LES POINTEURS DANS LE LANGAGE DE PROGRAMMATION GO

Le fonctionnement de mémoire

Fonctionnement

Avant de vous présenter le concept des pointeurs, une petite explication à propos de la **mémoire** s'exige.

Vous ne vous êtes jamais demandé comment votre machine manipule les valeurs de vos variables ?

Et bah c'est grâce aux **adresses** de vos variables !

"Hein une adresse comme une adresse postale ?" ?

Oui tout à fait, toute variable manipulée dans un programme est stockée quelque part dans la mémoire de votre machine. Cette mémoire est constituée d'**octets** qui sont identifiés de manière univoque par un numéro qu'on appelle adresse Donc si votre machine veut manipuler vos variables, elle aura besoin au préalable de connaître où se situe votre variable grâce à son **adresse mémoire** qui lui indiquera la position de la donnée dans la **mémoire vive** (la RAM de votre ordinateur).

Par exemple si on prend le fonctionnement de la poste, le facteur a besoin de l'adresse de destination pour savoir chez qui il doit renvoyer la lettre. Ici le facteur c'est votre ordinateur et l'adresse destination correspond à l'adresse de la variable que l'ordinateur souhaite chercher.

Afficher l'adresse d'une variable

On utilisera la fonction `Printf()` avec le symbole `%p` et le signe `&` pour accéder et afficher à l'adresse d'une variable.

Un exemple sera beaucoup plus parlant :

```
package main

import (
    "fmt"
)

func main() {
    var a int = 20
    fmt.Printf("Adresse de la variable a: %p\n", &a)
}
```

Résultat :

```
Adresse de la variable a: 0xc0000100a0
```

Schéma explicatif

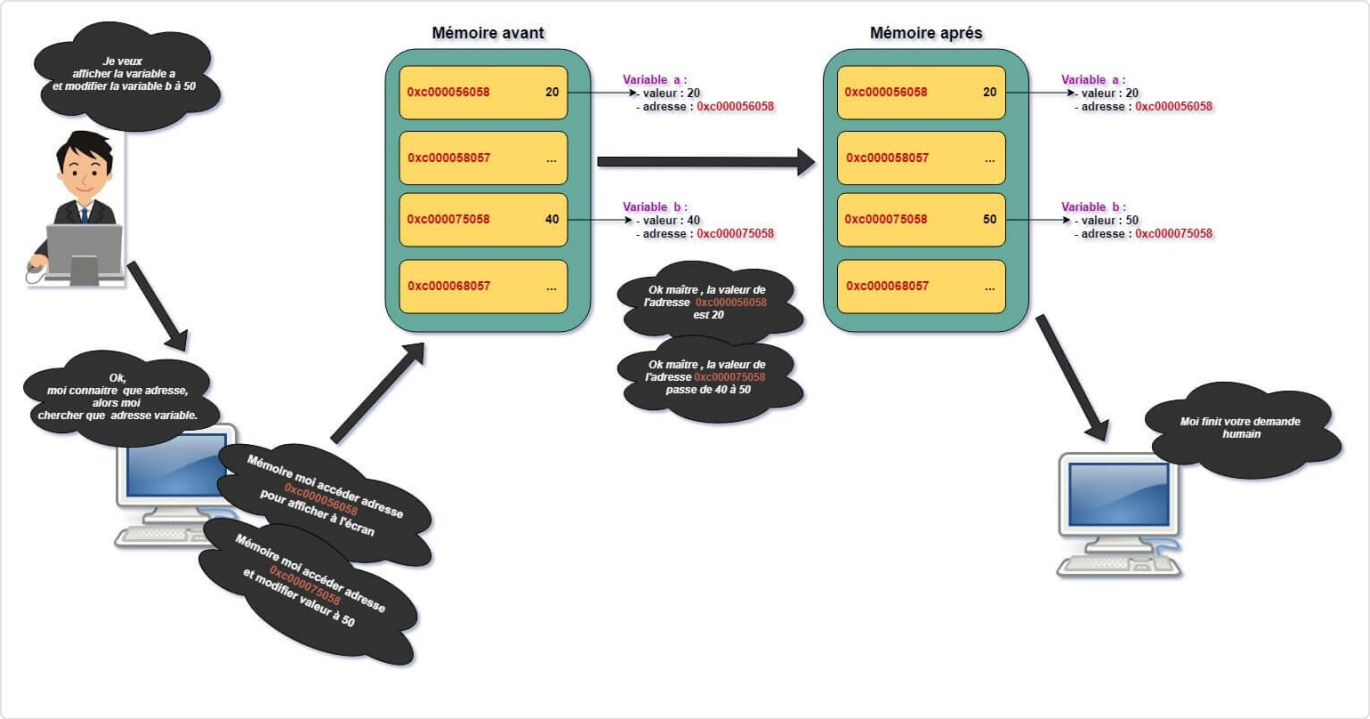
Je vous partage ici un code simple qui affiche et modifie une variable et je vais ensuite vous expliquer comment votre machine raisonne quand elle exécute votre code.

```
package main

import (
 "fmt"
)

func main() {
 var a int = 20
 var b int = 40
 fmt.Println(a)
 b = 50
}
```

Comme promis voici le schéma d'explication :



Agrandir l'image

Ça fonctionne comme à la poste, le facteur a besoin de connaître votre adresse pour vous livrer votre colis. C'est la même chose pour votre machine, elle a besoin de connaître l'adresse de votre variable pour la manipuler.

les pointeurs

Déclarer et afficher un pointeur

Maintenant que vous avez compris qu'est-ce une adresse mémoire et comment y accéder, alors je peux passer à la suite en vous expliquant les pointeurs !

Les pointeurs sont des variables dont le contenu est une adresse mémoire d'une autre variable, c'est-à-dire l'adresse directe de l'**emplacement mémoire d'une variable**.

Vous devez déclarer un pointeur avant de pouvoir l'utiliser pour **stocker l'adresse d'une variable**. La forme générale d'une déclaration de variable de pointeur est :

```
var nom_de_votre_variable *type_de_votre_variable
```

Ici, `type_de_votre_variable` est le type de base du pointeur (int, float32, etc ...) et `nom_de_votre_variable` est le nom de la variable de pointeur. L'astérisque `*` permet d'indiquer à votre compilateur que votre variable s'agit bien d'un pointeur (ce signe est obligatoire).

Après la théorie, passant à un peu de pratique. Pour cet exemple je vais déclarer un pointeur et une variable et accéder directement à la valeur de ma variable depuis son adresse grâce à mon pointeur.

```
package main

import "fmt"

func main() {
    var a int = 20
    var ap *int // création d'un pointeur
    ap = &a // stockage de l'adresse mémoire de la variable dans notre pointeur

    fmt.Printf("Adresse de votre variable a : %p\n", &a)

    fmt.Printf("Valeur de votre variable (pointeur) ap : %p\n", ap)

    fmt.Printf("Valeur de l'adresse %p: %d\n", ap, *ap)
}
```

Résultat :

```
Adresse de votre variable: 0xc000056058
Valeur de votre variable (pointeur) ap : 0xc000056058
Valeur de l'adresse 0xc000056058: 20
```

Même si j'ai mis quelques commentaires dans le code, une petite explication du code s'impose :

```
var a int = 20
```

Ici rien de compliqué, je déclare une variable nommée `a` de type int.

```
var ap *int
```

La aussi rien de compliqué, je déclare juste un pointeur nommé `ap` de type int.

```
ap = &a
```

Petit rappel *"un pointeur ne peut stocker que des adresses mémoire"*. Ça tombe bien car ici grâce au signe `&` on peut **accéder à l'adresse mémoire** de notre variable `a`

```
fmt.Printf("Adresse de votre variable a : %p\n", &a)
```

Ici j'affiche tout simplement l'adresse mémoire de la variable `a`

```
fmt.Printf("Valeur de votre variable (pointeur) ap : %p\n", ap)
```

Sur cette étape j'affiche la valeur du pointeur `ap` pour vous démontrer qu'il possède bel et bien comme valeur l'adresse de votre variable `a`

```
fmt.Printf("Valeur de l'adresse %p: %d\n", ap, *ap)
```

Pour cette étape j'accède directement à la valeur de la variable `a` depuis son adresse grâce au pointeur `ap`.

L'astérisque `*` nous permet entre autres de déclarer un pointeur mais il permet aussi d'accéder à la valeur de la variable pointée, sur cette ligne de code on l'utilise pour accéder à la valeur contenue dans l'adresse mémoire de la variable `a`

Valeur par défaut d'un pointeur

La valeur par défaut d'un pointeur est `nil`. Ça peut vous aider pour savoir si votre pointeur comporte une adresse mémoire ou pas, exemple :

```
package main

import "fmt"

func main() {
    var ptr *int

    if ptr == nil { // est ce que le pointeur est vide ?
        fmt.Println("Aucune adresse mémoire")
    } else {
        fmt.Printf("Votre adresse mémoire est %p\n", ptr)
    }

    var b int = 20
    var pb *int
    pb = &b

    if pb == nil {
        fmt.Println("Aucune adresse mémoire")
    } else {
        fmt.Printf("Votre adresse mémoire est %p\n", pb)
    }
}
```

Résultat :

```
Aucune adresse mémoire
Votre adresse mémoire est 0xc000056058
```

Modifier la valeur d'une variable depuis un pointeur

Comme on peut accéder à l'adresse mémoire d'une variable depuis un pointeur alors il est tout à fait réalisable de modifier la valeur qui se trouve dans cette adresse mémoire. Voyons voir cela de plus près :

```
package main

import "fmt"
```

```
func main() {
 var a int = 20
 var ap *int
 ap = &a

 fmt.Printf("Valeur de la variable a : %d\n", a)

 *ap = 30 // modification de la valeur de la variable "a" depuis un pointeur

 fmt.Printf("Valeur de la variable a : %d\n", a)

}
```

Résultat :

```
Valeur de la variable a : 20
Valeur de la variable a : 30
```

Voici l'explication du code :

```
*ap = 30
```

Sur cette ligne de code, j'accède directement à l'adresse mémoire de la variable `a` grâce à mon pointeur `ap` , que je remplace par une nouvelle valeur (ici 30).

Les pointeurs dans les fonctions

Il est concevable de déclarer des pointeurs en tant que paramètres dans une fonction.

> **Attention**
>
> Il faut savoir qu'un changement de valeurs d'une variable actionné depuis un pointeur est irréversible car le pointeur va directement modifier la valeur depuis votre adresse mémoire.

Voici un exemple qui illustre bien ce que je viens de vous expliquer.

```
package main

import "fmt"

func main() {
    var a int = 20

    fmt.Printf("Avant de modifier la variable de a : %d\n", a)

    rajouterCinq(&a)

    fmt.Printf("Après modification par des pointeurs de la variable de a : %d\n", a)

    affichage(&a)
}

// fonction qui prend en paramètre un pointeur
func rajouterCinq(pa *int) {
    *pa = *pa + 5 // modification de la valeur de la variable "a" depuis un pointeur
}

// fonction qui prend en paramètre un pointeur
func affichage(pa *int) {
    fmt.Println("affichage da valeur dans une fonction :", *pa)
}
```

Résultat :

```
Avant de modifier la variable de a : 20
Après modification par des pointeurs de la variable de a : 25
affichage da valeur dans une fonction : 25
```

J'ai modifié la valeur de ma variable `a` depuis le pointeur défini en tant que paramètre sur la fonction `rajouterCinq()`, comme je l'ai expliqué plus haut, les changements sont **irréversibles** peu importe l'endroit où je l'affiche (fonction `main()` ou autres).

Conclusion

Les pointeurs sont des fonctionnalités de **bas niveau**, il était donc nécessaire de vous expliquer le fonctionnement de la mémoire d'un ordinateur pour que vous puissiez ensuite bien comprendre comment les utiliser correctement.

Sur le chapitre qui traitait sur la portée des variables, je vous avais expliqué qu'il n'était pas possible de modifier une variable locale hors de sa fonction ou de son bloc mais avec les pointeurs il devient possible de modifier une variable locale même si cette dernière est déclarée dans une autre fonction ou dans un autre fichier (package) et c'est vraiment là toute la puissance des pointeurs. On peut déclarer notre variable n'importe où dans notre code et la modifier sur n'importe quel l'endroit de notre code.

Vous aurez le plaisir d'utiliser les pointeurs quand vous travaillerez sur des plus gros projets, pour le moment mon but était juste de vous faire comprendre la notion et l'utilisation basique des pointeurs mais sachez qu'on les utilisera sur d'autres chapitres comme le chapitre sur les structures.

Si vous avez des questions n'hésitez pas à me les poser sur mon LinkedIn (je suis très actif dessus !) .