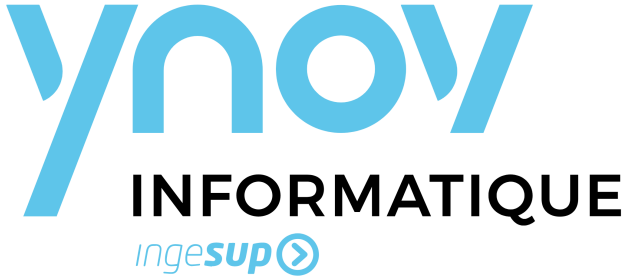
POINTERS



TOULOUSE **YNOV** CAMPUS

B1 - Filière **INFORMATIQUE**

09/11/20

Antoine ROQUES

La mémoire

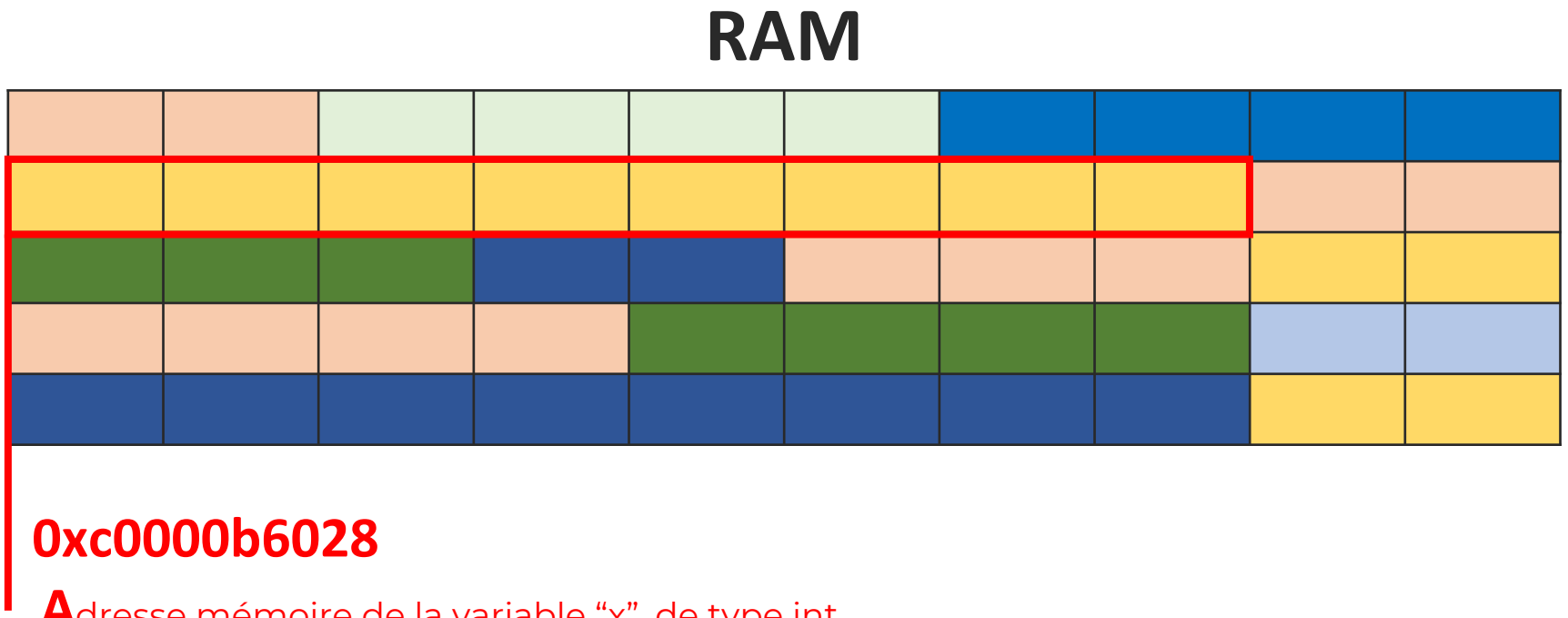
```go
func main() {

  // allocate an 8-bit memory area (X)
  var x int

  // allocate an 8-bit memory area and write the five value (Y)
  y := 5

}
```

La mémoire


RAM
0xc0000b6028
Adresse mémoire de la variable “x”, de type int

Les pointeurs

“Go has pointers. A pointer holds the memory address of a value.” - tour.golang.org

On **declare** un pointeur avec le symbole : *****

On **accède** à l’adresse mémoire d’une variable avec le symbole : **&**

On **accède** à la valeur d’un pointeur avec le symbole : *****

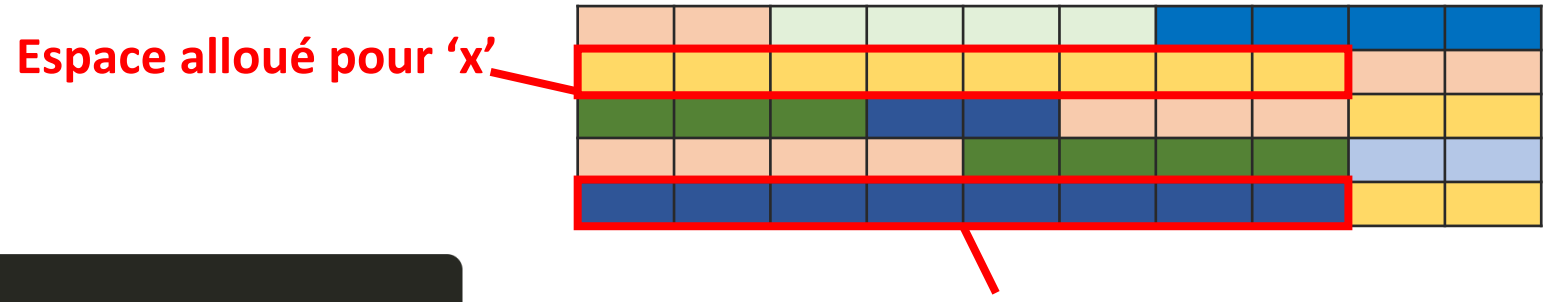
```
func main() {
  var myPointer *int
  var x int = 5

  myPointer = &x
  fmt.Println(x) // 5
  fmt.Println(myPointer) // 0xc00002c008
  fmt.Println(*myPointer) // 5
}
```

Les paramètres

GO utilise le **passage de paramètres par valeur**.
Il copie donc la valeur de la variable passée en paramètres.

=> On se retrouve donc avec deux variables ayant la même valeur, dans deux zones mémoire différentes.




```go
func main() {
    var x int = 5

    fmt.Println(x) // 5
    addOne(x)
    fmt.Println(x) // 5
}
```

```go
func addOne(y int) {
    y++
    fmt.Println(y) // 6
}
```

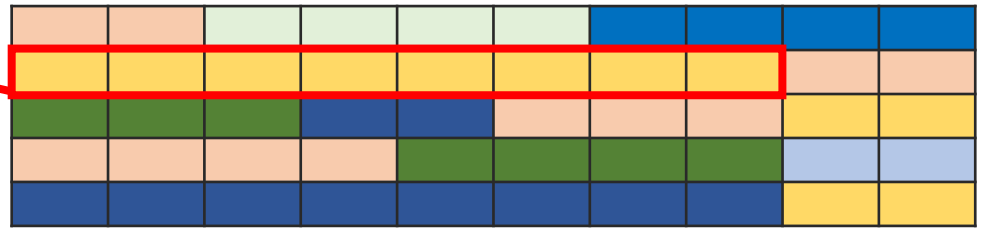
Les paramètres (avec pointeurs)

Les pointeurs permettent d'accéder à la zone mémoire d'une même variable.

```go
func main() {
    var x int = 5

    fmt.Println(x) // 5
    addOne(&x)
    fmt.Println(x) // 6
}
```

```go
func addOne(pX *int) {
    *pX++
    fmt.Println(*pX) // 6
}
```



STRUCTURES

TOULOUSE **YNOV** CAMPUS

B1 - Filière **INFORMATIQUE**

09/11/20

Antoine ROQUES

Les structures

“A struct is a collection of fields. - tour.golang.org

Une structure permet de créer un nouveau “type de variables” comprennant plusieurs champs.

Déclaration d’une structure :

```
type StructName struct {
    x int
    y int
}
```

Les structures

Initialisation d'une structure :

```
type User struct {
    lastName string
    firstNae string
    age int
}

func main() {
    me := User{"ROQUES", "Antoine", 74}
    fmt.Println(me) // {ROQUES Antoine 74}
}
```

Les structures

Les champs d'une structure sont accessibles de cette manière :

```
func main() {
    me := User{"ROQUES", "Antoine", 74}
    fmt.Println(me) // {ROQUES Antoine 74}

    me.age = 8
    fmt.Println(me.age) // 8
}
```

Les structures

Les pointeurs s'utilisent aussi avec les structures :

```
func main() {
    me := User{"ROQUES", "Antoine", 74}
    fmt.Println(me) // {ROQUES Antoine 74}

    happyBithday(&me)
    fmt.Println(me) // {ROQUES Antoine 75}
}
```

```
func happyBithday(user *User) {
    user.age++
}
```

Les méthodes

Les méthodes sont des fonctions propres à un type de structure.
Ces fonctions s'appellent depuis la structure directement.

```
func main() {
    me := User{"ROQUES", "Antoine", 74}
    me.hello()
}
```

```
func (user User) hello() {
    fmt.Println("Hello " + user.firstNae) // Hello Antoine
}
```

Les méthodes avec pointeurs

Les méthodes avec pointeurs permettent de modifier plusieurs champs d'une structure.

```go
func main() {
    president := User{"TRUMP", "Donald", 74}

    fmt.Println(president) // {TRUMP Donald 74}
    president.elections()
    fmt.Println(president) // {BIDEN Joe 77}
}
```

```go
func (user *User) elections() {
    user.lastName = "BIDEN"
    user.firstNae = "Joe"
    user.age = 77
}
```

TREES

TOULOUSE **YNOV** CAMPUS

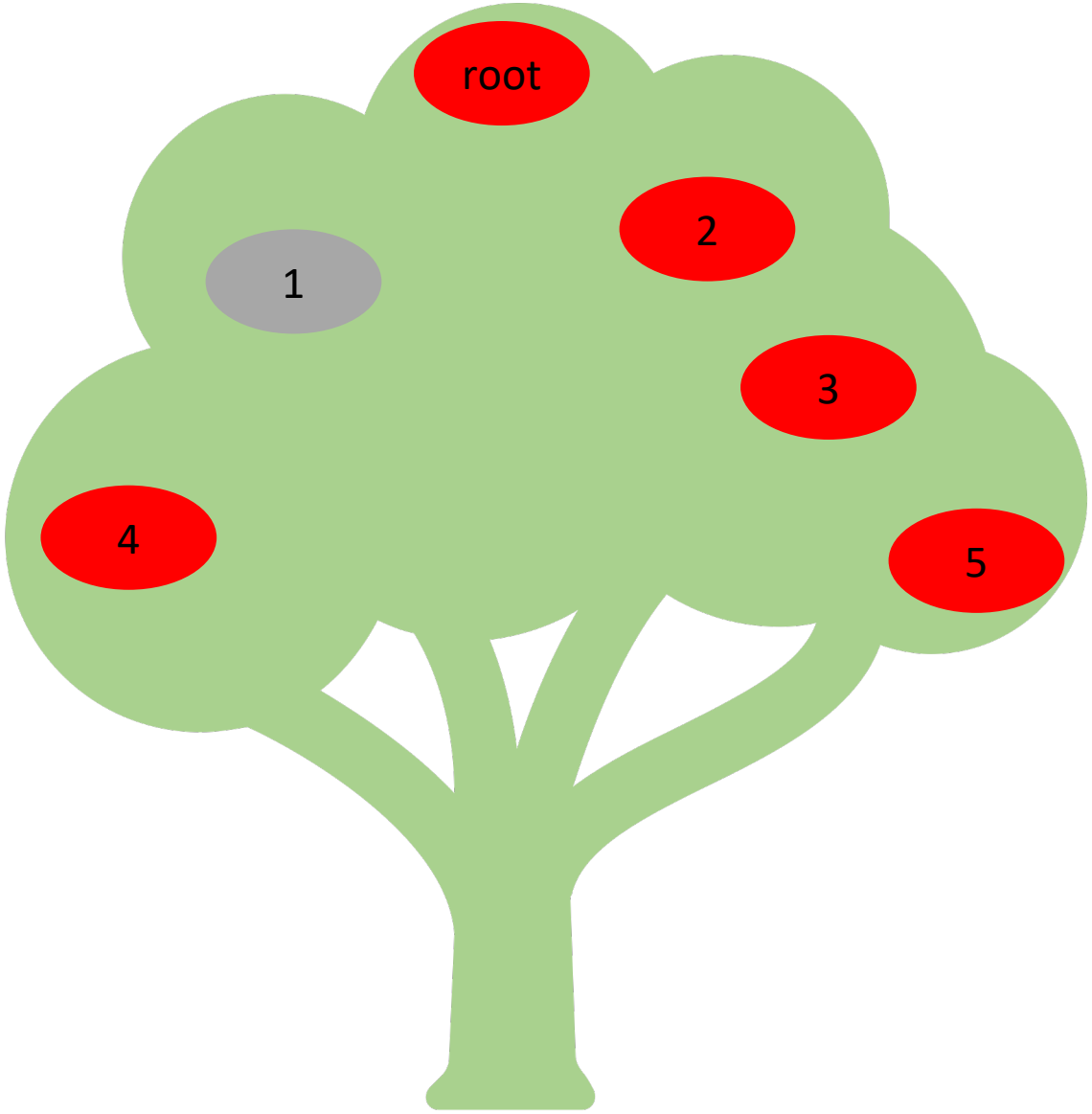
B1 - Filière **INFORMATIQUE**

09/11/20

Antoine ROQUES

Les arbres

Exemple avec un pommier :



Les arbres

Exemple avec un pommier :

```
type AppleBranch struct {
    LeftBranch  *AppleBranch
    HasApple    bool
    RightBranch *AppleBranch
}
```

```
func countApple(appleBranch *AppleBranch) int {
    localCount := 0
    if appleBranch == nil {
        return 0
    }

    if appleBranch.HasApple {
        localCount++
    }

    counterLeft := countApple(appleBranch.LeftBranch)
    counterRight := countApple(appleBranch.RightBranch)

    return counterLeft + counterRight + localCount
}
```

```
func main() {
    branch5 := AppleBranch{nil, true, nil}
    branch4 := AppleBranch{nil, true, nil}
    branch3 := AppleBranch{nil, true, &branch5}
    branch2 := AppleBranch{nil, true, &branch3}
    branch1 := AppleBranch{&branch4, false, nil}
    tree := AppleBranch{&branch1, true, &branch2}

    fmt.Println(countApple(&tree))
}
```

Les arbres

```
package main

import "fmt"

type AppleBranch struct {
	LeftBranch  *AppleBranch
	HasApple    bool
	RightBranch *AppleBranch
}

func countApple(appleBranch *AppleBranch) int {
	localCount := 0
	if appleBranch == nil {
		return 0
	}

	if appleBranch.HasApple {
		localCount++
	}

	counterLeft := countApple(appleBranch.LeftBranch)
	counterRight := countApple(appleBranch.RightBranch)

	return counterLeft + counterRight + localCount

}

func main() {
branch7 := AppleBranch{nil, false, nil}
branch6 := AppleBranch{nil, true, nil}
	branch5 := AppleBranch{&branch6, true, &branch7}
	branch4 := AppleBranch{nil, true, nil}
	branch3 := AppleBranch{nil, true, &branch5}
	branch2 := AppleBranch{nil, true, &branch3}
	branch1 := AppleBranch{&branch4, false, nil}
	tree := AppleBranch{&branch1, true, &branch2}

	fmt.Println(countApple(&tree))
}
```