

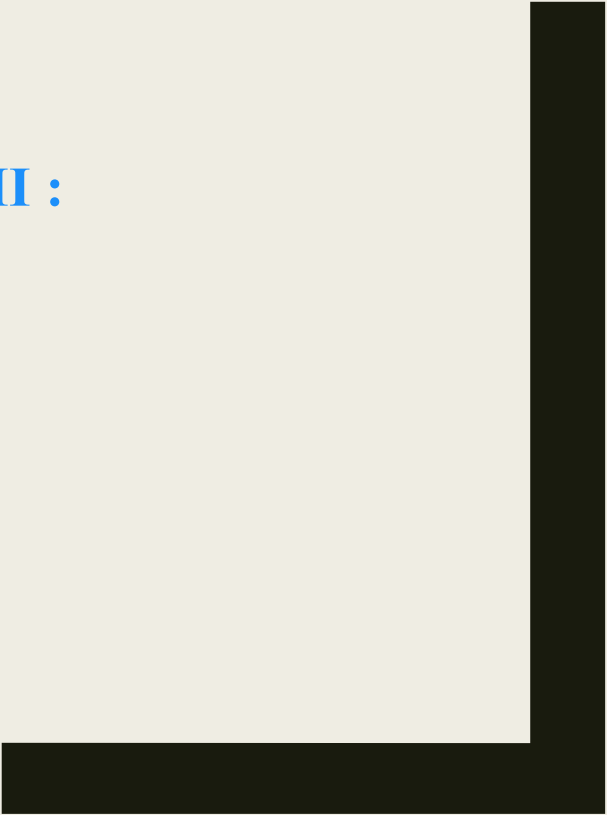


ATELIER PROGRAMMATION II :

LES POINTEURS

Monia Touil

LI1 – 2022 / 2023



Plan

❖ Introduction

❖ Les pointeurs

- ✓ Définition des pointeurs
- ✓ Arithmétique des pointeurs
- ✓ Allocation dynamique

❖ Pointeurs et tableaux



INTRODUCTION



Introduction

- **Toute variable** manipulée dans un programme est **stockée** quelque part en **mémoire centrale**.
- La mémoire peut être assimilée à un « tableau » dont chaque élément est identifié par une « adresse ».
- Pour retrouver une variable, il suffit, donc, de connaître l'adresse de l'élément-mémoire où elle est stockée.
- C'est le compilateur qui fait le lien entre l'identificateur d'une variable et son adresse en mémoire.
- Il peut être cependant plus intéressant de **décrire une variable** non plus par son identificateur mais directement par **son adresse** !

Lvalue : adresse et valeur

Définition

On appelle **Lvalue** (left value) toute **expression** du langage pouvant être placé à **gauche** d'un **opérateur d'affectation**.

Caractérisation

Une Lvalue est caractérisée par :

- ❑ **son adresse** : i.e., l'adresse mémoire à partir de laquelle l'objet est stocké ;
- ❑ **sa valeur** : i.e., ce qui est stocké à cette adresse.

Lvalue : exemple

```
int i, j;
```

```
i = 1;
```

```
j = i;
```

❖ Si le compilateur a placé la variable i à l'adresse 01 et j à l'adresse 05, alors :

Lvalue	Adresse	Valeur
i	01	1
j	05	1

Remarque

L'adresse d'une lvalue est **un entier** (16 bits, 32 bits ou 64 bits) et ce **quelque soit le type** de la valeur de lvalue.

Lvalue : récupérer l'adresse

L'opérateur adresse

Pour accéder à **l'adresse** d'une variable (lvalue) nous disposons de l'opérateur **&**

Par l'exemple :

```
i n t i ;
```

```
p r i n t f ( " l'adresse de i = %d " , & i ) ;
```

Si le compilateur a placé la variable `i` à l'adresse 4831830000 alors l'affichage sera :

l'adresse de i = 4831830000



LES POINTEURS



Notion de pointeur

Définition

Un pointeur est une **lvalue** dont la **valeur** est égale à l'**adresse** d'une autre lvalue.

Déclaration

type *nomPointeur; (où type est le type de l'élément pointé).

Exemple :

```
int i = 3;
```

```
int *p ;
```

```
p = &i ;
```

Lvalue	Adresse	Valeur
i	4830000	3
p	4830004	4830000

Pointeur : opérateur d'indirection (1/3)

Problème

Comment peut-on accéder directement à l'élément pointé par la valeur d'un pointeur ?

Exemple :

```
int i = 3;  
int *p ;  
p = &i ;
```

Lvalue	Adresse	Valeur
i	4830000	3
p	4830004	4830000

Solution

Utilisation d'un nouvel opérateur: *

Pointeur : opérateur d'indirection (2/3)

Exemple :

```
i nt i =3;  
i nt *p ;  
p = &i ;  
p r i n t f( "La valeur de *p = %d" , *p ) ;
```


Lvalue	Adresse	Valeur
i	<u>4830000</u>	<u>3</u>
p	4830004	4830000
*p	<u>4830000</u>	<u>3</u>

Pointeur : opérateur d'indirection (3/3)

```
int main (){  
    int i=3, j=6;  
    int *p0, *p1;  
    p0 = &i;  
    p1 = &j;  
}
```


Lvalue	Adresse	Valeur
i	4830000	3
j	4830004	6
p0	4835984	4830000
p1	4835980	4830004

**p0=*p1;*



Lvalue	Adresse	Valeur
i	4830000	6
j	4830004	6
p0	4835984	4830000
p1	4835980	4830004

p0=p1 ;



Lvalue	Adresse	Valeur
i	4830000	3
j	4830004	6
p0	4835984	4830004
p1	4835980	4830004

Arithmétique des pointeurs

- La valeur d'un pointeur est un entier.
- On peut appliquer à un pointeur quelque opérations arithmétiques :
 - **Addition** d'un entier à un pointeur.
 - **Soustraction** d'un entier à pointeur.
 - **Différence** entre deux pointeurs (de même type)

Arithmétique des pointeurs: l'addition et la soustraction

- Soit i un entier et p un **pointeur** sur un élément de type **type**: **type** $*p$;
- l'expression $p0 = p + i$ (resp. $p0 = p - i$) désigne un pointeur sur un élément de type **type**,
- la valeur de $p0$ est égale à la valeur de p incrémenté (resp. décrémenté) de $i * \text{sizeof}(\text{type})$

Si $\&i = 4830000$ alors :

Lvalue	Adresse	Valeur
i	4830000	5
$p0$	4830004	4830008
$p1$	4830008	4830000

$\leftarrow @i + 2 * \text{sizeof}(\text{int}) = \dots 0 + 2 * 4 = 8$
 $\leftarrow p0 - 2 * \text{sizeof}(\text{int}) = \dots 8 - 2 * 4 = 0$

```
int main () {  
    int i = 5;  
    int *p0, *p1;  
    p0 = &i + 2;  
    p1 = p0 - 2;  
}
```

Arithmétique des pointeurs: la différence

- Soit a et b des pointeurs sur des éléments de type type,
- l'expression a - b désigne un entier dont la valeur est : **(a - b)/sizeof (type)**,

Si `&i = 4830000` alors :

Lvalue	Adresse	Valeur
i	4830000	5
p0	4830004	4830008
p1	4830008	4830000
j	4830016	2

```
int main () {  
    int i =5;  
    int *p0 ,* p1 ;  
    p0 = &i + 2 ;  
    p1 = p0 - 2 ;  
    int j = p0 - p1;  
}
```

$(p0 - p1) / \text{sizeof}(\text{int}) = (8 - 0) / 4 = 2$

Initialisation d'un pointeur

- Avant toute utilisation, un pointeur doit être initialisé (sinon, il peut pointer sur n'importe quelle région de la mémoire !) :
- soit par l'affectation d'une valeur nulle à un pointeur : `p = NULL` ;
- soit par l'affectation de l'adresse d'une autre variable (lvalue) : `p = &i` ;
- soit par l'allocation dynamique d'un nouvel espace-mémoire...

Allocation dynamique : définition

Définition

L'allocation dynamique est l'opération qui consiste à réserver un espace-mémoire d'une taille définie.

- L'allocation dynamique en C se fait par la fonction de la librairie standard `stdlib.h` :

`char* malloc(nombreOctets)`

Allocation dynamique : exemple

```
#include <stdlib.h>
int main () {
    int i = 3 , *p ;
    p = ( int *) malloc ( sizeof ( int ) ) ;
    *p = i ;
}
```

Allocation dynamique : la libération mémoire

Définition

C'est l'opération qui consiste à libérer l'espace-mémoire alloué.

- En C, la libération mémoire se fait par l'intermédiaire de la fonction de la librairie standard `stdlib.h` :

`void free(nomPointeur)`



POINTEURS ET TABLEAUX



Pointeurs et tableaux à une dimension

A retenir

Tout tableau en C est un pointeur constant !

- Soit **int tab[N]** un tableau alors **tab** est un pointeur qui a comme valeur **&tab[0]**.

```
int main () {  
    int tab [ 5 ]= { 1 , 2 , 3 , 4 , 5 };  
    int i , *p ;  
    p = tab ;  
    for ( i=0 ; i <5 ; ++i )  
        printf ( "*(p+i) = %d = p [ i ] = %d = tab [ i ] = %d \n" , *( p+i ) , p [ i ] , tab [ i ] ) ;  
}
```

- ***(p+i)** représente le contenu de tab [i]
- **p+i** est l'adresse de tab [i]
- **p[i]** est la valeur de tab [i]