Go
5-Type composé

J. Vlasak

Go 5-Type composé

But du cours

- Savoir manipuler les types composés en go.

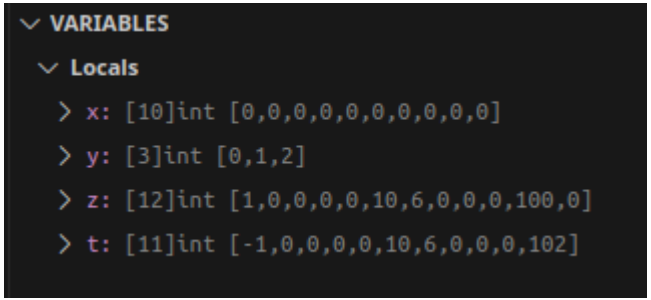
Go 5-Type composé

Les tableaux : trop rigide pour être utilisé directement

- Dans un tableau, la taille est fixée lors de la création
- Manipule souvent des données de taille inconnues

```
var x [10]int
var y = [3]int{0, 1, 2}
var z = [12]int{1, 5: 10, 6, 10: 100}
var t = [...]int{-1, 5: 10, 6, 10: 102}
```

Taille définie à la compilation

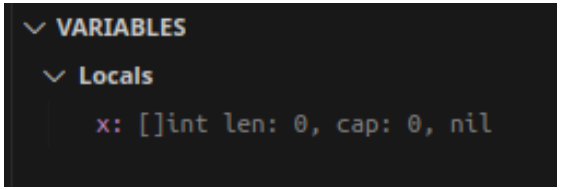


Go 5-Type composé

Les tranches (slice)

- Ressemble à des tableaux, mais pas de définition de taille

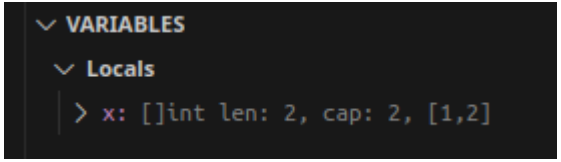
```
 9    var x []int
10
11    fmt.Println(x)
```



- Codée dans le langage, et se compose de 3 éléments :
  - Longueur
  - Capacité
  - Pointeur

- Initialisation explicite

```
var x = []int{1, 2}
```



Go 5-Type composé

Les tranches (slice)

- Initialisation à l'aide de fonctions intégrées dédiées. Ne nécessite pas de bibliothèque externe au langage.

```
var x = make([]int, 2)
x[0] = 1
x[1] = 2

fmt.Println(len(x), cap(x), x)
x = append(x, 3)

fmt.Println(len(x), cap(x), x)
var y = make([]int, 2)
copy(x, y)
fmt.Println(len(y), cap(y), y)
```

```
2 2 [1 2]

3 4 [1 2 3]

2 2 [1 2]
```

Go 5-Type composé

Extraction de tranche (slice)

- Il est possible de créer des tranches à partir de tranche
- Permet d'extraire facilement des données du slice

```
x := []int{1, 2, 3, 4}
y := x[:2]
z := x[1:]
a := x[1:3]
b := x[:]

fmt.Println("x: ", x)
fmt.Println("y: ", y)
fmt.Println("z: ", z)
fmt.Println("a: ", a)
fmt.Println("b: ", b)
```

```
x:  [1 2 3 4]
y:  [1 2]
z:  [2 3 4]
a:  [2 3]
b:  [1 2 3 4]
```

- Les données sont partagées
- b[0]=10 => modification de x,y,z et a éventuellement

```
x:  [10 2 3 4]
y:  [10 2]
z:  [2 3 4]
a:  [2 3]
b:  [10 2 3 4]
```

Go 5-Type composé

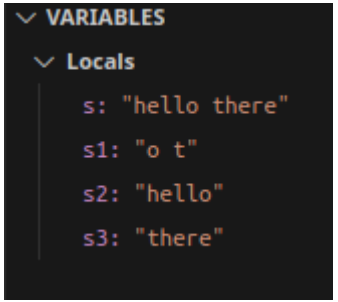
Extraction de tranche (slice)

- Retour sur le type string
- extraire des sous chaînes

```
var s string = "hello there"

s1 := s[4:7]
s2 := s[:5]
s3 := s[6:]

fmt.Println(s)
fmt.Println(s1)
fmt.Println(s2)
fmt.Println(s3)
```



- Les données sont partagées

Go 5-Type composé

Passage par paramètre ou référence ?

- Les variables de type tableau sont recopiées sur la pile
- Les slice par référence : c'est un pointeur qui est recopié sur la pile
- Faite le test !!

```
func modif(t [10]int) {
    t[0] = 10
}
func main() {

    var tab = [10]int{1, 2, 3}
    modif(tab)
    fmt.Println(tab)
```

```
> tab: [10]int [1,2,3,0,0,0,0,0,0,0]
```

```
func modif(t []int) {
    t[0] = 10
}
func main() {

    var tab = []int{1, 2, 3}
    modif(tab)
    fmt.Println(tab)
```

```
> tab: []int len: 3, cap: 3, [10,2,3]
```

Go 5-Type composé

Les tableaux associatifs (map)

- C’est une extension des tableaux
- L’accès n’est pas forcement un entier, mais de n’importe quels types.
- Simplifie les algorithmes ou une clef donne une valeur

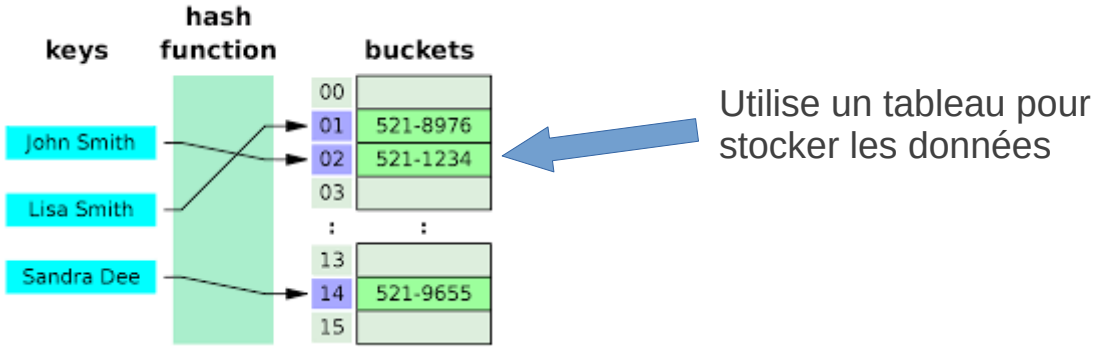
Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Utilisation remarquable : généralisation analyse paramètre des modules

Go 5-Type composé

Les tableaux associatifs (map)

- Il existe plusieurs manière d'implémenter une map
- En Golang, c'est une « hash map » qui est implémentée
- La difficulté est de trouver une fonction de hachage avec peu de collision



Go 5-Type composé

Les maps : utilisation

- S’utilise comme un tableau, avec les même fonctions natives

```
var demoMap map[string]int
demoMap = make(map[string]int)
demoMap["ee"] = 10
delete(demoMap, "ee")

mapOfInt := map[string]int{}
mapOfInt["ee"] = 10
```

- La taille de la map s’ajuste automatiquement

Go 5-Type composé

Les maps : utilisation

- Afficher tous les éléments d'une map

```
mapOfInt := map[string]int{}
mapOfInt["ee"] = 10
mapOfInt["ff"] = 20

for k, v := range mapOfInt {
    fmt.Printf("key : %s valeur :%d\n", k, v)
}
```

```
key : ee valeur :10
key : ff valeur :20
```

Go 5-Type composé

Les maps : utilisation

- Comment résoudre le problème de tester la présence d'une valeur? Dans notre code, « mapOgInt["zz"] » retourne 0

```
mapOfInt := map[string]int{}
mapOfInt["ee"] = 10
```

- « Utiliser l'idiome de la virgule et du bon »

```
v, ok := mapOfInt["zz"]

if !ok {
    fmt.Println("zz n'est pas present mais v a pour valeur: ", v)
}
```

```
zz n'est pas present mais v a pour valeur:  0
```

Go 5-Type composé

Les structures

- Il est possible d'utiliser une map pour passer des données de fonction en fonction
- Utilisation limité car les données doivent être homogènes
- Intérêt des structures
  - Regroupe des données hétérogènes
  - Définit un nouveau type. Utilisable dans les tableaux, ...
- Remarque : « type » permet de définir un nouveau type

```
type Personne struct {
    nom string
    age int
}
```

```
type Mymap map[string]int
b := Mymap{"ee": 10}
```

Go 5-Type composé

Les structures : utilisation

- Comme tout type, une structure s'initialise, se modifie et il est possible d'utiliser les pointeurs

```
type Personne struct {
    nom string
    age int
}

moi := Personne{}

moi = Personne{
    nom: "vlasak",
    age: 10,
}

ptrPersonne := &moi
ptrPersonne.age++

fmt.Println(ptrPersonne)
```

```
moi: main.Personne·1 {nom: "", age: 0}
  nom: ""
  age: 0
```

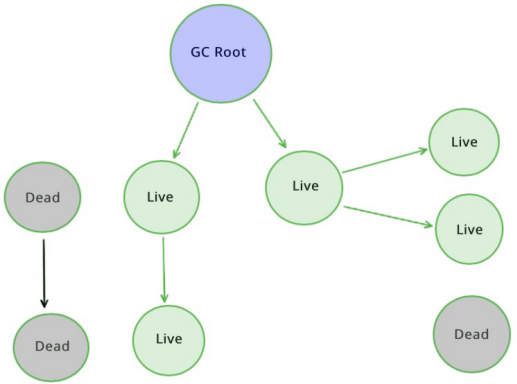
```
moi: main.Personne·1 {nom: "vlasak", age:…
  nom: "vlasak"
  age: 10
```

```
moi: main.Personne·1 {nom: "vlasak", age: 11}
  nom: "vlasak"
  age: 11
ptrPersonne: *main.Personne·1 {nom: "vlasak", age: 11}
  : main.Personne·1 {nom: "vlasak", age: 11}
    nom: "vlasak"
    age: 11
```

Go 5-Type composé

Les structures, map et autre objet : destruction

- Pas de destruction explicite, garbage collector
- Le GC fonctionne en suivant un ensemble de règles pour déterminer quels objets sont encore utilisés et quels objets sont devenus inutilisés. Les objets inutilisés sont alors marqués pour être libérés.



- Les avantages du GC
  - Il simplifie la gestion de la mémoire pour les développeurs.
  - Il réduit le risque de fuites de mémoire.
  - Il peut améliorer les performances du programme.
- Les inconvénients du GC
  - Il peut entraîner des pauses dans l'exécution du programme.
  - Il peut être plus difficile à déboguer que la gestion manuelle de la mémoire.

Go 5-Type composé