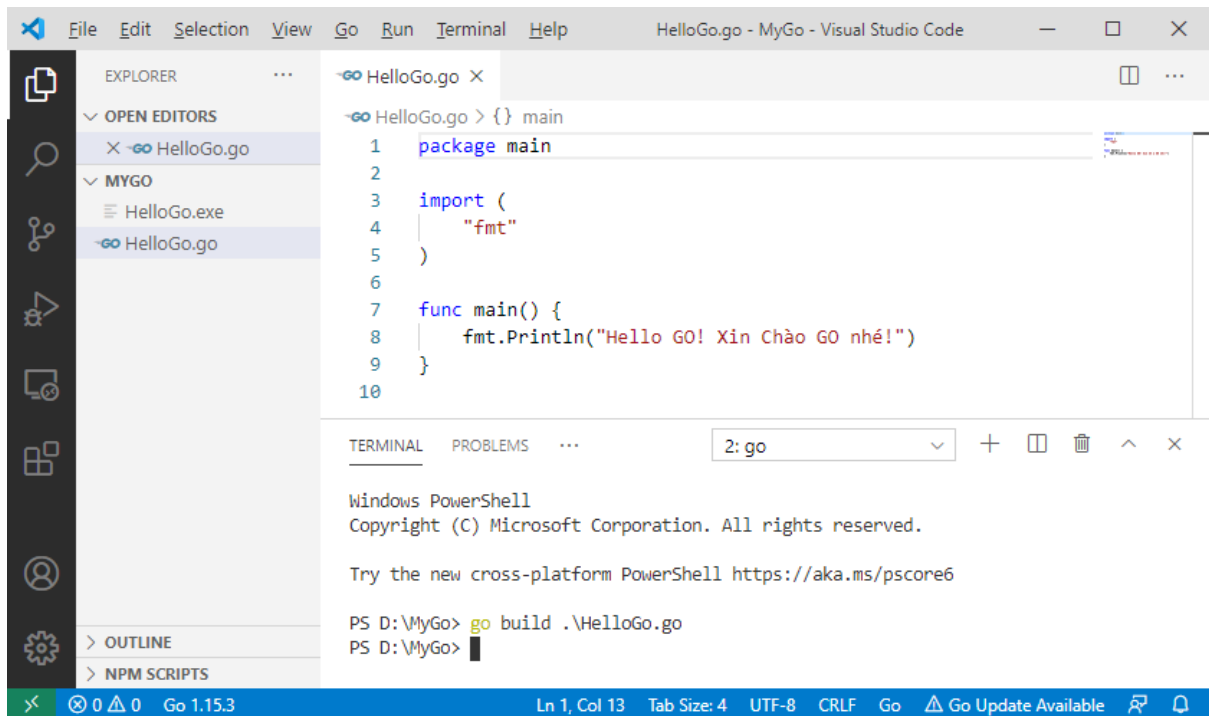
```
go build .\HelloGo.go
```

Kí hiệu dấu chấm có nghĩa là thư mục hiện hành. ".\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 tự như sau:

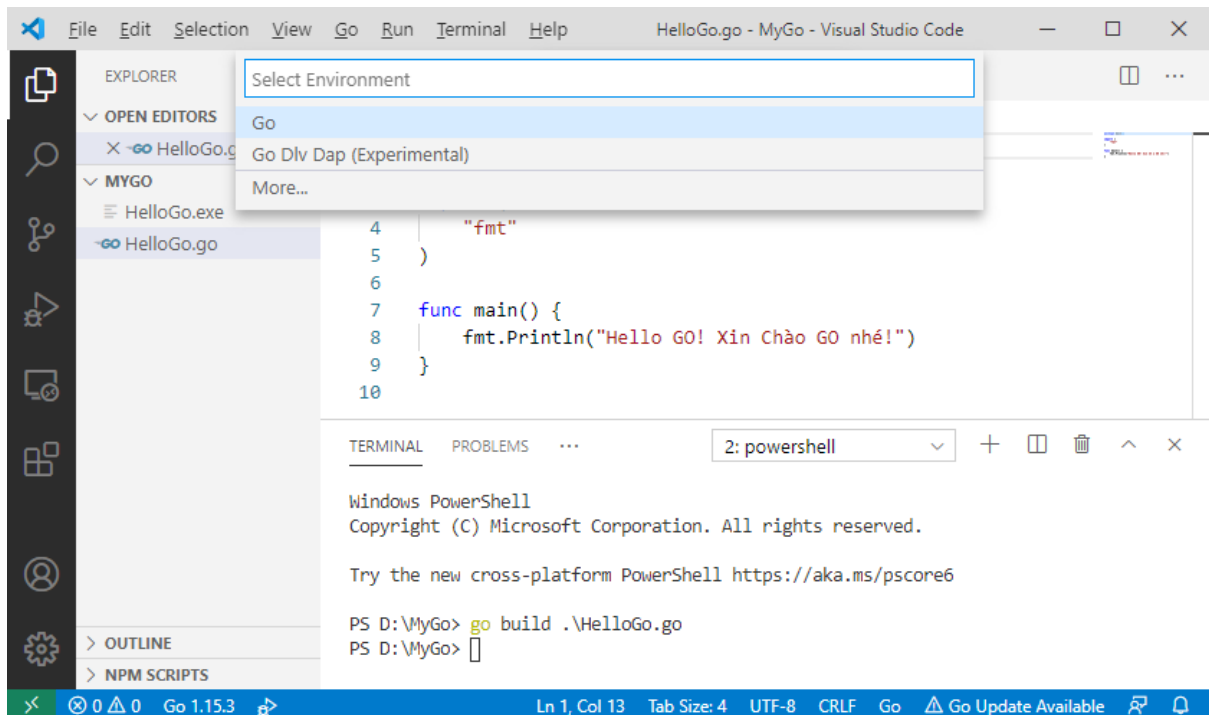
Bạn gõ chữ H rồi nhấn phím Tab, cửa sổ lệnh sẽ thông minh hiển thị sẵn cho mình lệnh .\HelloGo.exe. Xong nhấn Enter như hình bên dưới.

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

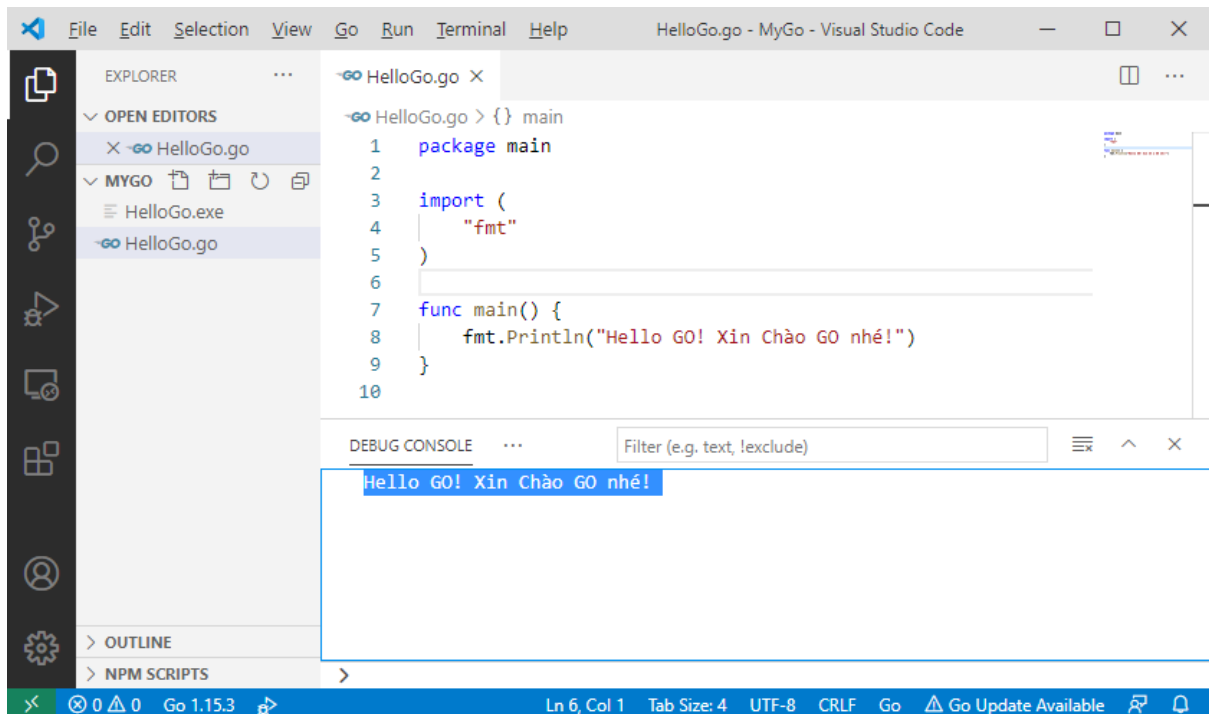


Chạy trực tiếp mã nguồn

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:

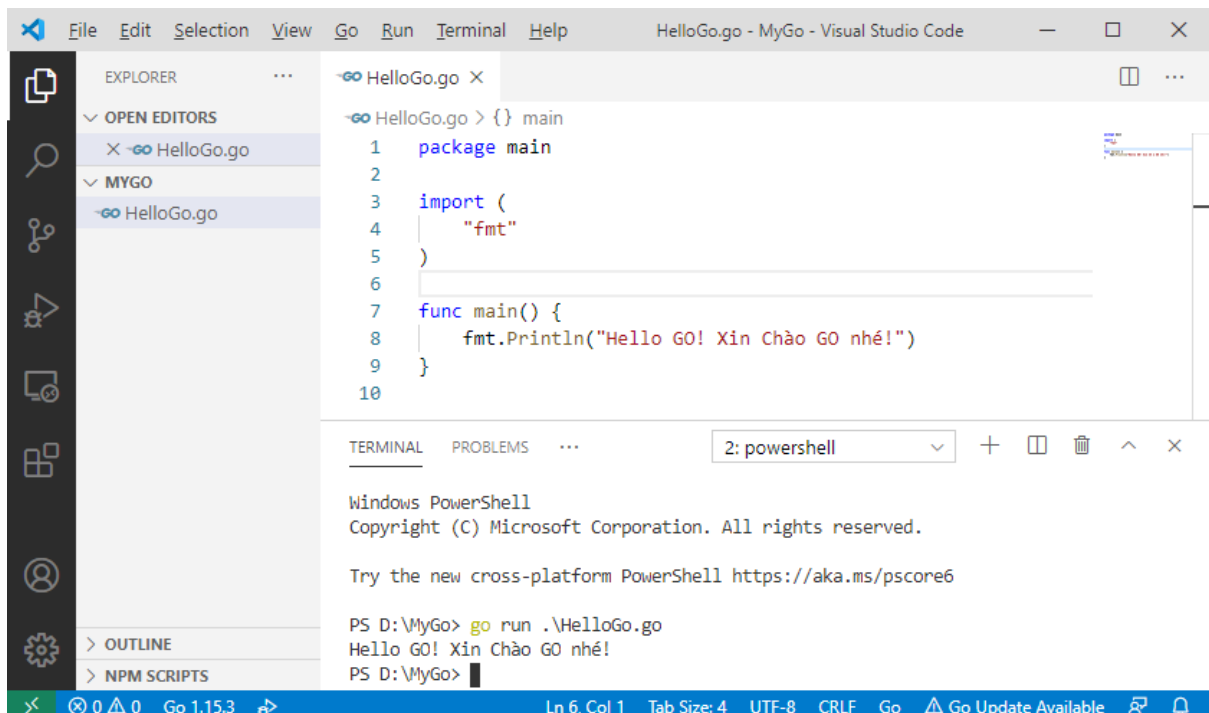


Đây là eBook của riêng bạn – đề nghị không chia sẻ cho ai khác nhé!



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

```
go run HelloGo.go
```



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

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!

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

Khái niệm hàm trong lập trình là cách để người lập trình chia nhỏ một chương trình lớn thành các đoạn chương trình nhỏ hơn. Các chương trình nhỏ này có thể được tái sử dụng nhiều lần trong các tình huống khác nhau bằng cách thay đổi các thông số đầu vào.

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

Để viết hàm thì dùng cú pháp như sau:

```
func tên_hàm(tham_số) kết_quả_trả_về {  
}
```

Khảo sát 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)  
}
```

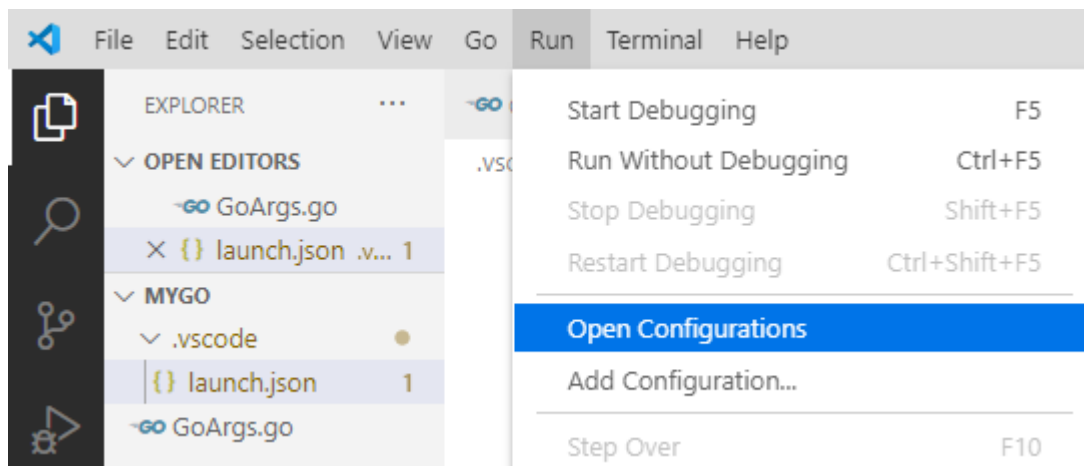
Vài điểm chú ý về khai báo hàm:

- 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

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



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": ["Hài", "2000"]
```

Lựa 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
```

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

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 chạy 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)  
    }  
}
```

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

Sử dụng Logging

Trong các bài trước để ghi các lỗi, hoặc các thông tin để theo dõi chương trình chạy ra màn hình thì các bạn dùng hàm `Println` trong thư viện `fmt`. Việc hiển thị các thông tin theo dõi, hoặc các lỗi như thế này gọi chung là quá trình truy vết (tracing) hoặc logging. Có vài vấn đề liên quan đến việc sử dụng hàm `Print`, `Println`, hoặc `Printf` trong tình huống này, nói chung là không hợp lý cho các dự án thực tế. Cụ thể:

- Các lỗi này phải được kết xuất ra một nơi nào đó (như file, gọi chung là logging file; hoặc database) để có thể phân tích sau này.
- Các lệnh truy vết như thế này phải đảm bảo không làm ảnh hưởng lớn đến tốc độ thực thi của chương trình. Ví dụ một lệnh `fmt.Println` đơn giản cũng phải tốn thời nhất định (dù rất ít).

Để thực hiện truy vết, theo dõi chương trình chạy thì trong thực tế người ta sử dụng các thư viện logging. Một trong các thư viện logging cho ngôn ngữ GO là thư viện `logrus` này:

github.com/sirupsen/logrus

Để cài đặt thư viện `logrus` thì thực hiện lệnh sau trong dấu nhắc lệnh của hệ điều hành.

```
go get github.com/sirupsen/logrus
```


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

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 cặp 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

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

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.Println` với định dạng là `%c`.

Kiểu dữ liệu số (Numeric data types)

Số nguyên (Integer)

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.

Dưới đây là bảng các giá trị nhỏ nhất và lớn nhất

uint8	0 ~ 255
uint16	0 ~ 65535
uint32	0 ~ 4294967295
uint64	0 ~ 18446744073709551615
int8	-128 ~ 127
int16	-32768 ~ 32767
int32	-2147483648 ~ 2147483647
int64	-9223372036854775808 ~ 9223372036854775807

Vài ví dụ để bạn tự khám phá:

Ví dụ 1:

```
package main

import "fmt"

func main() {

    var n uint8

    n = 255

    fmt.Println(n)
```

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

```
n = n + 1

fmt.Println(n)
}
```

Ví dụ 2:

```
package main

import "fmt"

func main() {

    var n uint8

    n = 255

    fmt.Println(n)

    n = n + 1

    fmt.Println(n)

    // New

    n = n - 1

    fmt.Println(n)

}
```

Ví dụ 3:

```
package main

import "fmt"

func main() {

    var n int8

    n = 127

    fmt.Println(n)
    n = n + 1

    fmt.Println(n)

    n = n - 1

    fmt.Println(n)

}
```

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

```
}
```

Trong GOLANG, một kí tự được xem như là một số nguyên. Khác với các ngôn ngữ lập trình khác có kiểu char. Hãy khảo sát đoạn chương trình sau:

```
package main

import (
    "fmt"
    "reflect"
)

func main() {

    ch := 'a'
    fmt.Println(ch)
    fmt.Printf("%c", ch)
    fmt.Println()
    fmt.Println(reflect.TypeOf(ch))

    ch = 'A'
    fmt.Println(ch)

    ch = '0'
    fmt.Println(ch)

    ch = '1'
    fmt.Println(ch)
}
```

Kết quả

```
97
a
int32
65
48
49
```

Như vậy kí tự ‘a’, ‘A’, ‘0’, ‘1’ có giá trị nguyên lần lượt là: 97, 65, 48, 49.

Để hiển thị kí tự ra màn hình tương ứng với giá trị nguyên của nó thì cùng kí hiệu %c trong hàm fmt.Printf. Hãy thử lệnh sau:

```
fmt.Printf("%c", 98)
```

Để hiển thị các biến số nguyên ra màn hình trong lệnh fmt.Print(..) thì dùng kí hiệu %d trong mẫu lệnh sau:

```
fmt.Printf("...%d...%d", i1, i2)
```

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

Số thực (floating-point numbers)

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

Vài ví dụ để bạn trải nghiệm với số thực bằng cách dự đoán kết quả của các lệnh `fmt.Println` và chạy lại chương trình để đúc kết kinh nghiệm.

Ví dụ 1:

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var f32 float32 = 1.2
    var f64 float64 = 1.3
    fmt.Println(f32)
    fmt.Println(f64)
    fmt.Println(math.MaxFloat32)
    fmt.Println(math.MaxFloat64)
}
```

Ví dụ 2:

```
package main

import "fmt"

func main() {
    n := 2
    m := 3
    result := n / m
    fmt.Println(result)
}
```

Ví dụ 3:

```
package main

import "fmt"

func main() {
    n := 2.0
    m := 3.0
    result := n / m
    fmt.Println(result)
}
```

Kết quả:

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

```
0.6666666666666666
```

Để hiển thị giá trị của các biến số thực ra màn hình trong các lệnh `fmt.Printf()` thì dùng 2 dạng sau:

- ① `fmt.Printf("... %.mf", f1)`
- ② `fmt.Printf("...%.nf...%.mf", f1)`

m trong 2 lệnh trên là con số cụ thể cho biết số lượng số lẻ trong phần thập phân cần hiển thị ra màn hình.

n.m trong lệnh có ý nghĩa như sau:

- ✓ Hiển thị m số lẻ trong phần thập phân
- ✓ Hiển thị thêm khoảng trắng phía trước sao cho tổng cộng (số khoảng trắng + số ký tự phần nguyên + dấu chấm + số ký tự phần thập phân) là n ký tự

Hãy tự trải nghiệm các đoạn chương trình sau.

Ví dụ 1: Sử dụng hàm `fmt.Printf(...)` để hiển thị số nguyên và số thực.

```
package main

import "fmt"

func main() {
    fmt.Printf("Năm %d là năm chẵn\n", 2020)
    fmt.Printf("Kết quả của 2 chia cho 3: %f", 2.0/3.0)
}
```

Ví dụ 2: Sử dụng hàm `fmt.Println(...)` hoặc `fmt.Print()` để hiển thị số thực.

```
package main

import "fmt"

func main() {
    n := 2.0
    m := 3.0
    result := n / m
    fmt.Println(result)
}
```

Ví dụ 3:

```
package main

import "fmt"

func main() {
    fNumber := 34567.123456789
}
```

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

```
fmt.Printf("Hiển thị số thập phân mặc định: %f", fNumber)
fmt.Println()
fmt.Printf("Hiển thị số thập phân với 2 số lẻ:\n%.2f", fNumber)
fmt.Println()
fmt.Printf("Hiển thị số thập phân với 2 số lẻ và thêm khoảng trắng
vào phía trước cho đủ 10 kí tự:\n%10.2f", fNumber)
}
```

Ví dụ 4: Tính giá trị của e^x với x chạy từ 0 đến 7. Sau đó in ra kết quả với lề được anh phải.

```
package main

import (
    "fmt"
    "math"
)

func main() {
    for x := 0; x < 8; x++ {
        fmt.Printf("x = %d, e^%d = %8.3f\n", x, x, math.Exp(float64(x)))
    }
}
```

Kết quả:

x = 0,	e^0 =	1.000
x = 1,	e^1 =	2.718
x = 2,	e^2 =	7.389
x = 3,	e^3 =	20.086
x = 4,	e^4 =	54.598
x = 5,	e^5 =	148.413
x = 6,	e^6 =	403.429
x = 7,	e^7 =	1096.633

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 Fibonacci 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:

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

- 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 tử là 1, 2, 3, 4.

```
package main

import "fmt"

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


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

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

Kết quả:

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

Bạn tự đoán ý nghĩa của 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`.

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

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 slice bên trong hàm thì sau khi kết thúc hàm thì giá trị slice 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))
}
```

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

Tạo slice 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)

}
```

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

Truy xuất các phần tử của slice

Để truy cập 1 phần tử của slice 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=", cap(aSlice))

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

Kết quả:

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

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

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

Bạn có rút ra được điều gì không?

Slice 2 chiều

Tương tự như mảng 2 chiều thì slice 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 sánh kết 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
```

Chuyển slice thành array

Tạm dừng việc làm quen kiểu dữ liệu `slice` ở đây. Còn nhiều điều thú vị về slice 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.

Ví dụ một bảng Map đơn giản ánh xạ 1 số nguyên thành 1 chữ (Text):

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

Key	Value
1	"Một"
2	"Hai"
...	

Cú pháp khai báo

```
map[key_type] value_type
```

key_type: là kiểu dữ liệu của khóa

value_type: là kiểu dữ liệu của giá trị

Ví dụ

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
```

Khởi tạo Map rỗng

Để tạo một bảng Map rỗng thì dùng lệnh make theo cú pháp chung như sau:

```
make(map[key_type] value_type)
```

Ví dụ tạo bảng Map có khóa là chuỗi, giá trị là số nguyên:

```
package main
```

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

```
import (  
    "fmt"  
    "reflect"  
)  
  
func main() {  
    iMap := make(map[string]int)  
  
    fmt.Printf("Number of items: %d\n", len(iMap))  
    fmt.Printf("Data type of iMap: %s", reflect.TypeOf(iMap))  
}
```

```
Number of items: 0  
Data type of iMap: map[string]int
```

Thời gian (Times & dates)

Khám phá đoạn code sau để làm quen với thư viện `time`:

```
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 2020-11-14T1  
2:04:09+07:00 Saturday 14 November 2020  
Time difference: 1.0013609s
```

Chuyển một chuỗi ngày tháng năm thành biến thời gian

Khi có nhu cầu chuyển một chuỗi “31/12/1980” với ý nghĩa là ngày 31 tháng 12 năm 1980 thành đối tượng kiểu thời gian thì cần sử dụng hàm `Parse` của thư viện `time`. Hãy thử đoạn chương trình sau:

```
package main  
  
import (  
    "fmt"  
    "reflect"  
    "time"
```

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

```
)  
  
func main() {  
    strDate := "31/12/1980"  
  
    // "2006-01-02" since that is the yyyy-mm-  
    dd formatting of the magical reference date  
    myDate, err := time.Parse("02/01/2006", strDate)  
    if err != nil {  
        fmt.Println(err)  
    } else {  
        fmt.Println(myDate)  
    }  
  
    fmt.Println("Type of myDate: ", reflect.TypeOf(myDate))  
}
```

Kết quả hiện thị như sau:

```
1980-12-31 00:00:00 +0000 UTC  
Type of myDate: time.Time
```

Vài nhận xét

- GOLANG sử dụng định dạng 02/01/2006 cho tham số thứ nhất trong hàm `time.Parse` như là định dạng kiểu ngày như dd/mm/yyyy. Ý nói 02 có nghĩa là ngày, 01 có nghĩa là tháng và 2006 có nghĩa là năm. Để trải nghiệm thêm thì hãy thử sửa lại 2 lệnh sau:

```
strDate := "12/31/1980"  
myDate, err := time.Parse("01/02/2006", strDate)
```

Kết quả sẽ hiển thị ra đúng Năm là 1980, tháng là 12, ngày là 31 như sau:

1980-12-31 00:00:00 +0000 UTC

00:00:00 là không giờ, không phút, không giây thì các bạn hiểu.

+0000 UTC là gì chưa hiểu thì thôi tạm bỏ qua nhé!

Tra cứu định dạng

Tra cứu các định dạng về ngày trong các ngôn ngữ GO, Java và C

Cú pháp GO	Cú pháp Java	Cú pháp C	Ghi chú
2006-01-02	yyyy-MM-dd	%F	ISO 8601
20060102	yyyyMMdd	%Y%m%d	ISO 8601
January 02, 2006	MMMM dd, yyyy	%B %d, %Y	
02 January 2006	dd MMMM yyyy	%d %B %Y	

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

02-Jan-2006	dd-MMM-yyyy	%d-%b-%Y	
01/02/06	MM/dd/yy	%D	US
01/02/2006	MM/dd/yyyy	%m/%d/%Y	US
010206	MMddyy	%m%d%y	US
Jan-02-06	MMM-dd-yy	%b-%d-%y	US
Jan-02-2006	MMM-dd-yyyy	%b-%d-%Y	US
06	yy	%y	
Mon	EEE	%a	
Monday	EEEE	%A	
Jan-06	MMM-yy	%b-%y	

Tra cứu các định dạng về giờ trong các ngôn ngữ GO, Java và C

Cú pháp GO	Cú pháp Java	Cú pháp C	Ghi chú chú
15:04	HH:mm	%R	
15:04:05	HH:mm:ss	%T	ISO 8601
3:04 PM	K:mm a	%l:%M %p	US
03:04:05 PM	KK:mm:ss a	%r	US

Tra cứu các định dạng về ngày giờ trong các ngôn ngữ GO, Java và C

Go layout	Java notation	C notation	Notes
2006-01-02T15:04:05	yyyy-MM-dd'T'HH:mm:ss	%FT%T	ISO 8601
2006-01-02T15:04:05-0700	yyyy-MM-dd'T'HH:mm:ssZ	%FT%T%z	ISO 8601
2 Jan 2006 15:04:05	d MMM yyyy HH:mm:ss	%e %b %Y %T	
2 Jan 2006 15:04	d MMM yyyy HH:mm	%e %b %Y %R	
Mon, 2 Jan 2006 15:04:05 MST	EEE, d MMM yyyy HH:mm:ss z	%a, %e %b %Y %T %Z	RFC 1123 RFC 822

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).

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

Ngày 2: Biểu diễn thông tin phức hợp

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

Bài 1 – 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}]
```

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

Con trỏ (Pointer)

Nếu bạn nào đã học lập trình C thì sẽ biết đến khái niệm con trỏ. Trong thường hợp bạn nghe khái niệm con trỏ (Pointer) lần đầu thì hiểu như sau: Khi bạn khai báo một biến (variable) thì có nghĩa là máy tính sẽ tạo một vùng nhớ trong thanh RAM và đặt tên vùng nhớ đó dưới dạng một cái tên (tên biến) để cho bạn lưu trữ dữ liệu tạm trong quá trình chương trình thực thi (Xem lại Bài 2).

Thông thường thì bạn chỉ cần biết tên biến và lấy dữ liệu của biến đó, hoặc thiết lập dữ liệu vào biến đó. Tuy nhiên trong vài tình huống đặc biệt thì bạn lại cần truy cập đến địa chỉ của vùng nhớ (memory address). GOLANG cung cấp 2 cú pháp để bạn làm việc với con trỏ:

- *: dấu sao để lấy giá trị của vùng nhớ mà con trỏ đang chỉ đến
- &: để lấy địa chỉ của vùng nhớ (memory address) của biến bình thường (biến không phải con trỏ)

Phân tích đoạn chương trình sau:

```
package main

import "fmt"

func main() {
    i := -10
    j := 25
    pI := &i
    pJ := &j
    fmt.Println("pI memory:", pI)
    fmt.Println("pJ memory:", pJ)
    fmt.Println("pI value:", *pI)
    fmt.Println("pJ value:", *pJ)
}
```

Kết quả:

```
pI memory: 0xc000012090
pJ memory: 0xc000012098
pI value: -10
pJ value: 25
```

Hãy thử phép gán sau và in ra giá trị của biến i sau đó:

```
*pI = 11
```

Con trỏ đến cấu trúc

Phân tích chương trình sau đây:

```
package main

import "fmt"
```

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

```
type aPerson struct {
    Name    string
    Weight  int // kg
    Height  int // cm
}

func changeInfo(p aPerson) {
    p.Height += 1
}

func main() {
    hai := aPerson{"Nguyễn Văn Hải", 74, 168}

    fmt.Println(hai)

    changeInfo(hai)

    fmt.Println("Sau khi gọi hàm changeInfo")
    fmt.Println(hai)
}
```

Kết quả:

```
{Nguyễn Văn Hải 74 168}
```

```
Sau khi gọi hàm changeInfo
```

```
{Nguyễn Văn Hải 74 168}
```

Nhận xét:

- Nếu có nhu cầu viết một hàm để thay đổi giá trị của struct được truyền từ tham số thì cách truyền biến thông thường sẽ không có tác dụng.

Cách xử lý vấn đề trên như sau.

Phân tích chương trình sau:

```
package main

import "fmt"

type aPerson struct {
    Name    string
    Weight  int // kg
    Height  int // cm
}

func changeInfo(p *aPerson) {
    (*p).Height += 1
}

func main() {
    hai := aPerson{"Nguyễn Văn Hải", 74, 168}
```

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

```
fmt.Println(hai)

changeInfo(&hai)

fmt.Println("Sau khi gọi hàm changeInfo")
fmt.Println(hai)
}
```

Kết quả:

```
{Nguyễn Văn Hải 74 168}
Sau khi gọi hàm changeInfo
{Nguyễn Văn Hải 74 169}
```

Chương trình này có một chút cải tiến:

- Sử dụng con trỏ cho tham số trong hàm `changeInfo`:

```
func changeInfo(p *aPerson)
```

- Để thay đổi giá trị của tham số `p` thì dùng cú pháp dấu `*` để lấy ra giá trị của struct. Sau đó thay đổi giá trị của thuộc tính của struct bằng phép gán bình thường:

```
(*p).Height += 1
```

Phép `+=` có nghĩa là cộng cho chính nó. Tức là lệnh trên có nghĩa là cộng thêm cho thuộc tính (property, gọi là biến – variable cũng được) `Height` của struct `aPerson` (mà biến `p` trỏ đến) lên 1 đơn vị.

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 về 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 Ty"
```

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

```
}  
  
func main() {  
    myAge, moonAge := calAge(1977)  
    fmt.Println(myAge, " ", moonAge)  
}
```

43 Đinh Ty

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 là `tuple` (bộ)

Minh họa hàm `strconv.Atoi`

Hàm `Atoi` 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 rằng 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)  
}
```


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

```
}
```

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

Hãy sử dụng 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

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

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.

Sử dụng các hàm thông dụng về String

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

Regular expressions and pattern matching

Tạm dịch mục này là Biểu thức chính qui và so trùng chuỗi. Hơi khó hiểu phải không? Hãy xem nhu cầu sau đây:

Đôi lúc bạn cần tìm kiếm hoặc nhận diện một phần của chuỗi theo một quy tắc nào đó. Các quy tắc được biểu diễn dưới dạng biểu thức gọi là biểu thức chính quy (Regular expression). Ví dụ:

Biểu thức “H\d” ý nói là một chuỗi có dạng H1 hoặc H2 ... hoặc H9. Tức là sau chữ H là một kí số. Kí hiệu \d ý nói là 1 digit character.

Kí tự xuyệt trái (back slash) \ là một kí tự đặc biệt trong chuỗi. Dấu xuyệt này thường được dùng để kết hợp với một kí tự tiếp theo để thể hiện một ý nghĩa đặc biệt nào đó. Ví dụ: \n là biểu diễn kí tự xuống hàng.

Trong thường hợp một chuỗi có nội dung là “\n” tức gồm 2 kí tự là dấu xuyệt và n thì làm sao? Vì nếu viết lệnh `fmt.Println("\n")` thì máy tính sẽ in ra kí tự xuống hàng (kí tự này bạn không thấy bằng mắt mà thực sự là con trỏ đánh dấu chỗ hiển thị kí tự trên màn hình sẽ xuống 1 hàng để chuẩn bị hiển thị nội dung cho các lệnh Print tiếp theo). Để giải quyết tình huống này thì tác giả ngôn ngữ lập trình GO qui ước là dùng thêm một kí tự xuyệt trái nữa. Cụ thể là kí tự ‘\’ được thể hiện trong chuỗi là “\\”. Chuỗi “\n” được biểu diễn là “\\n”

Đoạn chương trình sau đây sử dụng hàm **MatchString** trong thư viện `regexp` để kiểm tra một chuỗi trong biến `st` có xuất hiện chuỗi H0 hoặc H2 ... hoặc H9 không (sau H là một kí số từ 0 đến 9)

```
package main

import (
    "fmt"
    "regexp"
)

func main() {
    st := "H1"
    match, _ := regexp.MatchString("H\\d", st)
    fmt.Println(match)
}
```

Kết quả là biến `match` sẽ trả lại là: `true`

Hãy thử thay đổi biến `st` với các giá trị sau và quan sát kết quả của biến `match`: `aH1`, `H1b`, `aH1b`, `h1`, `ah1`, `ah1b`

Từ đó rút ra nhận xét cho riêng mình.

Hãy đọc tài liệu về Regular Expression trong GO tại trang web:

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

<https://golang.org/pkg/regexp/>

Trong đó hàm MatchString được giải thích như sau:

func MatchString

```
func MatchString(pattern string, s string) (matched bool, err error)
```

MatchString reports whether the string `s` contains any match of the regular expression `pattern`. More complicated queries need to use `Compile` and the full `Regexp` interface.

Hãy đọc và thử thực hành các hàm khác.

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

Bài 2 – Viết hàm cho cấu trúc

Khảo sát đoạn chương trình sau:

```
package main

import (
    "fmt"
)

type Employee struct {
    name    string
    height  int
    weight  int
}

func (e Employee) calculateBMI() float64 {
    heighthMet := float64(e.height) / 100.0
    return float64(e.weight) / (heighthMet * heighthMet)
}

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

    fmt.Println(p1)
    bmi := p1.calculateBMI()

    fmt.Printf("BMI of %s is %f", p1.name, bmi)
}
```

Phân tích hàm calculateBMI cho struct Employee

Chú ý cách gọi hàm tính chỉ số BMI cho biến p1:

```
bmi := p1.calculateBMI()
```

Biến p1 có kiểu là Employee tức diễn đạt theo ngôn ngữ tự nhiên p1 là một Nhân viên có tên là Thạch, cân nặng là 72kg, chiều cao là 1m65. Để tính chỉ số BMI của nhân viên thì gọi hàm calculateBMI của nhân viên đó. Đây chính là tư tưởng của lập trình hướng đối tượng (Object Oriented Programming). Việc gọi hàm chính là gửi thông điệp (send message) cho đối tượng (ở đây là biến p1).

Để lập trình được hàm cho cấu trúc như trên thì bạn chú ý cách viết hàm calculateBMI:

```
func (e Employee) calculateBMI() float64 {
}
```

Cú pháp **func** dùng để định nghĩa hàm như bình thường.

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

Điểm mới ở đây là cú pháp giống như khai báo biến trước tên hàm:

(e Employee)

Chỗ này có nghĩa là biến e có kiểu là Employee.

Phần tiếp theo giống khi khai báo hàm bình thường:

calculateBMI() float64

Tóm lại ý nghĩa của hàm trên diễn đạt như sau: khi báo hàm cho biến mang tính đại diện tên là e có kiểu là cấu trúc Employee; tên hàm là calculateBMI; kết quả hàm trả lại là số thực 64 bit.

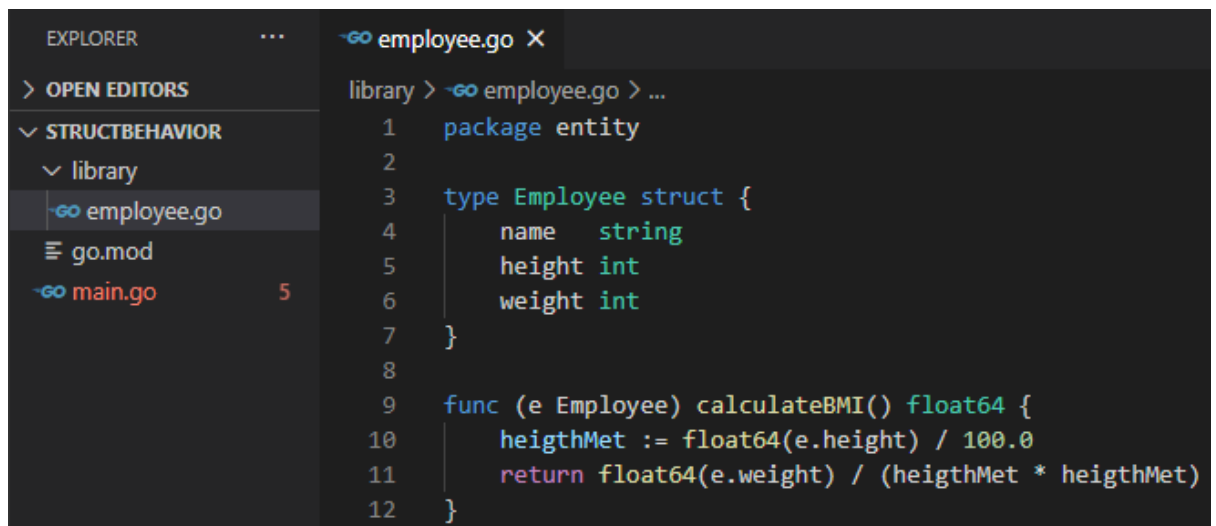
Tổ chức thành thư viện (module)

Tình huống tiếp theo cho bạn là cần tổ chức mã nguồn của cấu trúc Employee thành module để có thể dùng lại.

Đứng trong thư mục của dự án, tạo file go.mod bằng lệnh sau:

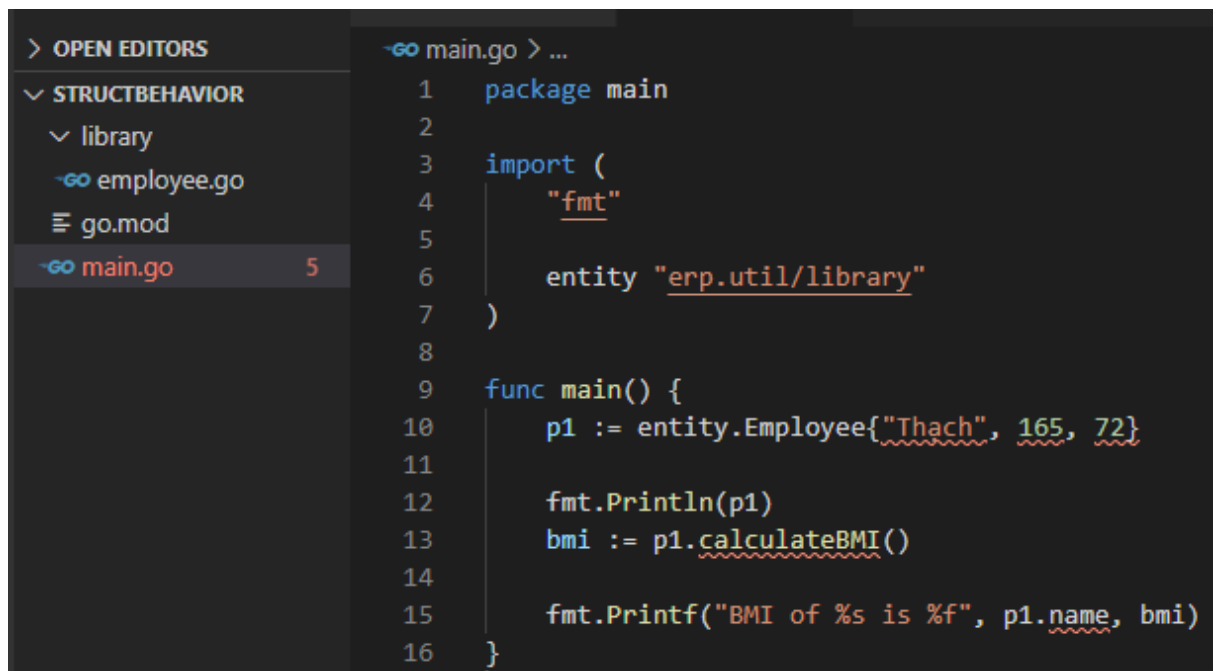
```
go mod init erp.util
```

Tiếp theo tạo thư mục “library” và file library\employee.go. Chuyển mã nguồn của khai báo cấu trúc Employee và hàm calculateBMI từ file main.go vào file library\employee.go như sau:



Mã nguồn của file main.go khai báo sử dụng package “entity” trong thư viện “erp.util/library” như sau:

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



```
1 package main
2
3 import (
4     "fmt"
5
6     entity "erp.util/library"
7 )
8
9 func main() {
10     p1 := entity.Employee{"Thach", 165, 72}
11
12     fmt.Println(p1)
13     bmi := p1.calculateBMI()
14
15     fmt.Printf("BMI of %s is %f", p1.name, bmi)
16 }
```

Về mặt cấu trúc mã nguồn thì tạm ổn. Tuy nhiên bạn sẽ thấy mã nguồn `main.go` bị lỗi ở các dòng 10, 13, 15 trong việc sử dụng cấu trúc và hàm của cấu trúc. Lý do là các biến của cấu trúc và tên hàm bắt đầu bằng **chữ thường**.

Để truy cập được các biến của cấu trúc và tên của hàm thì bạn cần điều chỉnh lại phạm vi (scope) của cấu trúc bằng cách sử dụng ký tự Hoa đầu tiên.

Cụ thể mã nguồn của file `employee.go` được chỉnh lại như sau:

```
package entity

type Employee struct {
    Name    string
    Height  int
    Weight  int
}

func (e Employee) CalculateBMI() float64 {
    heightMet := float64(e.Height) / 100.0
    return float64(e.Weight) / (heightMet * heightMet)
}
```

Chú ý những chỗ viết Hoa và in đậm.

Quy tắc sử dụng ký tự in Hoa đầu tiên cho các biến, tên hàm (gọi chung là **identifer**) trong các module được khái quát lên như sau:

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

Trong GOLANG, để các định danh (tên cấu trúc, tên biến, tên hàm, v.v...) trong một module được truy cập được từ bên ngoài thì cần viết Hoa kí tự đầu tiên.

Code trong file main.go sửa lại như sau:

```
package main

import (
    "fmt"

    entity "erp.util/library"
)

func main() {
    p1 := entity.Employee{"Thạch", 165, 72}

    fmt.Println(p1)
    bmi := p1.CalculateBMI()

    fmt.Printf("BMI of %s is %f", p1.Name, bmi)
}
```


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

Bài 3 – Dữ liệu dạng JSON

JSON là một dạng tài liệu văn bản rất phổ biến để giúp các lập trình viên thực hiện xử lý dữ liệu trong JavaScripts.

Đọc dữ liệu JSON

Ngày 3: Cấu trúc điều khiển

Trong ngày 2, bài 5 bạn đã làm quen với vòng lặp for. Bài này giúp bạn khám phá đầy đủ hơn các cấu trúc điều khiển trong GOLANG

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

Bài 7 – Cấu trúc rẽ nhánh

Lệnh if

```
if <biểu thức điều kiện> {  
    Các lệnh  
}
```

```
if <biểu thức điều kiện> {  
    Các lệnh 1  
} else {  
    Các lệnh 2  
}
```

Switch

Switch <biểu thức> {}

Khảo sát đoạn chương trình sau để hiểu cách dùng switch <biểu thức> để có hướng xử lý cho từng trường hợp giá trị của biểu thức:

```
package main  
  
import (  
    "fmt"  
)  
  
func main() {  
    st := "1234"  
  
    for i := 0; i < len(st); i++ {  
        switch st[i] {  
            case '1':  
                fmt.Println("Một")  
            case '2':  
                fmt.Println("Hai")  
            case '3':  
                fmt.Println("Ba")  
            default:  
                fmt.Printf("%c chưa học", st[i])  
        }  
    }  
}
```

```
}  
}  
}
```

```
Một  
Hai  
Ba  
4  
Chưa học
```

Học được kiến thức mới hoặc ôn tập:

- Để lấy độ dài của chuỗi `st` thì dùng hàm `len(st)`
- Dùng vòng lặp `for` để duyệt qua từng vị trí của các kí tự trong chuỗi `st`.
- Sử dụng cấu trúc `switch case` để xem xét các kí tự, trường hợp là các kí tự '1', '2', '3' thì hiển thị ra màn hình chữ tương ứng. Ngược lại (mặc định) thì hiển thị kí tự đã gặp và kèm chữ "chưa học".
- Sử dụng hàm `Printf` của thư viện `fmt` để hiển thị một câu có lồng giá trị các biến số. Cụ thể là cú pháp `%c` để hiển thị kí tự tương ứng với tham số.

Switch {}

Một cách dùng khác của `switch` không có biểu thức đi kèm.

```
package main  
  
import (  
    "fmt"  
)  
  
func main() {  
    var NUMTEXT = [4]string{"không", "một", "hai", "ba"}  
    st := "12345"  
  
    for i := 0; i < len(st); i++ {  
        switch {  
        case st[i] < '4':  
            fmt.Println(NUMTEXT[st[i]-'0'])  
        case st[i] > '4':  
            fmt.Printf("%c chưa học\n", st[i])  
        default:  
            fmt.Printf("%c đang học\n", st[i])  
        }  
    }  
}
```

```
}  
}
```

Bổ sung vài kiến thức mới:

- Dùng từ khóa `var` để khai báo mảng chứa chữ tương ứng với vị trí của mảng. Việc này thuận lợi cho nhu cầu cần biết chữ tương ứng của số `n` (trong ví dụ này `n` từ 0 đến 3) thì chỉ cần truy xuất `NUMTEXT[n]`.
- Khi `switch` không có biểu thức đi kèm thì biểu thức sau `case` là biểu thức trả lại giá trị đúng hoặc sai.
- Ký tự `\n` trong chuỗi của lệnh `fmt.Printf` là ký tự xuống hàng. Có nghĩa là sau khi hiển thị chuỗi ra màn hình thì con trỏ (cursor) sẽ xuống hàng để sẵn sàng cho các lệnh `fmt.Print...` tiếp theo.

Bài 8 – Vòng lặp

Vòng lặp (loops)

Vòng lặp for

Cấu trúc vòng lặp

```
for A; B; C {  
    }
```

A là biểu thức sẽ được thực hiện khi khởi tạo vòng lặp. Ví dụ phép gán chỉ số vòng lặp i.

B là biểu thức điều kiện, nếu B đúng thì vòng lặp được thực hiện. Ngược lại, vòng lặp sẽ kết thúc.

C là biểu thức sẽ được thực hiện khi kết thúc một bước lặp và chuẩn bị bước tiếp theo. Bước tiếp theo được hiểu là điều kiện B sẽ được kiểm tra. Nếu B đúng thì đoạn code trong vòng lặp được thực hiện.

```
for i := 0; i < 10; i++ {  
    }
```

Lệnh break trong vòng for

Khảo sát đoạn code sau để biết cách dùng lệnh `break` để thoát vòng lặp `for`:

```
package main  
  
import (  
    "fmt"  
)  
  
func main() {  
    var n int  
  
    for i := 0; i < 10; i++ {  
        n = i * 2  
        fmt.Println("i x 2 = ", n)  
        if n > 40 {  
            break  
        }  
    }
```

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

```
}  
}
```

Ở đây bạn học thêm lệnh `if`. Nếu $n > 40$ thì sẽ dừng vòng lặp bằng lệnh `break`.

Nếu bạn là người đã quen ngôn ngữ lập trình C, C++ hay Java thì phát hiện ra một điều mới là khi viết biểu thức `if` hoặc `for` thì không cần cặp dấu ngoặc (). Còn nếu bạn chưa biết các ngôn ngữ kia thì đáng mừng vì không phải suy nghĩ về ý này, coi như tôi chưa nói đoạn này với bạn.

Lệnh continue trong vòng for

Khám phá đoạn code sau để hiểu lệnh `continue` nhé.

```
package main  
  
import (  
    "fmt"  
)  
  
func main() {  
    var n int  
  
    for i := 0; i < 10; i++ {  
        if i%2 == 0 {  
            continue  
        }  
  
        n = i * 2  
        fmt.Println(i, " x 2 = ", n)  
  
        if n > 40 {  
            break  
        }  
    }  
}
```

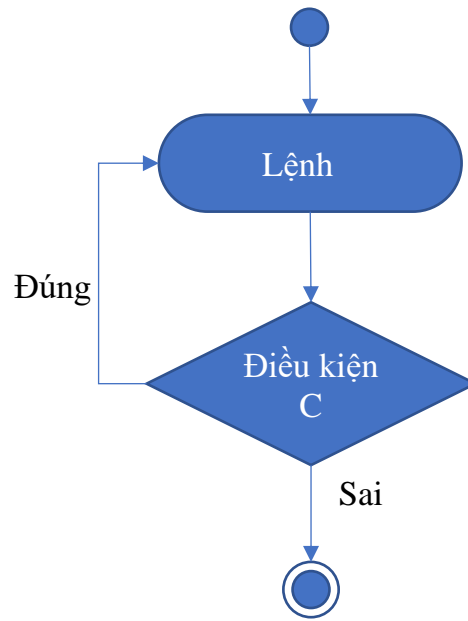
Đây là eBook của riêng bạn – đề nghị không chia sẻ cho ai khác nhé!

Vòng lặp for nâng cao

Nếu các bạn đã biết lập trình cơ bản thì chắc có nghe tới vòng lặp **do...while** và **while ... do**.

Vòng lặp do..while

Ý nghĩ **do...while** là cần lặp lại công việc nào đó nhiều lần theo sơ đồ sau:



Tức là lệnh (hoặc nhiều lệnh được thực hiện. Sau đó kiểm tra điều kiện (biểu thức C) được thiết lập. Nếu C là đúng thì các lệnh trong vòng lặp được thực hiện tiếp. Ngược lại nếu C là sai thì sẽ dừng vòng lặp.

Để triển khai ý tưởng này trong GOLANG thì phân tích chương trình sau:

```
package main

import "fmt"

func main() {
    fmt.Println("Use for loop as do...while")
    n := 0
    for ok := true; ok; ok = n < 10 {
        n = n*2 + 1
        fmt.Println(n)
    }
}
```

Để ý một chút thì thấy các biểu thức A B C trong vòng lặp for

```
for A; B; C {
}
```

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

được tổ chức lại như sau:

A là biểu thức (lệnh) gán true cho biến ok. Vừa gán vừa khởi tạo biến ok nên dùng cú pháp hai chấm bằng. Lệnh này được thực hiện một lần khi bắt đầu vòng lặp for.

B là biểu thức ok. Tức là biến ok có kiểu true/false. ok là đúng thì vòng lặp được thực hiện. Tức là các lệnh bên trong cặp dấu {} được thực thi.

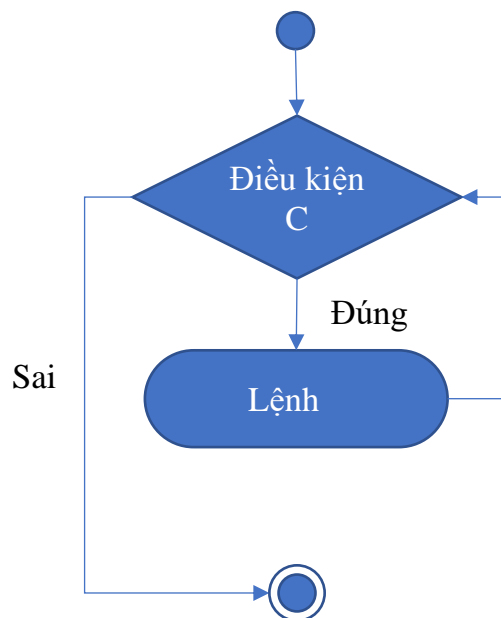
C là biểu thức (lệnh) gán kết quả so sánh $n < 10$ cho ok. Lúc này chỉ thay đổi giá trị của biến ok nên dùng dấu bằng. Lệnh này được thực hiện cuối vòng for. Tức là sau khi các lệnh bên trong cặp dấu {} được thực hiện thì biến ok được tính toán lại với biểu thức điều kiện $n < 10$.

Khi bắt đầu lần lặp tiếp theo thì biểu thức B, tức là biến ok được kiểm tra trước khi các lệnh bên trong cặp dấu {} được thực hiện. Nếu ok là false thì dừng vòng lặp.

Hơi dài dòng một ít dành cho các bạn chưa quen lập trình!

Vòng lặp while..do

Tình huống đặt ra cho các bạn là với sơ đồ vòng lặp do...while ở trên, bạn muốn thay đổi ý tưởng là cần phải kiểm tra biểu thức điều kiện trước rồi hãy thực hiện lệnh. Ý tưởng sơ đồ như sau:



Tức là vòng lặp thì kiểm tra điều kiện trước. Nếu đúng thì hãy thực hiện lệnh (hoặc các lệnh) bên trong vòng lặp. Sau đó kiểm tra tiếp điều kiện. Nếu điều kiện có giá trị là sai thì kết thúc vòng lặp.

Hãy chạy thử và phân tích kết quả, đối chiếu với các dòng lệnh trong chương trình sau:

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

```
package main

import "fmt"

func main() {
    fmt.Println("Use for loop as while...do")
    n := 0
    for n < 10 {
        n = n*2 + 1
        fmt.Println(n)
    }
}
```

Quan sát vòng lặp for chỉ có một biểu thức điều kiện đơn giản $n < 10$. Tức là cấu trúc tổng quát của vòng lặp for:

```
for A; B; C {
}
```

được tối giản còn

```
for B {
}
```

Trong ví dụ trên B là biểu thức $n < 10$.

Nếu diễn giải bằng lời thì như sau: Trong khi $n < 10$ thì thực hiện các lệnh sau...

Đến đây bạn đọc cảm nhận được chỉ với một cú pháp **for** thì GOLANG có cách giúp bạn triển khai được 3 nhóm ý tưởng mà các ngôn ngữ lập trình khác cần phải có 3 bộ cú pháp riêng như:

- ① Cần lặp lần lượt theo một chỉ số
- ② Cần lặp lại một việc nào đó rồi kiểm tra biểu thức
- ③ Cần kiểm tra biểu thức để lặp một việc nào đó

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

Bổ sung extension cho Visual Code

