Piscine Golang - Rush00

Created @October 30, 2021 12:56 PM

요약 : 이 문서는 cgo, os/exec, http 모듈학습을 다룹니다.

서브젝트 오류 문의 : bigpel66@icloud.com

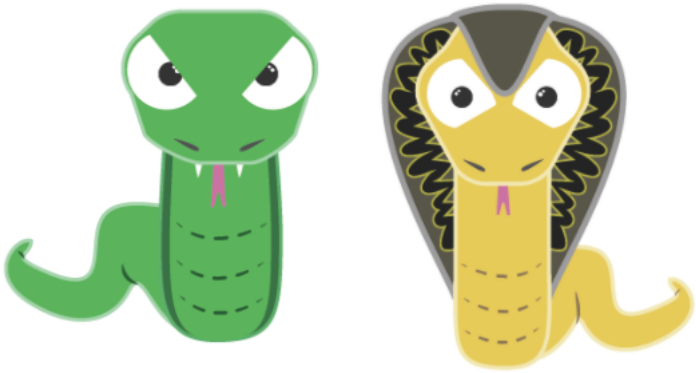
Contents

Chapter I - General Rules

- 모든 Exercise들은 Golang 1.17로 진행하며 테스트 코드와 함께 작성되어야 합니다. 작성된 테스트 코드들은 Exercise와 함께 제출되어야 합니다.
- 제출된 테스트 코드 내에 테스트 케이스는 반드시 존재해야 합니다.
- 적극적인 테스트 케이스는 권장되며, 특히 예외 케이스에 신경써야 합니다.
- 평가자는 자신이 생각했던 테스트 케이스가 존재하지 않는 경우, 피평가자에게 테스트 케이스를 요청할 수 있습니다. 이 때 피평가자의 코드가 동작하지 않으면 평가는 종료됩니다.
- 테스트 진행은 "testing" 패키지를 이용해야 하며, "testing"과 "github.com/stretchr/testify/assert" 둘 중 하나를 일관성 있게 사용해야 합니다.
- 테스트 진행 시 testing 객체를 이용한 벤치마킹 역시 적극 권장됩니다.
- Exercise의 제출은 동료 평가 및 Moulinette 채점으로 **PASS** 혹은 **FAIL**이 결정됩니다.
- Exercise들은 gofmt를 이용하여 포맷팅이 되어 있어야 합니다. 동료 평가 진행 시, 제출된 Exercise가 포맷팅이 되어 있지 않다면 **FAIL**입니다.
- Exercise들은 golangci-lint를 이용했을 때, 별다른 문제가 없다는 것이 검증되어야 합니다. golangci-lint에서 문제가 있는 부분이 발견되면 **FAIL**입니다.
- 동료 평가 진행을 위한 주석은 적극 허용됩니다. 각 함수 및 변수들에게 적용된 주석은 Golang에서 정식으로 지원하는 godoc을 이용하여 Document로 만들 수 있습니다.
- 컴파일은 **go build**를 이용하며, 이를 위해 **go mod**를 먼저 이해해야 합니다. **go mod**를 이용하지 않고 제출된 Exercise 역시 **FAIL**입니다.
- 제공된 서브젝트를 철저히 파악하여 Golang의 기초 지식을 모두 얻을 수 있도록 하십시오.
- 서브젝트에서 요구하는 내용들이 짧더라도, 이 내용들을 이해하고 예상한대로 작동되도록 시간을 쏟는 것은 굉장히 가치 있는 행동입니다. 다양한 시도를 권장합니다.
- *Go ? Ahead ! Write in Go.*

go mod를 이용하기 전에 GOPATH, GOROOT를 알아보십시오.

Chapter II - Preamble



주로 백엔드 어플리케이션을 개발, 테스트, 배포하는 과정에서 다른 환경에서 서로 다른 Configuration을 사용하게 된다. 예를 들어 실제 프로덕션 DB와 테스팅 DB를 따로 두게 된다면 테스팅할 때와 배포할 때 DB의 주소나 포트가 바뀌게 되고, DB 접근에 필요한 유저나 패스워드, 토큰 등이 바뀌게 된다. 따라서 별도의 파일을 두어서 Configuration을 저장하고, 이를 파싱하거나, 가상 환경에 환경변수로 지정하여 사용하게 된다.

Golang에서는 위와 같은 과정을 viper라는 패키지로 해결할 수 있다. viper는 config 파일이나 환경변수, 커맨드 라인 플래그에서 값 등을 읽어오고, 지정하고, Unmarshal 하는 것을 지원한다. 그리고 json, toml, yaml, env, ini 등의 다양한 파일 형식을 지원한다. 특히 viper는 etcd 혹은 Consul 같은 원격 시스템에서도 사용할 수 있다.

viper의 개발자가 만든 cobra 패키지도 둘러보는 것을 권장한다.

- https://github.com/spf13/viper
- https://github.com/spf13/cobra

Chapter III

Main Subject

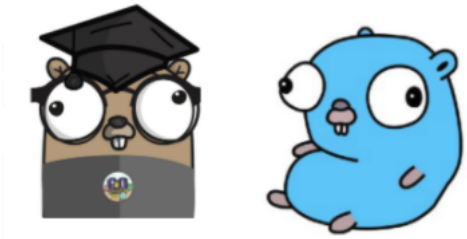
여러분은 c로 만들어진 다음 3개의 프로젝트에 대한 테스터를 만들어야 합니다.

- 컴파일 안 해도 되고, 함수의 리턴만 검사하면 되는 c 프로젝트
- 컴파일 안 해도 되고, stdout으로 출력하는 함수가 있는 코드
- Makefile이 주어져 컴파일을 해야하고, 실행 시 argc, argv로 인자를 주어야 하고, stdout을 검사해야하는 c 프로젝트

여러분은 다음과 같은 사항을 숙지하여야 합니다.

- 각각의 c 프로젝트 파일은 ex0x/ 디렉토리 안에 만들어져야 합니다.
- 여러분은 프로젝트가 제대로 돌아가는지 확인하는 테스트코드를 짜야합니다. 지금까지와 다른 점은 프로젝트가 C코드로 이루어졌다는 점입니다.
- 수동으로 일일이 모든 케이스를 작성하는 것은 금지됩니다. 쉽게 놓칠 수 있는 크리티컬한 케이스만 작성하면 됩니다. 많은 테스트 케이스를 작성해야 하는 상황이면, 어떻게 하면 코드를 통해 추상화하여 쉽게 테스트할 수 있을지 고민하십시오.
- *.go 파일에서는 c 파일에 있는 함수를 불러오기 위한 import와 Wrapping 함수에 대한 코드를 작성하게 됩니다. 테스트 로직은 *_test.go에서만 작성하십시오.
- 여러분의 테스트 코드는 **go test**로 실행되어야 합니다.

Rush00를 끝내게 되면, 앞으로는 Golang를 사용해 c 과제 용도의 테스터를 빠르게 만들 수 있습니다. 다른 Pisciner와 Cadet들을 보다 쉽게 혼내주십시오.



Chapter IV

Exercise 00 : Prime number

Aa Name	Tags
Turn-in directory	ex00
Files to turn in	prime/prime.c , prime/prime.h , prime/prime.go , prime/prime_test.go , go.mod
Module name	ex00
Forbidden packages	None

여러분은 `컴파일 안 해도 되고, 함수의 리턴만 검사하면 되는 c 프로젝트`를 테스트 하는 코드를 작성하게 됩니다.

yongckim과 min-jo는 알고리즘 문제 풀기 내기를 했습니다. 숫자 num을 매개변수로 주면 num이 소수이면 1을, 소수가 아니면 0을 리턴하는 isPrime 함수를 누가 더 빨리 돌아가게 작성할 수 있는지 내기했습니다. yongckim은 isPrimeFast 함수를 작성했고, min-jo는 isPrimeSlow를 작성했습니다. 이 코드들을 테스트하는 테스트코드를 만들어 누가 얼마나 더 빠른지 알려주십시오.

주어지는 c 프로젝트와 관련된 파일들은 다음과 같습니다. yongckim과 min-jo가 코드를 보여주기 너무 부끄럽다고 코드에서 몇몇 부분을 지우고 줬습니다. 이 파일들에서 `//#` 로 표시된 부분을 적절하게 채워 코드가 돌아갈 수 있게 해주십시오.

```
// prime.h
#ifndef PRIME_H
# define PRIME_H

int isPrimeFast(int num);
int isPrimeSlow(int num);

#endif
```

```
// prime.c
#include <math.h>

int isPrimeFast(int num)
{
  int sq;
  int i;

  //#
  sq = sqrt(num);
  i = //#;
  while (i <= sq)
  {
    if (num % i //# 0)
      return (0);
    i++;
  }
  return (1);
}

int isPrimeSlow(int num)
{
  int flag;
  int i;

  //#
  flag = //#;
  i = //#;
  while (i < num)
  {
    if (num % i //# 0)
      flag = //#;
    i++;
  }
  return (flag);
}
```

여러분의 코드는 다음 사항을 만족해야 합니다.

- prime/prime.go 파일에는 c 파일에 있는 함수를 불러오기 위한 import와 Wrapping 함수에 대한 코드만 있어야 합니다. 테스트 로직은 *_test.go에서만 작성하십시오.
- 테스트 코드는 **go test**나 **go test -bench=.** 로 실행될 수 있습니다. 반드시 **go test -bench=.** 를 사용해야하는 것은 아닙니다.
- 테스트 코드는 Benchmark를 활용해 isPrimeFast 함수와 isPrimeSlow 함수 사이의 실행 시간 차이를 보여줄 수 있어야 합니다.

> 💡 go에서 c를 실행하는 방법에 대해 찾아보십시오.

Chapter V

Exercise 01

Aa Name	Tags
Turn-in directory	ex01
Files to turn in	gopher/gopher.go , gopher/gopher_test.go , go.mod
Module name	ex01
Forbidden packages	None

여러분은 `컴파일 안 해도 되고, stdout으로 출력하는 함수가 있는 코드` 를 테스트하는 코드를 작성하게 됩니다.

jseo는 gopher들을 환영해주기 위해 다음과 같은 아스키 아트를 출력하는 함수를 작성했습니다. 함수가 다음 그림을 토시하나 틀리지 않고 잘 출력하는지 테스트 해주세요.

```
.                          ,_---~~~~~----._
                     _,,_,*^____      _____``*g*\"*,
                    / __/ /'     ^.  /      \ ^@q   f
                   [  @f | @))    |  | @))   l  0 _/
                    \`/   \~____ / __ \_____/    \
                     |           _l__l_           I
                     }          [______]           I
                     ]            | | |            |
                     ]              ~ ~            |
                     |                             |
                      |                            |
     _,----.     _,.---._        _ __   ,--.-,,-,--,    ,-----.
   _.='.'-,  \  ,-.' , -  `.    .-`.' ,`./==/  /|=|  | ,-.--` , \  .-.,.---.
  /==.'-     / /==/_,  ,  - \ /==/, -   \==|_ ||=|, ||==|-  _.-` /==/  `   \
 /==/ -   .-' |==|   .=.     |==| _ .=. |==| ,|/=| _||==|    `.-.|==|-, .=., |
 |==|_   /_,-.|==|_ : ;=:  - |==| , '=',|==|- `-' _ /==/_ ,    /|==|   '='  /
 |==|  , \_.' )==| , '='     |==|-  '..'|==|  _     |==|    .-' |==|- ,   .'
 \==\-  ,    ( \==\ -    ,_ /|==|,  |   |==|   .-. ,\==|_  ,`-._|==|_  . ,'.
  /==/ _  ,  /  '.=. -   .' /==/ - |   /==/, //=/  /==/ ,     //==/  /\ ,  )
  `--`------'     `--`--''   `--`---'   `--`-' `-`--`--`-----`` `--`-`--`--'
```

함수는 다음처럼 작성되어 있고 Escape Sequence와 Backtick 때문에 출력이 잘 예측되지 않습니다.

```
// gopher.go
package gopher

import "fmt"

func PrintGopher() {
  fmt.Println(`
.                          ,_---~~~~~----._
                     _,,_,*^____      _____`+"`"+"`"+`*g*\"*,
                    / __/ /'     ^.  /      \ ^@q   f
                   [  @f | @))    |  | @))   l  0 _/
                    \`+"`"+`/   \~____ / __ \_____/    \
                     |           _l__l_           I
                     }          [______]           I
                     ]            | | |            |
                     ]              ~ ~            |
                     |                             |
                      |                            |
     _,----.     _,.---._        _ __   ,--.-,,-,--,    ,-----.
   _.='.'-,  \  ,-.' , -  `+"`"+`.   .-`+"`"+`.' ,`+"`"+`./==/  /|=|  | ,-.--`+"`"+` , \  .-.,.---.
  /==.'-     / /==/_,  ,  - \ /==/, -   \==|_ ||=|, ||==|-  _.-`+"`"+` /==/  `+"`"+`   \
 /==/ -   .-' |==|   .=.     |==| _ .=. |==| ,|/=| _||==|    `+"`"+`.-.|==|-, .=., |
 |==|_   /_,-.|==|_ : ;=:  - |==| , '=',|==|- `+"`"+`-' _ /==/_ ,    /|==|   '='  /
 |==|  , \_.' )==| , '='     |==|-  '..'|==|  _     |==|    .-' |==|- ,   .'
 \==\-  ,    ( \==\ -    ,_ /|==|,  |   |==|   .-. ,\==|_  ,`+"`"+`-._|==|_  . ,'.
  /==/ _  ,  /  '.=. -   .' /==/ - |   /==/, //=/  /==/ ,     //==/  /\ ,  )
  `+"`"+`--`+"`"+`------'     `+"`"+`--`+"`"+`--''   `+"`"+`--`+"`"+`---'   `+"`"+`--`+"`"+`-' `+"`"+`-`+"`"+`--`+"`"+`--`+"`"+`-----`+
`)
}
```

함수 PrintGopher를 테스트하는 테스트 코드를 작성하십시오. string을 검사하는 하드 코딩은 금지됩니다. 복사 붙여넣기만으로 테스트코드를 만들 수 있는 방법이 있을 겁니다!

여러분의 코드는 다음 사항을 만족해야 합니다.

- gopher_test.go 파일에는 다음과 같은 함수가 포함되어야 합니다.

```
func ExamplePrintGopher()
```

- 테스트코드는 **go test**로 실행할 수 있어야 합니다.

💡 stdout으로 출력해버리는 코드를 테스트하려면 다음 링크를 참고하십시오.
https://go.dev/blog/examples

Chapter VI

Exercise 02 : Star

Aa Name	Tags
Turn-in directory	ex02
Files to turn in	star/star.go , star/star_test.go , go.mod
Module name	ex02
Forbidden packages	None

여러분은 `Makefile이 주어져 컴파일을 해야하고, 실행 시 argc, argv로 인자를 주어야 하고, 리턴이랑 stdout을 검사해야하는 c 프로젝트`를 테스트하는 코드를 작성하게 됩니다.

- c파일과 Makefile은 채점 과정에서 ex02/c/ 폴더에 자동으로 추가됩니다. 그 전에 여러분들이 코드의 내용을 알 방법은 없습니다.
- 파일의 이름은 star.c와 Makefile로 주어지므로 이름과 같은 파일을 만들지 마십시오!
- 여러분은 테스트 코드를 Golang으로 작성해야합니다. 여러분의 코드는 주어지는 c파일을 Makefile 컴파일하고, **go test**를 통해 테스트 케이스들을 검사하게 됩니다.

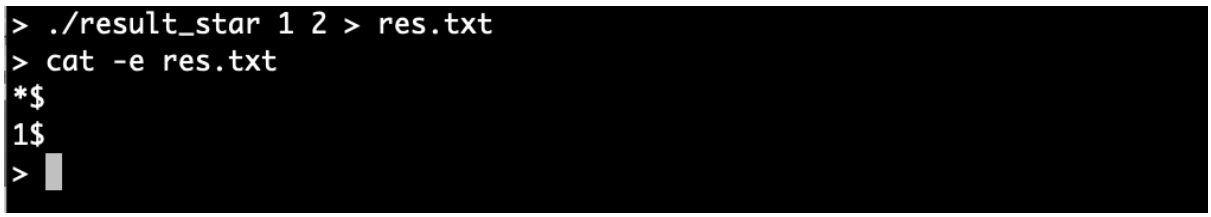
주어지는 c 프로젝트를 컴파일한 결과를 실행해보면 다음과 같습니다. 마지막에는 출력한 *의 갯수를 출력합니다.

```
> make
> make clean
> ./result_star 3
```

```
> ./result_star 0 > res.txt
> cat -e res.txt
0$
>
```

```
> ./result_star 1 > res.txt
> cat -e res.txt
*$
1$
>
```

```
> ./result_star 2 > res.txt
> cat -e res.txt
   *$
  ***$
 *   *$
*** ***$
12$
>
```



주어지는 c코드는 특정 값이 들어갔을 때 예상되는 결과와 다른 출력을 내보냅니다! 여러분의 테스터는 어떤 값이 들어갔을 때 c코드가 오류를 내는지 보여줄 수 있어야 합니다. 이상한 문자열을 넣지 않고, 60이하의 숫자만 넣을 것입니다. 모든 경우를 일일이 다 손으로 적을 수는 없을 것입니다. 어쩌면 Golang으로 c 바이너리처럼 동작하는 코드를 먼저 작성해보는 것이 테스트에 유용할지도 모릅니다.

여러분의 star_test.go 파일에는 다음과 같은 함수가 있어야 합니다.

```
func TestStar(t *testing.T)
func TestInputBehavior(t *testing.T)
func TestMain(m *testing.M)
```

그 외의 조건은 아래와 같습니다.

- TestStar 함수는 star.go에 있는 Run 함수에 적절한 인자를 주어 다양한 테스트 케이스를 실행해야 합니다. 아래와 같은 케이스가 포함됩니다.

```
./restul_star 0
./restul_star 1
```

- TestInputBehavior는 다양한 인자를 주고, 이를 테스트 해야 합니다. 이상한 문자열이 올 수 없고, 60이하의 숫자만 들어갑니다. 아래와 같은 케이스가 포함됩니다.

```
./restul_star 1 2 3
./restul_star 1 2
```

- TestMain 함수는 c파일과 Makefile이 적절한 위치에 있는지 확인하고, 컴파일을 수행해야 합니다. 주어진 c파일의 컴파일과 관련한 오류가 발생하면, 테스트 케이스를 실행하지 않습니다. 또한 다음 사항을 충족해야 합니다.
  - c/ 폴더 안에 star.c가 없으면 `star.c does not exist in c/` 오류 메세지를 보여줄 수 있어야 합니다.
  - c/ 폴더 안에 Makefile이 없으면 `Makefile does not exist in c/` 오류 메세지를 보여줄 수 있어야 합니다.
  - **make**에 실패했을 때 `make fail` 오류 메세지를 보여줄 수 있어야 합니다.
  - **make clean**에 실패했을 때 `make clean fail` 오류 메세지를 보여줄 수 있어야 합니다.

```
> go test
2021/11/05 12:16:32 Makefile does not exist in c/
2021/11/05 12:16:32 stat c/Makefile: no such file or directory
exit status 1
FAIL    ex02/star       0.161s
>
```

여러분의 star.go 파일에는 다음과 같은 함수 하나만 있어야 합니다.

```
func Run(given ...string) (output string, err error)
```

- Run 함수는 컴파일된 c 바이너리 파일을 실행시키고, 결과를 받아와서, 바이너리 파일이 stdout으로 출력한 내용을 output으로 반환해야 합니다.
- Run 함수의 매개변수는 다음처럼 사용되어, 실행할 c 바이너리에 매개변수를 넘겨주는 용도로 사용됩니다.

```
func main() {
  Run("./result_star", "3")
}
```

여러분이 작성한 테스트 코드는 **go test**를 통해 실행되어야 하며, TestStar 함수에서는 다음처럼 60이하의 숫자를 입력해보는 테스트를 진행해야 합니다. 별다른 이상이 없으면 passed를 출력해야 합니다.

```
55 passed
56 passed
57 passed
58 passed
59 passed
60 passed
PASS
ok      ex02/star       18.222s
>
```

이상이 있다면, 해당 부분을 출력하고 끝내면 됩니다. 아래 그림은 숫자 2에서 아무것도 출력을 하지 않는 예시입니다. 예시일 뿐 정확히 일치할 필요는 없습니다. Error를 출력하고, 반복을 정상적으로 종료할 수 있어야 합니다.

```
> go test
-1 passed
0 passed
1 passed
--- FAIL: TestStar (0.00s)
    star_test.go:122:
                Error Trace:    star_test.go:122
                Error:          Not equal:
                                expected: "   *\n  ***\n *   *\n*** ***\
n12\n"
                                actual  : ""

                                Diff:
                                --- Expected
                                +++ Actual
                                @@ -1,6 +1 @@
                                -   *
                                -  ***
                                - *   *
                                -*** ***
                                -12

                Test:           TestStar
FAIL
exit status 1
FAIL    ex02/star       0.522s
> 
```

os/exec 모듈을 이용하면 shell 명령어를 실행할 수 있습니다.

Chapter VII

Bonus

Aa Name	Tags
Turn-in directory	bonus
Files to turn in	all you need
Module name	bonus
Forbidden packages	None

앞의 과제에서 다루어 보았던 http 패키지를 이용하여 Moulinette를 만드십시오. 조건은 아래와 같습니다.

- 다음과 같은 명령어로 서버를 실행할 수 있어야 합니다.

```
go build
./bonus -p=42424
```

- 웹브라우저 URL에 아래와 같이 입력하면, 깃허브에서 클론을 하고 테스트 코드를 실행할 수 있어야 합니다. 테스트 코드를 실행한 결과는 웹브라우저에 표시될 수 있어야 합니다. 이 때 실행되는 테스트 코드는 클론한 테스트 코드가 아니라 여러분이 작성한 테스트 코드여야 합니다.

```
localhost:42424/ex00/prime&LINK=GITHUB_URL
```

- 여러분의 Moulinette은 다음과 같은 플래그를 만족해야 합니다.
  - -h or -help : 어떤 flag들이 있는지 도움말을 보여줍니다.
  - -p : 서버를 열 포트 번호를 설정합니다.

- 여태까지 모든 서브젝트들은 문제마다 테스트 코드들을 만들도록 권장했습니다. 여러분은 아마 모든 문제마다 테스트 코드들을 가지고 있을 겁니다. 다른 서브젝트들의 테스트 코드도 돌려볼 수 있을 것입니다.

Piscine Golang의 Moulinette도 이런식으로 만들었을 것입니다. 아마도요...