کتاب‌خانه‌ی پردردسر

برای مدیریت بهتر کتاب‌هایمان به یک کتاب‌خانه نیاز داریم. این کتاب‌خانه مقداری ظرفیت دارد که بیشتر از آن نمی‌تواند کتاب در خود جا دهد.

این کتاب‌خانه مشخصات زیر را دارد:

- از هر کتاب دقیقاً یکی دارد که یا موجود است و یا به یک نفر امانت داده شده است.
- اسم کتاب‌ها به بزرگی و کوچکی حساس نیست، مثلاً `Golestan` و `gOlEsTan` دقیقاً یک کتاب هستند.
- یک نفر می‌تواند چند کتاب را امانت بگیرد و از این نظر محدودیتی نیست.
- در صورتی که یک کتاب امانت گرفته شده باشد، جایش همچنان محفوظ است و ظرفیت آن برای اضافه شدن کتاب دیگری باز نمی‌شود.
- پیام‌های خطای زیر از متدهای مختلف کتاب‌خانه دریافت می‌شود. در صورتی که هیچ خطایی نباشد نیز `OK` داده می‌شود:

```
The book has not been borrowed
The book is already borrowed by <borrower name>
The book is already in the library
The book is not defined in the library
Not enough capacity
OK
```

تشخیص این‌که کدام ارور در چه زمان(ها)ی باید برگردانده شود بر عهده‌ی شماست.

جزئیات پروژه

پروژه‌ی اولیه را از این لینک دانلود کنید. ساختار فایل‌های پروژه به صورت زیر است:

```
library/
├── go.mod
├── go.sum
├── main.go
└── main_sample_test.go
```

در فایل `main.go` چند تابع و متد وجود دارد که شما باید آن‌ها را کامل کنید.

تابع `NewLibrary`

این تابع یک اشاره‌گر به یک شی از نوع کتاب‌خانه برمی‌گرداند. در این تابع باید یک شی جدید بسازید و ساختار داخلی آن را مطابق آن‌چه خودتان تعریف کرده‌اید مقداردهی کنید.

main.go

```go
type Library struct {
}

func NewLibrary(capacity int) *Library {
    // TODO
    return nil
}
```

متد `AddBook`

در این متد شما اسم یک کتاب را گرفته و در صورتی که شرایط برقرار بود (مثلاً کتابخانه جا داشت و این کتاب قبلا در کتابخانه نبود) آن را اضافه می‌کنید و `OK` بر می‌گردانید. در غیر این صورت بسته به مورد ارور مناسب را در قالب یک رشته برمی‌گردانید.

main.go

```go
func (library *Library) AddBook(name string) string {
    // TODO
    return ""
}
```

راهنمایی: در صورتی که کتاب‌خانه پر باشد و کتابی با نام تکراری وارد شود، خطای مربوط به کتاب تکراری (و نه پر بودن ظرفیت) باید برگردانده شود.

متد `BorrowBook` و `ReturnBook`

در این دو متد یک نفر یک کتاب را امانت می‌گیرد و سپس آن را پس می‌دهد.

در مواردی نیز ممکن است خطا پیش آید مثلا کتابی که امانت گرفته شده را دوباره کسی بخواهد امانت بگیرد یا کسی کتابی که در حال حاضر امانت گرفته نشده را بخواهد پس دهد.

main.go

```go
func (library *Library) BorrowBook(bookName, personName string) string {
    // TODO
    return ""
}

func (library *Library) ReturnBook(bookName string) string {
    // TODO
    return ""
}
```

آن‌چه باید آپلود کنید

پس از پیاده‌سازی توابع خواسته شده، فایل `main.go` را آپلود کنید.

یکی در میون

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

قلی در شرکت اسنپ مشغول یادگیری اصول برنامه‌نویسی هم‌روند در زبان Go است. او به یک مشکل هم‌روندی برخورده که نیاز به کمک شما دارد.

آن‌چه باید پیاده‌سازی کنید

پروژه‌ی اولیه را از این لینک دانلود کنید. ساختار فایل‌های پروژه به صورت زیر است:

```
.
├── go.mod
├── main.go
├── utils.go
└── main_sample_test.go
```

در فایل `main.go` کدی به شما داده می‌شود که به شکل زیر است:

```go
type TikTak struct {
    n int
}

func NewTikTak(n int) *TikTak {
    return &TikTak{
        n: n,
    }
}

func (t *TikTak) Tik() {
    // TODO
}

func (t *TikTak) Tak() {
    // TODO
}
```

در تست‌ها، یک نمونه از `TikTak` به دو گوروتین مختلف پاس داده می‌شود:

- گوروتین `A` متد `Tick()` را فراخوانی می‌کند.
- گوروتین `B` متد `Tak()` را فراخوانی می‌کند.

وظیفه شما این است که برنامه داده‌شده را تغییر دهید تا خروجی آن `n` بار به صورت `TikTak` چاپ شود.

نکته مهم: برای چاپ کردن عبارت‌ها باید حتما از توابع داخل فایل `utils.go` استفاده کنید.

مثال

در ازای `n = 2` خروجی برنامه باید به این شکل باشد:

```
TikTakTikTak
```

و در ازای `n = 3` :

```
TikTakTikTakTikTak
```

نکته: برای درک دقیق منطق سوال به فایل `main_sample_test.go` رجوع کنید.

آن‌چه باید آپلود کنید

یک فایل `main.go` که شامل پیاده‌سازی کامل ساختارها و توابع ذکر شده باشد. حواستان باشد فقط فایل `main.go` را آپلود کنید وگرنه در روند تست‌ها تداخل ایجاد می‌شود.

ایرپاد قلی

قلی به تازگی ایرپاد خریده و هرجا که می‌رود شروع می‌کند به پز دادن. آنقدر در شرکت درباره‌ی ایرپادش حرف زده که رئیسش دیگر از هرچی ایرپاد است متنفر شده! یک روز رئیس تصمیم می‌گیرد تا یک مسئله درباره‌ی ایرپاد به قلی بدهد تا قلی هم از ایرپاد متنفر شود، شاید کمتر پز بدهد. برای آنکه شما هم از ایرپاد متنفر شوید، قلی تصمیم گرفت که حل این سوال را به شما بسپارد.

در این سوال یک ایرپادکیس داریم که دو تا ایرپاد دارد. هر ایرپاد می‌تواند سه وضعیت داشته باشند :

- `Docked` : زمانی که ایرپاد داخل ایرپادکیس قرار دارد.
- `Connected` : زمانی که ایرپاد داخل ایرپادکیس قرار ندارد و به گوشی وصل است.
- `Disconnected` : زمانی که ایرپاد داخل ایرپادکیس قرار ندارد و به گوشی وصل نیست.

فرض کنید ساختار ایرپاد و ایرپادکیس توسط ما ساخته می‌شود و دیتا به آن از طریق یک گوشی موبایل ارسال می‌شود. همچنین انتظار می‌رود در زمانی که ایرپاد متصل است، صدا را در قالب بایت‌ها پخش کند و ما به عنوان انسان آن را می‌شنویم.

آن‌چه باید پیاده‌سازی کنید

در این قسمت ساختارهایی که باید پیاده سازی کنید را مشاهده می‌کنید:

```
type Airpod struct {
}

type AirpodCase struct {
}
```

استراکت `Airpod` : نمایانگر ساختار یک ایرپاد است که یک نمونه از آن داخل هر گوش قرار می‌گیرد.

استراکت `AirpodCase` : در آن ایرپادهای گوش راست و چپ قرار می‌گیرد و وظیفه متصل کردن ایرپادها به گوشی را دارد.

در این قسمت توابعی که باید پیاده سازی کنید را مشاهده می‌کنید که در ادامه به توضیح آن می‌پردازیم:

```
func NewAirpodCase() *AirpodCase {
}
func (a *AirpodCase) GetRightAirpod() *Airpod {
}

func (a *AirpodCase) GetLeftAirpod() *Airpod {
}

func (a *Airpod) GetState() string {
}

func (a *AirpodCase) UndockLeft() *Airpod {
}

func (a *AirpodCase) UndockRight() *Airpod {
}

func (a *AirpodCase) DockLeft() error {
}

func (a *AirpodCase) DockRight() error {
}

func (a *Airpod) GetChannel() chan byte {
}

func (c *AirpodCase) ConnectBluetooth(ch chan byte) error {
}
```

متد NewAirpodCase

استراکت AirpodCase را می‌توانید به طور دلخواه تعریف کنید و در این متد فیلدهای آن را مقداردهی کنید و باید یک نمونه پوینتری از AirpodCase را برگردانید.

متد GetRightAirpod

این متد که روی ایرپاد کیس تعریف شده است، شی ایرپاد راست را برمی‌گرداند. (این متد تنها شی ایرپاد راست را برمی‌گرداند و به معنی خارج کردن ایرپاد راست از ایرپادکیس نیست.)

متد GetLeftAirpod

این متد که روی ایرپاد کیس تعریف شده است، شی ایرپاد چپ را برمی‌گرداند. (این متد تنها شی ایرپاد چپ را برمی‌گرداند و به معنی خارج کردن ایرپاد چپ از ایرپادکیس نیست.)

متد GetState

این متد که روی ایرپاد تعریف شده است، وضعیت ایرپاد را در آن لحظه به صورت یک رشته برمی‌گرداند. (وضعیت یک ایرپاد می‌تواند `Connected` یا `Disconnected` یا `Docked` باشد.)

متد UndockLeft

این متد که روی ایرپاد کیس تعریف شده است، ایرپاد چپ را از ایرپادکیس خارج می‌کند و شی ایرپاد چپ را برمی‌گرداند. این خارج کردن به معنی تغییر وضعیت از `Docked` به `Connected` یا `Disconnected` است. اگر قبلا `Undock` شده باشد، وضعیتش تغییر نمی‌کند و `nil` برمی‌گردد.

متد UndockRight

این متد که روی ایرپاد کیس تعریف شده است، ایرپاد راست را از ایرپادکیس خارج می‌کند و شی ایرپاد راست را برمی‌گرداند. این خارج کردن به معنی تغییر وضعیت از `Docked` به `Connected` یا `Disconnected` است. اگر قبلا `Undock` شده باشد، وضعیتش تغییر نمی‌کند و `nil` برمی‌گردد.

متد DockLeft

این متد که روی ایرپاد کیس تعریف شده است، ایرپاد چپ را داخل ایرپادکیس قرار می‌دهد و باید وضعیت ایرپاد از `Connected` یا `Disconnected` به `Docked` تغییر کند. اگر ایرپاد چپ داخل ایرپادکیس باشد، باید ارور برگرداند و در غیر این صورت `nil` برگرداند.

متد DockRight

این متد که روی ایرپاد کیس تعریف شده است، ایرپاد راست را داخل ایرپادکیس قرار می‌دهد و باید وضعیت ایرپاد از `Connected` یا `Disconnected` به `Docked` تغییر کند. اگر ایرپاد راست داخل ایرپادکیس باشد، باید ارور برگرداند و در غیر این صورت `nil` برگرداند.

متد GetChannel

این متد که روی ایرپاد تعریف شده است، باید چنلی از جنس *byte* برگرداند. این چنل مانند اسپیکر ایرپاد است و هر چیزی که قرار باشد اسپیکر پخش کند، باید از طریق این چنل خروجی داده شود. تضمین می‌شود که از بیرون چیزی داخل این چنل نوشته نمی‌شود.

متد ConnectBluetooth

این متد که روی ایرپاد کیس تعریف شده است، این متد وظیفه متصل کردن گوشی موبایل به ایرپادکیس ما را دارد، برای اینکار یک چنل از جنس بایت دریافت می‌کند که ارتباط گوشی و کیس از طریق این چنل صورت می‌گیرد.

توجه کنید که به این چنل فقط از گوشی اطلاعات فرستاده می‌شود و از داخل خوانده می‌شود.

همچنین این تابع وظیفه تغییر دادن وضعیت ایرپادها را نیز دارد، به این صورت که در زمان اتصال ایرپادهایی که در وضعیت `Disconnected` قرار دارند را به `Connected` تغییر دهد. همچنین اگر ایرپادی بعد از اجرا شدن این متد *Undock* شود باید خود به خود اتصالش با گوشی برقرار شود.

خروجی این تابع یک ارور می‌باشد که در صورتی که اتصال برقرار باشد و دوباره برای اتصال تلاش کنیم بر می‌گردد.

نکات بیشتر

- انتقال داده فقط برای داده‌هایی انجام می‌شود که بعد از اتصال ایرپاد فرستاده می‌شوند.
- `undock` شدن می‌تواند قبل از اتصال و بعد از اتصال باشد و در هر دو حالت اتصال باید به درستی انجام شود.
- ممکن است فقط یکی از دو گوشی ایرپاد متصل باشد و در این صورت فقط دیتا به همان ایرپاد فرستاده می‌شود.
- با داک شدن دوباره گوشی بعد از اتصال دیگر داده‌ای در چنل مربوطه ذخیره نمی‌شود (تا زمانی که دوباره آنداک شود).
- به صورت کلی گوشی ارتباط خود را با کیس قطع نمی‌کند و بعد از اتصال دیگر ارتباط خود را با کیس قطع نمی‌کند و فقط ایرپادها داک و آنداک می‌شوند.
- متد `Dock` و `Undock` ممکن است به صورت همروند (*Concurrent*) تست شود.

جزئیات پروژه

پروژه‌ی اولیه را از این لینک دانلود کنید. ساختار فایل‌های پروژه به صورت زیر است:

```
.
├── go.mod
├── go.sum
├── main.go
└── main_sample_test.go
```

در فایل `main.go` چند تابع و متد وجود دارد که شما باید آن‌ها را کامل کنید.

آن‌چه باید آپلود کنید

پس از پیاده‌سازی توابع خواسته شده، فایل `main.go` را آپلود کنید.

لاگِ بی باگ

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

قلی در شرکت اسنپ تازه‌وارد است و هنوز نمی‌تواند تسک‌های سنگین را پیاده‌سازی کند. بنابراین فعلا به او مسئولیت‌های متفرقه می‌دهند، البته قلی با مهارت‌های برنامه‌نویسی خود سعی می‌کند این تسک‌ها را ساده‌تر کند. این‌بار از قلی خواسته شده است که لاگ‌های بخشی از سیستم که لاگر آن به تازگی نوشته شده است را زیر نظر بگیرد تا بتواند باگ‌های آن را شناسایی کند. وظایف او به شرح زیر است:

۱. تمام فایل‌های لاگ در یک دایرکتوری مشخص را بخواند.

۲. تعداد پیام‌های لاگ بر اساس سطح آن `ERROR` ، `INFO` ، `WARNING` را محاسبه کند.

۳. به ازای هر فایل، تعداد لاگ‌های با باگ (لاگ‌هایی که به درستی ثبت نشده‌اند) را محاسبه کند.

نکته: جمله‌ی پایین نشان‌دهنده‌ی یک لاگ بی باگ است. لاگ با باگ به لاگی گفته می‌شود که یک یا چند مورد از ساختار زیر را ندارد:

```
WARNING | 2024-08-24 19:27:05 | env variable is not set
```

همانطور که می‌بینید یک لاگ بی‌باگ از چهار بخش تشکیل شده: سطح، تاریخ، زمان و پیام لاگ. و همیشه با یکی از کلمات `WARNING` ، `INFO` ، `ERROR` آغاز می‌شود.

۴. قلی باید وابستگی بین فایل‌ها (در واقع تقدم و تاخر آن‌ها) را در نظر بگیرد. برخی از فایل‌های لاگ ممکن است با یک خط خاص آغاز شوند که به شکل زیر است:

```
...anotherLogFile.txt
```

این بدان معناست که باید ابتدا فایل اشاره‌شده (`anotherLogFile.txt`) پردازش شود و سپس به پردازش فایل فعلی ادامه داده شود. این باعث ایجاد یک وابستگی (`dependency tree`) میان فایل‌های لاگ می‌شود که باید به درستی مدیریت شود تا ترتیب لاگ‌ها حفظ شود. لازم به ذکر است که این وابستگی تنها بین دو فایل وجود دارد (رابطه یک به یک). به عبارت دیگر: یک فایل نمی‌تواند وابسته به چند فایل باشد و یک فایل نمی‌تواند لازمه‌ی چند فایل باشد.

۵. قلی باید بتواند پردازش لاگ‌ها را به صورت هم‌زمان `concurrently` یا ترتیبی `sequentially` انجام دهد.

۶. در انتها باید یک فایل به نام `output.txt` ایجاد شود و همه‌ی لاگ‌های بی باگ که در سطح مشخص شده هستند را به ترتیب درست (با توجه به وابستگی‌ها) در آن چاپ کند.

۷. در انتهای فایل `output.txt` باید نام هر فایل لاگ به همراه تعداد لاگ‌های با باگ در آن به شکل زیر چاپ شود:

نکته: ترتیب نوشته شدن تعداد باگ به ازای هر فایل در این بخش از خروجی اهمیتی ندارد.

```
...
log1.txt: 2 bugs
log2.txt: 0 bugs
log3.txt: 1 bugs
...
```

مثال

ساختار فولدر لاگ‌ها:

```
logs
   ├── log1.txt
   ├── log2.txt
   ├── log3.txt
   ...
```

نکته: اسم فایل‌های لاگ می‌تواند متفاوت باشد و لزوماً با کلمه `log` شروع نمی‌شود.

محتویات هر فایل:

```
log1.txt
    ...log2.txt
    INFO | 2024-08-24 19:27:05 | Application started

log2.txt
    WARNING | 2024-08-24 19:27:07 | High memory usage
    ERROR | 2024-08-24 | Application failed

log3.txt
    ERROR | 2024-08-24 19:27:06 | Failed to connect to database
```

خروجی صحیح در فایل `output.txt` :

```
WARNING | 2024-08-24 19:27:07 | High memory usage
INFO | 2024-08-24 19:27:05 | Application started
ERROR | 2024-08-24 19:27:06 | Failed to connect to database
log1.txt: 0 bugs
log2.txt: 1 bugs
log3.txt: 0 bugs
```

لطفاً به قلی کمک کنید. :)

آن‌چه باید پیاده‌سازی کنید

پروژه‌ی اولیه را از این لینک دانلود کنید. ساختار فایل‌های پروژه به صورت زیر است:

```
.
├── go.mod
├── go.sum
├── main.go
└── main_sample_test.go
```

شما باید فایل `main.go` را تکمیل کنید.

ساختارها و متدهایی که باید پیاده‌سازی کنید:

```go
type LogAnalyzer struct {
    // TODO
}
```

شما باید یک ساختار `LogAnalyzer` ایجاد کنید که تمامی داده‌های لازم برای پردازش لاگ‌ها را نگهداری کند. می‌توانید هر فیلدی به این استراکت اضافه کنید.

```go
func NewLogAnalyzer(
    severity string,
    outputPath string,
    logsDir string,
) LogAnalyzerInterface {
     //TODO
}
```

این تابع یک نمونه از `LogAnalyzer` را مقداردهی اولیه می‌کند و برمی‌گرداند.

ورودی اول این تابع سطح شدت (`severity`) است که برای تحلیل لاگ‌ها استفاده می‌شود. برنامه‌ی ما باید **لاگ‌هایی با این سطح را جمع‌آوری کند** و در فایل خروجی چاپ کند. ورودی دوم `outputPath` مسیر فایل خروجی را مشخص می‌کند. ورودی سوم `logsDir` آدرس فولدر لاگ‌ها را مشخص می‌کند.

نکته مهم: متدهای زیر را باید به شکل `receiver` برای `LogAnalyzer` پیاده‌سازی کنید. نکته: امضای هیچ‌کدام از متدها را عوض نکنید، در این صورت تست‌ها پاس نمی‌شوند.

```go
Worker()
```

این متد باید فایل‌های لاگ را بخواند و بر اساس نوع پیام‌های موجود در آن‌ها، شمارنده‌های مربوطه را به‌روزرسانی کند. همچنین باید به وابستگی‌ها توجه کند. اگر فایلی وابسته به فایل دیگری است، باید ابتدا فایل وابسته پردازش شود.

```go
ConcurrentWorker(filePath string)
```

این متد مشابه `Worker` عمل می‌کند، اما باید به صورت هم‌زمان اجرا شود. بنابراین این متد باید بتواند به صورت `concurrent` کار کند و وابستگی‌های بین فایل‌ها را نیز مدیریت کند. چون این تابع قرار است به صورت هم‌روند کار کند پس به عنوان ورودی آدرس یک فایل لاگ را می‌گیرد، تا بتوانیم هنگام اجرا تعداد زیادی از آن را باگوروتین‌ها اجرا کنیم.

نکته: دقت کنید که متد `Worker` به صورت ترتیبی و متد `ConcurrentWorker` به صورت کانکارنت تست خواهند شد.

نکته: در تست‌های `sequentially` ترتیب لاگ‌های داخل یک فایل اهمیت دارد، اما در تست‌های `concurrent` این ترتیب اهمیت ندارد و فقط ترتیب وابستگی‌ها تست می‌شود.

نکات بیشتر و تضمین‌ها

- به دلیل وجود وابستگی بین فایل‌ها، باید توجه کنید که فایل‌ها به ترتیب درست و با توجه به وابستگی‌هایشان پردازش شوند.
- تضمین می‌شود که هیچ فایلی وابسته به دو فایل نیست و می‌تواند تنها به یک فایل دیگر وابسته باشد، و برعکس، یعنی هیچ فایلی نیست که دو فایل به آن وابستگی داشته باشند.
- پس از پردازش، تمامی لاگ‌ها با سطحی که مشخص شده، باید به ترتیب صحیح در یک فایل به نام `output.txt` ذخیره شوند.
- می‌توانید از تست‌های فایل `main_sample_test.go` استفاده کنید. توصیه می‌کنیم به همین روش برای خود تست‌های بیشتری بنویسید و برنامه خود را در حالت‌های مختلف تست کنید. همچنین با خواندن تست کیس‌ها می‌توانید مسئله را بهتر درک کنید.
- برای پردازش لاگ‌ها می‌توانید از `regular expressions` استفاده کنید.
- تضمین می‌شود که فرمت لاگ بی‌باگ همیشه به همان صورتی که توصیف شد خواهد بود، فرمت تاریخ و زمان نیز همیشه به این صورت خواهد آمد: `2024-08-24 19:27:05` .
- تضمین می‌شود که در وابستگی بین فایل‌ها هرگز حلقه رخ نخواهد داد. مثلاً اگر فایل A به فایل B وابسته است، فایل B به فایل A وابسته نخواهد بود.
- تضمین می‌شود که دایرکتوری `logsDir` وجود دارد و در آن حداقل یک فایل لاگ قرار دارد.

آن‌چه باید آپلود کنید

یک فایل `main.go` که شامل پیاده‌سازی کامل ساختارها و توابع ذکر شده باشد.

ردیاب طلا (امتیازی)

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۱۰۲۴ مگابایت

اصغر که به تازگی آموزش گولنگ را شروع کرده، از سختی این زبان جدید می‌نالد و امیدی به درآمدزایی از این راه ندارد. بنابراین او در وسایل قدیمی انباری خانه پدربزرگش یک نقشه گنج و یک ردیاب طلا یافته است. گنج صندوقی پر از طلا است. زمینی در همان حوالی خانه پدربزرگ او است که نقشه به همان زمین اشاره دارد. زمینی که گنج در آن واقع است، یک مستطیل است که هر بخش آن یک متر در یک متر است. از آن جا که کسی که برای یادگیری گولنگ تلاش نمی‌کند، حتی برای یافتن گنج هم تلاشی نمیکند (گنج اصلی همان گولنگ بود که رها کرد!!). پس از شما میخواهیم این گنج را با استفاده از ردیاب طلا برای اصغر پیدا کنید.

مشکل اما به همینجا ختم نمی‌شود. ردیاب قدیمی است و محدودیت‌هایی دارد. ردیاب یک دکمه دارد که با زدن آن صدایی نسبت به موقعیت قبلی خودش تولید می‌کند. این صدا با توجه به نزدیک یا دور شدن از طلا، تغییر می‌کند.

ما می‌توانیم دستگاه را به هر نقطه از زمین برده، و با گزارش موقعیت مکانی خود به صورت دو عدد صحیح اعلام وضعیت نزدیک یا دور شدن از طلا را از ردیاب بگیریم.

اگر آن نقطه، همان نقطه گنج باشد تست بسرعت pass میشود و شما موفق به پیدا کردن گنج شده‌اید. اگر مختصات گزارش شده اشتباه باشد، دستگاه خراب می‌شود (تست بسرعت fail می‌شود).

آن‌چه باید پیاده‌سازی کنید

شما باید یک تابع پیاده سازی کنید که در آرگومان‌های ورودی آن یک چنل دریافت کرده اید. این چنل راه ارتباط دوطرفه شما با دستگاه ردیاب است.

```
func FindTreasure(ch chan interface{}) {
    // TODO
}
```

دستگاه یک رفتار خاص در چنل خود دارد که شما موظف به رعایت آن هستید. اگر به درستی ساختار کار با چنل را رعایت نکنید، به دلیل بافر نشدن چنل، در یکی از خطوط تست یا برنامه شما، برنامه می‌ایستد و تست fail می‌شود.

در ابتدا که چنل را دریافت می‌کنید، سه عدد صحیح از تایپ `int` در آن ریخته می‌شود. اولین عدد `maxQuery` که حداکثر تعداد پرسش شما از دستگاه است را تعیین میکند. دومین عدد `n` که تعداد سطر زمین و سومین عدد `m` که تعداد ستون زمین است، به شما داده می‌شود.

$$1 \leq n, m \leq 10000$$

$$1 \leq maxQuery \leq n * m$$

مختصات خانه های جدول در بازه زیر میباشد:

$$1 \leq i \leq n$$

$$1 \leq j \leq m$$

```
maxQueryInterface := <-ch
nInterface := <-ch
mInterface := <-ch
```

بعد از آن شما با دادن مختصات هر نقطه از زمین، می‌توانید اعلام وضعیت دستگاه را نسبت به موقعیت قبلی از آن بگیرید (هر اعلام وضعیت یک پرسش محسوب می‌شود که حداکثر تعداد پرسش مجاز را در ابتدا از چنل خوانده‌اید). پس از هر پرسش از طریق چنل دستگاه باید منتظر پاسخ دستگاه باشید:

```
ch <- i
ch <- j
answerInterface := <-ch
```

پاسخ دستگاه چهار حالت ممکن دارد:

۱. در اولین پرسش از دستگاه همواره در پاسخ عبارت `Go!!` می‌آید.

۲. اگر موقعیت جدید گزارش شده از موقعیت قبلی نزدیک تر به گنج باشد عبارت `$s$s$` در پاسخ برگردانده می‌شود.

۳. اگر موقعیت جدید گزارش شده از موقعیت قبلی دورتر به گنج باشد عبارت `s.s.s` در پاسخ برگردانده می‌شود.

۴. اگر موقعیت جدید گزارش شده فاصله مشابه نسبت به موقعیت قبلی داشته باشد عبارت `same as before` در پاسخ برگردانده می‌شود.

نکات و تضمین ها

۱. تضمین میشود حداقل تعداد پرسش مورد نیاز برای یافتن گنج به شما داده می‌شود.

آن‌چه باید آپلود کنید

پروژه‌ی اولیه را از این لینک دانلود کنید. یک فایل `main.go` آپلود کنید که در آن تابع گفته شده به شکل مناسب پیاده‌سازی شود (امضا تابع نباید تغییر کند).