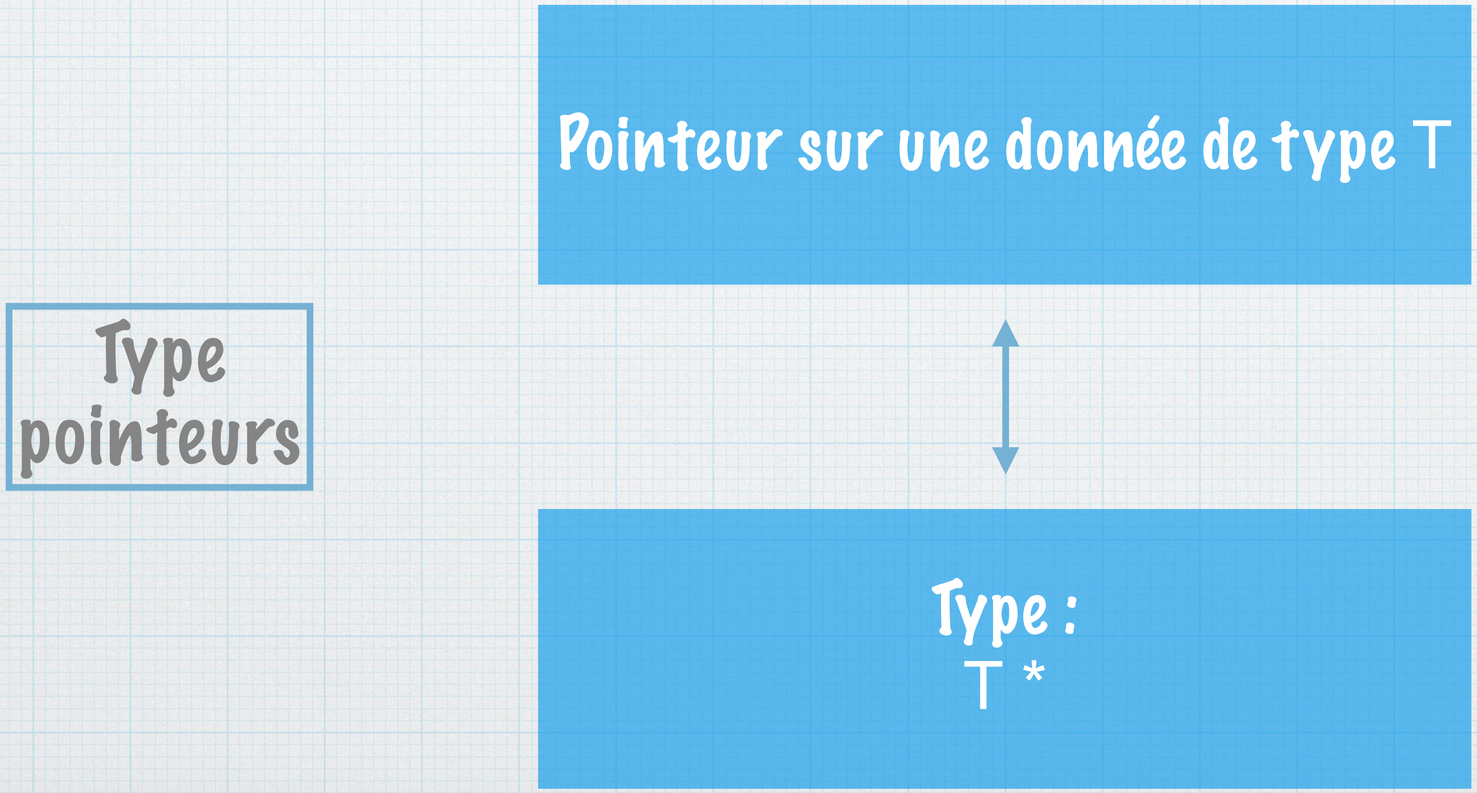
Introduction à la programmation (C)

Cours 4 : pointeurs et structures

Pointeur

Un pointeur est simplement une adresse mémoire.

Type pointeurs



Type pointeur sur un entier :
int *

Type pointeur sur un float :
float *

Pointeur

On peut alors définir des variables (normales) de type pointeur : les pointeurs.

```
int main ()
{
    int * pi ;
    char * pc ;
}
```

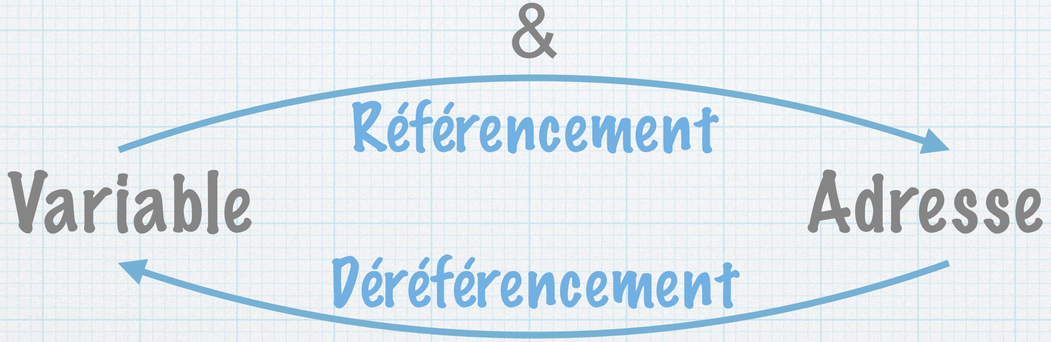
pi variable contenant des adresses d'entiers.

pc variable contenant des adresses de caractères.

Il est recommandé de les désigner par des noms commençant par « p »

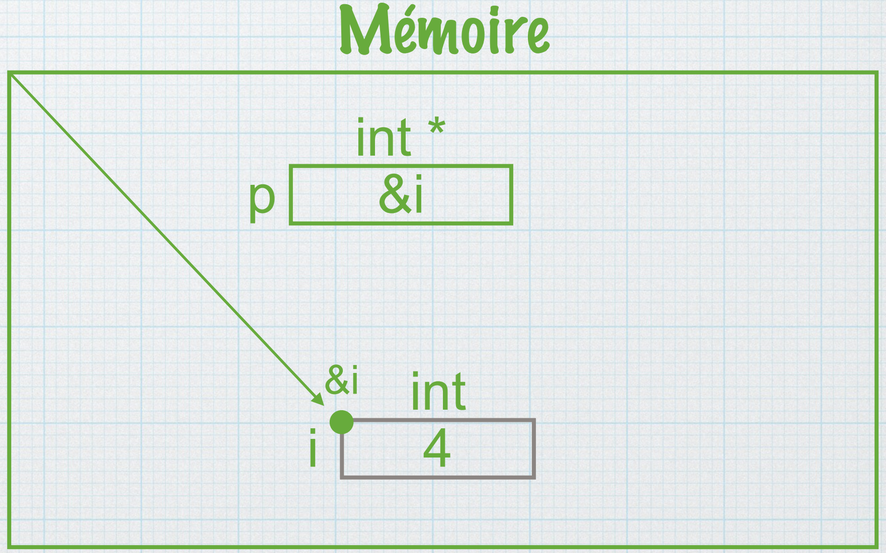
Manipulation d'adresses

Pour récupérer l'adresse d'une variable / à partir d'une adresse accéder à la donnée stockée :



* a deux sens :
→ après un type : type pointeur
→ avant une variable pointeur : déréférencement

```
int i = 4 ;
int * p = &i ;
*p = *p + 2 ;
```



Pointeurs et fonctions

Pour qu'une fonction puisse modifier une variable : on passe un pointeur sur cet variable comme argument.

```
void swap (int * p1, int * p2)
{
    int tmp = *p1 ;
    *p1 = *p2 ;
    *p2 = tmp ;
}
```

```
int main ()
{
    int i = 2, j = 3 ;
    swap (&i, &j)  ;
}
```

Cela ne vous rappelle pas scanf ?

Pointeurs et fonctions

Règle sur les arguments (v2 finale)

Pour qu'une fonction modifie une variable :

* On passe l'adresse de cette variable en argument (donc pointeur)

Si un tableau est passé en argument, comme un tableau est l'adresse de sa première case :

* La fonction peut modifier le contenu des cases

Pointeurs et tableaux (dynamiques)

En C :

Donc un tableau est un pointeur (cf. TP arithmétique des pointeurs)

```
float T[10] ;
```

T identique à &T[0]
*T identique à T[0]

T est de type float * (similaire à float [])

Pointeurs et tableaux (dynamiques)

La création (allocation) de tableaux dynamiques se fait par malloc :

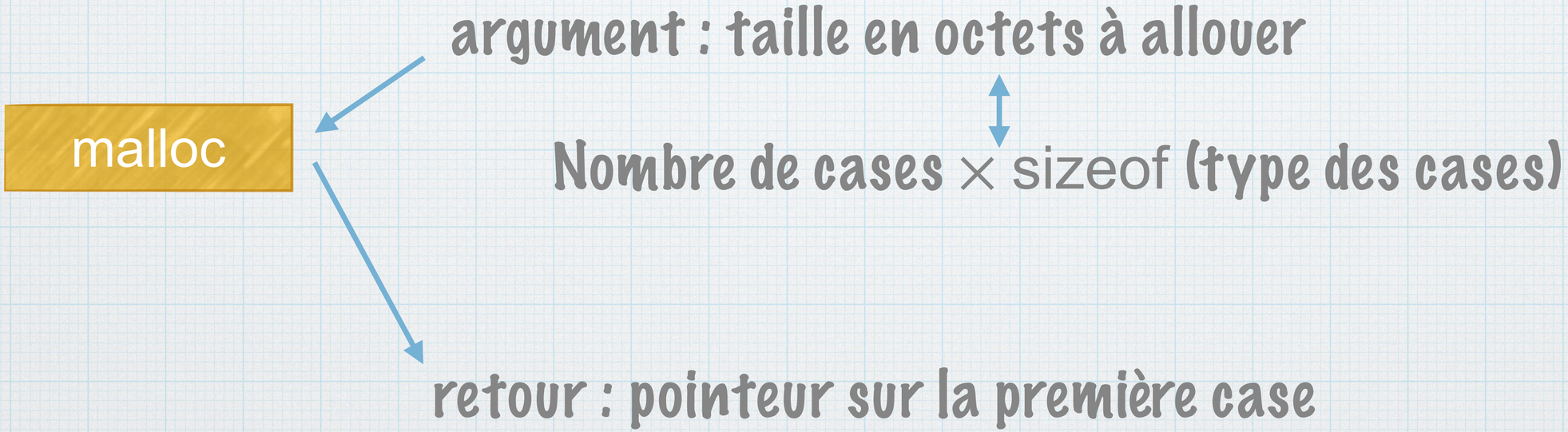


Tableau de n float :

```
float * T ;
T = malloc(n*sizeof(float)) ;
```

T est vraiment un tableau : T[0], T[1] …

Redéfinitions de types

Il est possible de redéfinir des types :

typedef ancien_type nouveau_type ;

Utile, par exemple, pour clarifier le code :

```
typedef int bool ;

bool tableau_egaux (float * T1, int n1, float * T2, int n2) ;
```

Structures

Les structures permettent de définir de nouveaux types (complexes) :

* Contenant des données de différents types (champs)
* Chacune est désignée par un nom ou label

```
struct  nom_structure {
    type1 label1 ;
    type2 label2 ;
    …
} ;
```

Définit le type
struct nom_structure

Accès aux champs :
variable.label

Définition à placer après les entêtes

```
struct couple {
    int left ;
    char right ; } ;
```

```
int main ()
{
    struct couple c ;
    c.left = 1 ;
    c.right = 'a' ;
    …
```

Structures et redéfinitions de types

```
struct  nom_structure {
    type1 label1 ;
    type2 label2 ;
    …
} ;
```

Définit le type

struct nom_structure

Trop long → redéfinition

```
struct couple_tmp {
    int left ;
    char right ; } ;
typedef   struct couple_tmp   couple ;

int main ()
{
    couple c ;
    c.left = 1 ;
    c.right = 'a' ;
    …
```