Maps

Définition

Structure associant des clés à des valeurs

On peut mettre en clé tout ce qui est comparable (on peut mettre une structure comme clé)

Syntaxe

La syntaxe “longue” de déclaration d’une map est la suivante :

```
var m map[KeyType]ValueType
---------------------------
var m map[string]int = make(map[string]int)
```

Grâce à l’inférence des types à la déclaration, on peut encore simplifier cette syntaxe en :

```
m2 := make(map[string]int)
```

Opérations

Pour ces exemples, nous avons une map qui a pour clé une chaine de caractère et pour valeur un entier.

```
myMap := make(map[string]int)
```

On peut récupérer la taille de map en utilisant une fonction que nous connaissons depuis les slices `len()`

```
fmt.Printf("Map size %v\n", len(myMap))
```

Assignation

L’assignation et très simple, trèèèèès simple !

```
myMap["hello"] = 5
myMap["goodbye"] = 10
```

Récupération

Pour récupérer la valeur, comme pour l'assignation, il suffit de faire comme avec un tableau

```
fmt.Printf("key=hello, value=%v\n", myMap["hello"])
```

Présence d’une clé

Pour tester la présence, on utilise le retour multiple caché dans la récupération d’une valeur

```
j, isPresent := myMap["helo"]
fmt.Printf("j=%v, isPresent =%v\n", j, isPresent )
```

`isPresent` est un type booléen et est égale à `false` si la clé n’existe pas.

Dans le cas où la clé n’existe pas, la valeur sera celle par défaut du type (0 pour un entier, chaine vide pour une chaine de caractères, ...)

Si on souhaite mettre ce test dans une condition, on peut faire de cette manière :

```
if _, present = myMap["hel"]; present {
// ... code
}
```

Supprimer une clé / valeur

On utilise la fonction `delete`

```
delete(myMap, "hello")
```

Assignation rapide et parcours

On peut assigner des valeurs dans notre map dès la déclaration comme avec les tableaux

```
myMap := map[string]int{
"Noé": 10,
"Paul": 15,
"Swann": 18,
"Nathanael" : 0
}
```

Pour parcourir une map, on peut utiliser `range`

```
for name, idk := range myMap{
fmt.Printf("name=%v, idk=%v\n", name, idk)
}

// Only keys
for name := range m {
fmt.Printf("name=%v\n", name)
}
```

Map & Struct

Pour illustrer la mise en place d'une map constitué de structures, on va utiliser les structures suivantes

```
type User struct {
name string
}

type Key struct {
ID   int
Name string
}
```

On peut créer une map de cette manière :

```
myMap := make(map[Key]User)

myMap[Key{1,"ceo"}] = User{"Swann"}
```

```
fmt.Printf("%v"), myMap[Key{1,"ceo"}])
```

> Lorsqu’on récupère une structure depuis une map, on récupère en réalité une copie de cette dernière. Pour palier à ça on peut transformer notre map en une map de pointeurs

```
myMap := make(map[Key]*User)

myMap[Key{1,"ceo"}] = &User{"Swann"}

fmt.Printf("%v"), *myMap[Key{1,"ceo"}])
```

Revision #1
Created 2022-05-09 19:20:23 UTC by Noé Larrieu-Lacoste
Updated 2022-05-09 19:21:18 UTC by Noé Larrieu-Lacoste