[실습과제2] Go코드 분석

예제로 제공하는 소스코드를 분석하라.

SortKeys.go

사용패키지와 SortKeys.go 소스코드 구현 내용 모두를 분석하라

SortKeys.go코드에서 fmt, sort 패키지를 가져온다.

```
import (
    "fmt"
    "sort"
)
```

사용하는 구조체 타입의 명명(naming)은 플레닛(Planet)으로 행성을 뜻한다.

```
type Planet struct {
    name     string
    mass     earthMass
    distance au
}
```

행성이 가지고 있는 변수는 3개로 문자열 타입의 이름(name), 실수형 질량(mass), 실수형 거리(distance)이다.

```
type earthMass float64
type au float64
```

지구와 태양 사이의 거리를 1au로 정의하고 있기 때문에 각 행성이 가지고 있는 값은 상대적 비율이다. 질량도 지구의 질량을 1로 정의하고 다른 행성과 상대적 비율이 된다. 두 실수형의 변수를 확실하게 구분하기 위해 질량(mass)를 earthMass, 거리(distance)를 au로 재정의한다.

```
var planets = []Planet{
}
```

변수 행성들(planets)는 플레닛(Planet) 타입의 슬라이스(Slice)로 가변적으로 추가(append), 삭제(delete)를 할 수 있다.

```
    {"Mercury", 0.055, 0.4},
    {"Venus", 0.815, 0.7},
    {"Earth", 1.0, 1.0},
    {"Mars", 0.107, 1.5},
```

행성들의 요소는 수성(Mercury), 금성(Venus), 지구(Earth), 화성(Mars)로 구성되어 있다. 이름으로 지구(Earth), 화성(Mars), 수성(Mercury), 금성(Venus)로 정렬된다. 질량으로 수성(0.055), 화성(0.107), 금성(0.815), 지구(1)로 정렬된다. 거리로는 수성(0.4), 금성(0.7), 지구(1), 화성(1.5)로 정렬된다.

```
func(p1, p2 *Planet)
```

행성을 정렬하기 위해서 클로저(closure)를 사용하여 행성(Planet) 구조체(Structure)를 이용한다.

```
name := func(p1, p2 *Planet) bool {
        return p1.name < p2.name
    }
```

이름(name) 클로저는 두 개의 행성 구조체의 포인터를 받아서 앞의 행성 이름이 작으면 참(true)를 반환하고, 앞 행성의 이름이 크면 거짓(false)를 반환한다.

```
fmt.Println(p1.name, p2.name, p1.name < p2.name)
```

이름 클로저 작동 확인을 위해 출력한다.

```
Venus Mercury false
Earth Venus true
Earth Mercury true
Mars Venus true
Mars Mercury true
Mars Earth false
```

이름 클로저는 두 행성의 이름을 받아서 알파벳 순으로 되어 있으면 참(true) 역순으로 되어 있으면 거짓(false)로 출력이 확인된다.

```
mass := func(p1, p2 *Planet) bool {
        return p1.mass < p2.mass
    }
```

질량(mass) 클로저는 두 개의 행성 구조체의 포인터를 받아서 앞의 행성의 질량이 작으면 참(true)를 반환하고, 앞 행성의 질량이 크면 거짓(false)를 반환한다.

```
fmt.Println(p1.name, ":", p1.mass, p2.name, ":", p2.mass, p1.mass < p2.mass)
```

질량 클로저 작동 확인을 위해 출력한다.

```
Mars : 0.107 Earth : 1 true
Mercury : 0.055 Earth : 1 true
Mercury : 0.055 Mars : 0.107 true
Venus : 0.815 Earth : 1 true
Venus : 0.815 Mars : 0.107 false
```

질량 클로저는 두 행성의 질량을 입력 받아 질량 순으로 되어 있으면 참(true) 역순으로 정렬되어 있으면 거짓(false)로 출력된다.

```
    distance := func(p1, p2 *Planet) bool {
        return p1.distance < p2.distance
    }
```

거리(distance) 클로저는 두 개의 행성 구조체의 포인터를 방아서 앞 행성의 거리가 작으면 참(true)를 반환하고, 앞 행성의 거리가 크면 거짓(false)를 반환한다.

```
fmt.Println(p1.name, ":", p1.distance, p2.name, ":", p2.distance, p1.distance
< p2.distance)
```

거리 클로저의 작동 확인을 위해 출력한다.

```
Mars : 1.5 Mercury : 0.4 false
Venus : 0.7 Mars : 1.5 true
```

```
Venus : 0.7 Mercury : 0.4 false
Earth : 1 Mars : 1.5 true
Earth : 1 Venus : 0.7 false
```

거리 클로저는 두 행성의 거리를 입력 받아 거리 순으로 되어 있으면 참(true)로 역순으로 정렬되어 있으면 거짓(false)로 출력된다.

```
    decreasingDistance := func(p1, p2 *Planet) bool {
        return distance(p2, p1)
    }
```

거리 역정렬(descresingDistance)는 위의 거리 클로저의 매개변수에 앞 행성을 두 번째 매개변수로 뒤 행성을 첫 번째 매개변수로 입력하여 역정렬이 되도록 한다.

```
Mercury : 0.4 Venus : 0.7 true
Mercury : 0.4 Earth : 1 true
Venus : 0.7 Earth : 1 true
Mercury : 0.4 Mars : 1.5 true
Venus : 0.7 Mars : 1.5 true
Earth : 1 Mars : 1.5 true
```

거리 클로저 작동을 위해 출력을 한 코드도 거리 역정렬 클로저에도 출력 확인할 수 있다.

```
// Sort the planets by the various criteria.
```

여러 기준을 가지고 행성들을 정렬한다.

```
By(name).Sort(planets)
```

이름을 기준으로 정렬(Sort)함수를 호출하여 정렬한다.

```
By name: [{Earth 1 1} {Mars 0.107 1.5} {Mercury 0.055 0.4} {Venus 0.815 0.7}]
```

결과를 보면 알파벳 순서를 가지고 정렬을 출력한다. 입력되는 변수와 불러오는 함수를 통해 처리 결과가 출력되는 과정을 살펴보면 다음과 같다.

```
type By func(p1, p2 *Planet) bool
```

타입(Type)문을 통해 By라는 함수의 원형을 정의한다. By함수의 구현은 less라는 함수의 인터페이스를 구현한다. 행성의 인수를 입력되면 호출된다. sort패키지에서 구현되어야한 인터페이스는 3가지 메서드(method) 원형이 있다.

```
type Interface interface {
    // Len is the number of elements in the collection.
    Len() int
    // Less reports whether the element with
    // index i should sort before the element with index j.
    Less(i, j int) bool
    // Swap swaps the elements with indexes i and j.
    Swap(i, j int)
}
```

Len() 함수는 주어진 원소가 몇 개가 되는지 정수(integer)값을 반환한다. Less() 함수는 원소의 크고 작음을 판단한다. Swap() 함수는 대소 관계를 판별된 위치 변환한다. 정렬(sort)에서 선언된 인

터페이스는 공통적인 행위(operation)을 하는 추상화하여 묶어준다. Len(), Less(), Swap()을 하는 것들을 정렬(sort)라는 상위 개념으로 만든다. 인터페이스는 공통의 메소드를 통해서 암시(implicit)적으로 충족된다.

```
func (s *planetSorter) Len() int {
    return len(s.planets)
}
```

Len() 함수는 planetSorter의 포인터 리시버를 사용한다. planetSorter의 포인터의 행성들의 길이를 반환한다.

```
type planetSorter struct {
    planets []Planet
    by      func(p1, p2 *Planet) bool // Closure used in the Less method.
}
```

planetSorter는 포인터 리시버를 사용하지 않으면 값 전달을 하게 된다. 구조체의 값을 복사하여 전달한다. 메소드 안에서 구조체의 값이 변경되더라도 호출한 데이터는 반영되지 않는다. 반면 포인터 리시버는 함수의 값이 변경되면 그대로 값이 반영되어 변경된다. 행성들(planets)의 기준을 따라 정렬된다면 호출한 데이터가 반영되어야 하기 때문에 포인터 리시버를 사용한다. planetSorter 구조체는 행성들(planets)의 변수명을 가지고 있는 행성(Planet)슬라이스와 행성(planet)의 포인터를 매개변수를 함수 입력 값으로 사용하여 참 거짓을 판별하는 by 함수로 구성된다.

```
func (s *planetSorter) Swap(i, j int) {
    s.planets[i], s.planets[j] = s.planets[j], s.planets[i]
}
```

Swap() 함수는 매개변수로 입력된 I, j정수의 planetSort의 행성들의 I와 j를 교환(swap)한다.

```
func (s *planetSorter) Less(i, j int) bool {
    return s.by(&s.planets[i], &s.planets[j])
}
```

Less() 함수는 매개변로 입력된 I, j정수의 대소 관계를 판별해 참, 거짓을 반환한다. 값 비교를 레퍼런스(reference) 파라미터(parameter)로 입력한다. 실제로 by 함수를 구현하지 않았지만 Less() 함수 내부에서 by를 호출함으로써 인터페이스 내부에서 선언된 메소드를 가져오게 된다.

```
fmt.Printf("mem_ps: %p, mem_planets: %p, mem_by: %p\n", &ps, &ps.planets, &ps.
by)
fmt.Printf("Less_mem_s: %p, Less_mem_planets: %p, Less_mem_by: %p\n", &s, &s.p
lanets, &s.by)
```

Sort 함수 내부에서 ps와 내부 변수와 함수의 메모리 값을 출력해보고 Less() 함수 내부의 s와 변수와 함수 메모리 값을 출력하여 확인한다.

```
mem_ps: 0xc000006028, mem_planets: 0xc000004480, mem_by: 0xc000004498
Less_mem_s: 0xc000006038, Less_mem_planets: 0xc000004480, Less_mem_by: 0xc0000
04498
```

레퍼런스를 받아서 할당했기 때문에 행성들(planets)와 by함수 메모리 값이 동일한 것을 확인 할 수 있다. 암시적 할당을 통해 연결되어 있는 것을 메모리 출력을 통해 확인할 수 있다.

```
By name: [{Earth 1 1} {Mars 0.107 1.5} {Mercury 0.055 0.4} {Venus 0.815 0.7}]
```

함수의 내용을 출력해보면 둘 간의 이름을 기준으로 비교해서 앞 더 크면 거짓(false)로 앞이 작으면 참(true)로 판별해서 반환한다.

```
func (by By) Sort(planets []Planet) {
    ps := &planetSorter{
        planets: planets,
        by:      by,
    }
    sort.Sort(ps)
}
```

Sort함수는 Planet슬라이스를 함수인자로 받아들여서 ps라는 planetSorter의 레퍼런스(reference) 패스 값을 선언 후 저장한다. sort패키지 Sort함수를 호출하여 정렬한다.

```
By(mass).Sort(planets)
```

위에 이름(name)을 기준으로 출력한 과정과 동일하게 질량(mass)를 기준으로 정렬한다.

```
fmt.Println("By mass:", planets)
```

정렬된 결과를 질량을 기준으로 출력한다.

```
By mass: [{Mercury 0.055 0.4} {Mars 0.107 1.5} {Venus 0.815 0.7} {Earth 1 1}]
```

질량(mass)를 기준으로 정렬된 결과를 볼 수 있다.

```
By(distance).Sort(planets)
```

거리(distance)를 기준으로 정렬한다.

```
fmt.Println("By distance:", planets)
```

거리를 기준으로 정렬된 결과를 출력한다.

```
By distance: [{Mercury 0.055 0.4} {Venus 0.815 0.7} {Earth 1 1} {Mars 0.107 1.5}]
```

거리를 기준으로 정렬된 결과를 볼 수 있다.

```
By(decreasingDistance).Sort(planets)
```

거리(distance)를 기준으로 역으로 정렬한다.

```
fmt.Println("By decreasing distance:", planets)
```

거리를 역정렬한 결과를 출력한다.

```
By decreasing distance: [{Mars 0.107 1.5} {Earth 1 1} {Venus 0.815 0.7} {Mercury 0.055 0.4}]
```

거리역정렬한 결과를 볼 수 있다.