UNIVERSITÉ ALIOUNE DIOP DE BAMBEY
UFR SCIENCES APPLIQUÉES ET TECHNOLOGIE DE L'INFORMATION ET DE LA COMMUNICATION
DÉPARTEMENT INFORMATIQUE

Chapitre 5 - LES POINTEURS

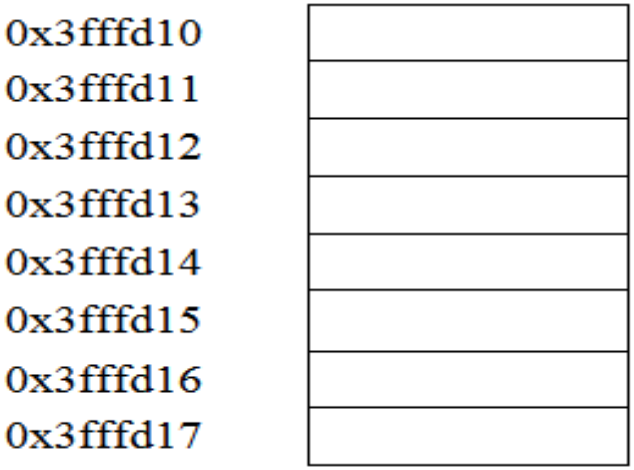
Programmation en LANGAGE C
AMRT 2
2015-2016

I. Introduction: notion d'adresse

La mémoire centrale utilisée par les programmes, est découpée en octets. Chacun de ces octets est identifié par un numéro séquentiel.

Par convention, ce numéro est noté en hexadécimal et précédé par 0x.

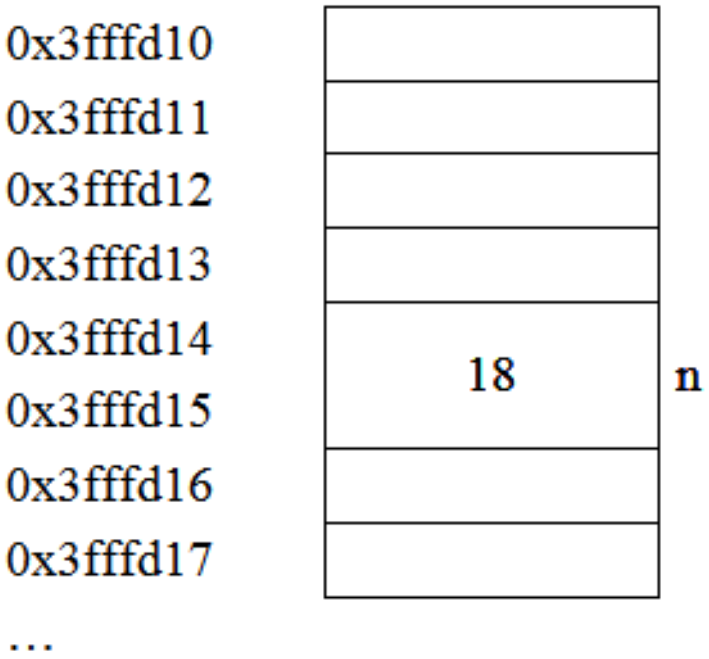
Exemple



Déclarer une variable consiste à réserver une zone en mémoire occupant un certain nombre d'octets (sa taille) et à laquelle on donne un nom. Le numéro du premier octet de cette zone correspond à l'adresse de la variable.

Exemple

short int n = 18;



Dans cet exemple, la valeur de la variable n est 18 et son adresse est 0x3fffd14.

Pour accéder à la valeur contenue dans une variable, on utilise tout simplement son nom.

Toutefois, il est parfois très pratique de manipuler une variable par son adresse.

II. Notion de pointeur

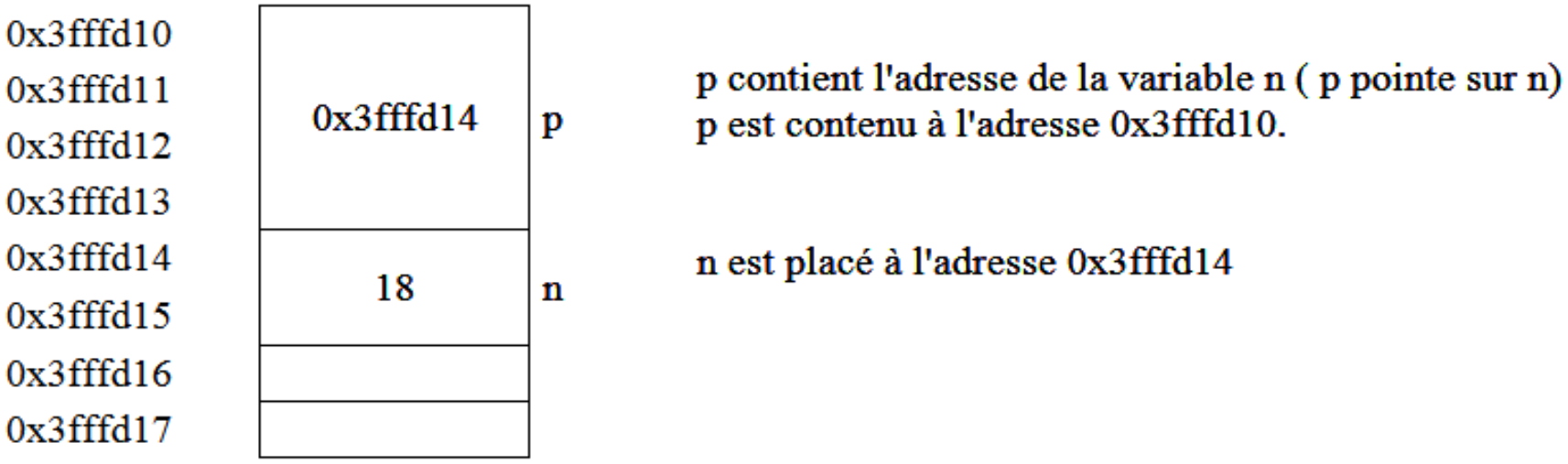
II-1. Définition

Un pointeur est une variable dont la valeur est l'adresse d'une autre variable.

On dit que le pointeur pointe sur la variable dont il contient l'adresse.

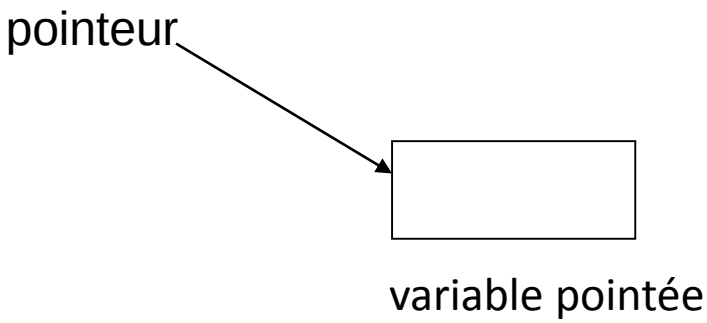
Un pointeur est associé à un type de variable sur lequel il peut pointer. Par exemple, un pointeur sur entier ne peut pointer que sur des variables entières.

Un pointeur est lui-même une variable et à ce titre il possède une adresse.

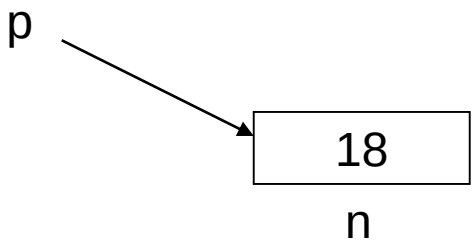


Il convient de ne pas confondre l'adresse de la variable pointeur et l'adresse contenue dans le pointeur (adresse de la variable pointée).

Représentation :



Exemple



II-2. Déclaration d'un pointeur

syntaxe :

```
type * nom_pointeur;
```

Exemple

```
int * p;
```

II-3. Manipulation d'un pointeur

Après avoir déclaré un pointeur, on peut l'initialiser en lui donnant l'adresse d'une variable existante.

Exemple

```
main()
{
int n;
int * p;

p = &n;
…  }
```

Il est possible d'accéder à la zone mémoire pointée en utilisant l'opérateur *. Ainsi, *p désigne la zone mémoire (**la variable**) pointée par le pointeur p.

Exemple

```
main()
{
int n = 33;
int *p;          //déclaration du pointeur p
p = &n;          // p pointe sur n
printf("%d", *p) ; //affiche 33 à l'écran, la valeur de n
*p = 34 ;        // n vaut maintenant 34
}
```

Remarque

Par défaut lorsque l'on déclare un pointeur, on ne sait pas sur quoi il pointe. Si on veut que le pointeur pointe nulle part, il faut l'initialiser à NULL.

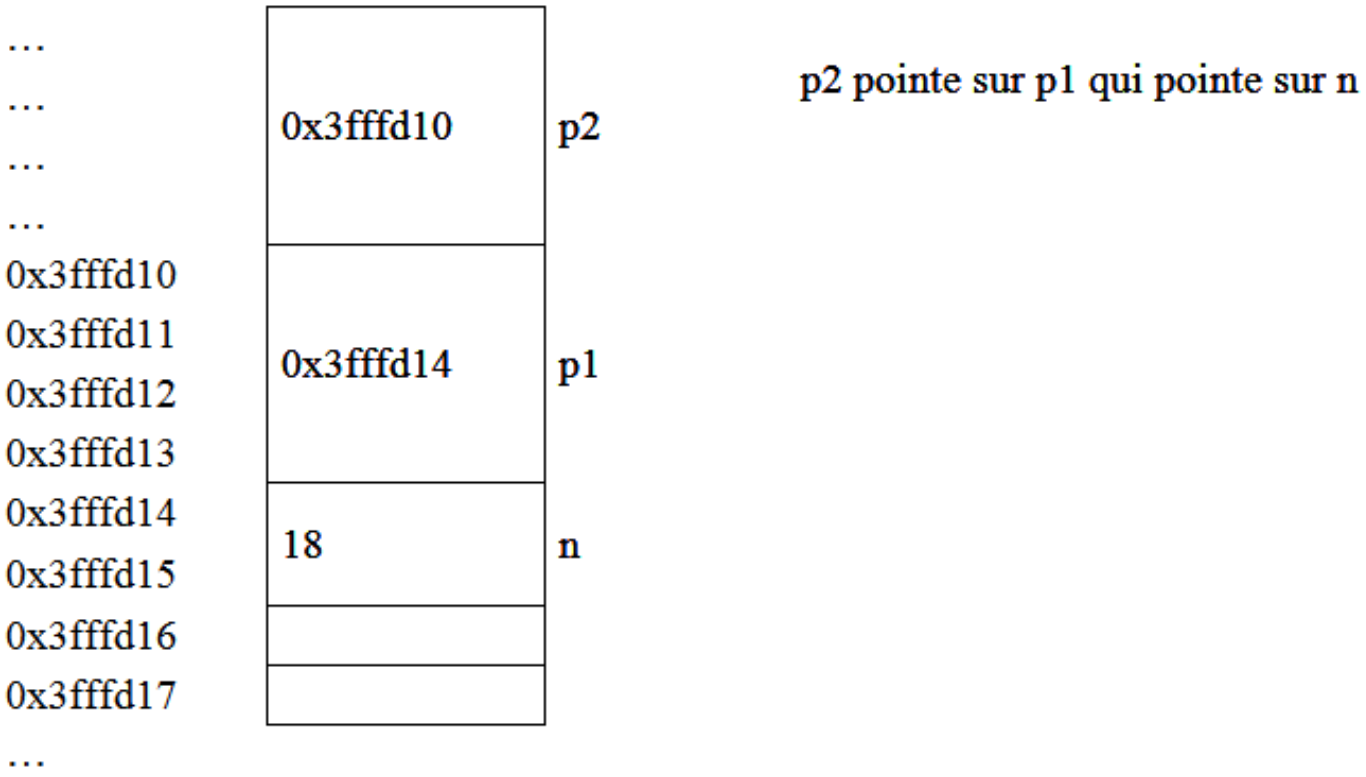
Exemple

```
int * p = NULL ;
```

II-4. Double indirection

Le fait qu'un pointeur pointe sur une variable s'appelle **indirection** (comme accès **indirect**). Quand un pointeur pointe sur un autre pointeur, il y a **double indirection**.

Exemple :



On peut accéder à n par p2 en utilisant deux fois l'opérateur *

* *p2 est équivalent à *p1 et à n

III. Allocation dynamique de mémoire

Au lieu d'affecter à un pointeur l'adresse d'une variable existante, on peut réserver de manière dynamique une nouvelle zone en mémoire et affecter au pointeur l'adresse de cette zone. Le pointeur va alors pointer sur cette zone.

En C, l'allocation dynamique de mémoire peut se faire avec les fonctions **malloc** et **calloc** de la bibliothèque standard **stdlib.h**:

- La fonction **malloc** prend comme paramètre la taille de la zone qu'on veut allouer et retourne, en cas de succès, l'adresse de la zone allouée.

Le type de l'adresse retournée par malloc est void*. Il faut alors le convertir au type du pointeur auquel on va l'affecter.

Par exemple, si on veut réserver un nouvel emplacement pour un caractère, on peut faire:

```
char *pcar;
pcar = (char *)malloc(sizeof(char ));
```

En cas d'échec, la fonction malloc retourne NULL.

- La fonction **calloc** prend en paramètre un nombre d'éléments nb et la taille t d'un élément et permet d'allouer une zone contiguë de nb éléments de taille t octets.

Elle retourne, en cas de succès, l'adresse de la zone allouée.

Le type de l'adresse retournée par calloc est void*. Il faut alors le convertir au type du pointeur auquel on va l'affecter.

Par exemple, si on veut allouer un emplacement pour 10 éléments de type int, on peut faire:

```
int * p;
p = (int *)calloc(10, sizeof(int));
```

En cas d'échec, la fonction calloc retourne NULL.

Remarque

La libération d'une zone mémoire allouée de manière dynamique n'est pas automatique en C. Une telle zone continue à occuper de l'espace mémoire jusqu'à la fin de l'exécution du programme si on ne la libère pas de manière explicite. Pour ce faire, il faut utiliser la fonction free de la bibliothèque standard stdio.h selon la syntaxe suivante:

```
free(nom_pointeur);
```

Exemple

L'instruction

```
free(p);
```

permet de libérer la zone mémoire pointée par p.

IV. Pointeurs et structures

Il est très courant d'utiliser un pointeur pour mémoriser l'adresse d'une variable structure.

Si p est un pointeur sur une structure, *p désigne la structure pointée.

On peut donc accéder à un champ de la structure pointée par l'expression : **(*p).champ**

L'usage de parenthèses est ici indispensable car l'opérateur * a une priorité moins élevée que l'opérateur point (.).

Cette notation peut être simplifiée grâce à l'opérateur →. L'expression précédente est strictement équivalente à :

p→ champ

Exemple

```
struct date
{
int jour;
int mois;
int annee;
};

struct date d;          // d est une variable structure
struct date * p;        //p est un pointeur sur une structure

p = &d          // p pointe sur la structure d
```

Alors d.jour est équivalent à p → jour

V. Arithmétique des pointeurs

Si i est un entier et p est un pointeur sur un objet de type unType, l'expression p + i désigne un pointeur sur un objet de type unType dont la valeur est égale à la valeur de p incrémentée de i*sizeof(unType).

Il en a de même pour la soustraction d'un entier à un pointeur, et pour les opérateurs d'incrémentation et de décrémentation ++ et --.

Ainsi, pour un pointeur sur un char, l'incrémentation de ce pointeur correspond à un décalage de 1 octet de la zone mémoire pointée, alors que pour un pointeur sur un float, la même incrémentation correspondra à un décalage de 4 octets.

Soient p1 et p2 deux pointeurs de même type. L'instruction

p1 = p2;

signifie qu'on copie dans p1 l'adresse contenue dans p2 et, donc, que p1 va pointer au même endroit que p2.

VI. Pointeurs et tableaux

L'usage des pointeurs en C est, en grande partie, orienté vers la manipulation des tableaux.

En C, le nom d'un tableau est en fait un pointeur sur le premier élément du tableau.

Par exemple, dans la déclaration

```
int tab[10];
```

tab est un **pointeur constant** dont la valeur est l'adresse du premier élément du tableau. Autrement dit, tab a pour valeur &tab[0]. Cette valeur n'est pas modifiable, néanmoins elle peut être utilisée pour initialiser un pointeur sur le premier élément du tableau.

Exemple

```
int tab[5]={1,2,3,4,5};
int * p = tab;
```

p pointe au même endroit que tab c'est à dire sur le premier élément du tableau

tab étant un pointeur sur le premier élément (élément d'indice 0) du tableau, tab+i donne le pointeur vers l'élément d'indice i du tableau.

Un tableau correspond en mémoire à un zone contiguë dont la taille est le produit du nombre d'éléments du tableau par l'espace nécessaire pour stocker un élément. On peut donc utiliser un pointeur pour accéder aux différents éléments d'un tableau.

Exemple Programme qui affiche un tableau d'entiers

```
#include <stdio.h>
#define N 5
main()
{
int tab[N] = {1, 2, 6, 0, 7};
int *p;
for (p=tab; p<=tab+N-1; p++)
printf(" %d \n",*p);
}
```

Remarque

On peut déclarer un tableau comme un pointeur sur le type de ses éléments.

Exemple

```
int * tab;
```

Mais dans ce cas, aucune place n'est réservée en mémoire pour le tableau et le pointeur tab n'est pas un pointeur constant.

Pour faire la réservation en mémoire, on peut utiliser les fonctions d'allocation dynamique **malloc** ou **calloc** dans la bibliothèque stdlib.h

Exemple

```
int * tab;
int nb=10;
tab = (int *)malloc(nb*sizeof(int));
```

ou bien

```
tab = (int *)calloc(nb, sizeof(int));
```

On peut ensuite utiliser tab comme un tableau normal. Par exemple, pour désigner l'élément i de ce tableau, on peut écrire tab[i] ou *(tab+i);

VII. Pointeurs et chaînes de caractères en C

On a vu précédemment qu'une chaîne de caractères était un tableau à une dimension d'éléments de type char, se terminant par le caractère nul ‘\0’.

Le nom d’un tableau étant un pointeur sur le premier élément, on peut donc manipuler toute chaîne de caractères à l'aide d'un pointeur sur un élément de type char.

Ainsi, il existe deux manières de déclarer une chaîne de caractères :

- Comme un tableau de caractères:

Exemple:

```
char ch[10];
```

Dans ce cas il y a réservation d’espace mémoire pour la chaîne et le nom de la chaîne (ch) est un pointeur constant et ne peut donc pas apparaître dans le membre gauche d'une affectation. On dit dans ce cas que l'espace mémoire nécessaire au stockage des caractères de la chaîne est réservé "statiquement".

- Comme un pointeur sur un caractère:

Exemple

```
char * ch;
```

Dans ce cas il n'y a pas de réservation d'espace mémoire pour la chaîne. Pour faire la réservation en mémoire, on peut utiliser les fonctions d'allocation dynamique **malloc** ou **calloc.**

Le pointeur ch n'est pas un pointeur constant et peut apparaître dans le membre gauche d'une affectation. On peut faire, par exemple, ch = "bonjour";