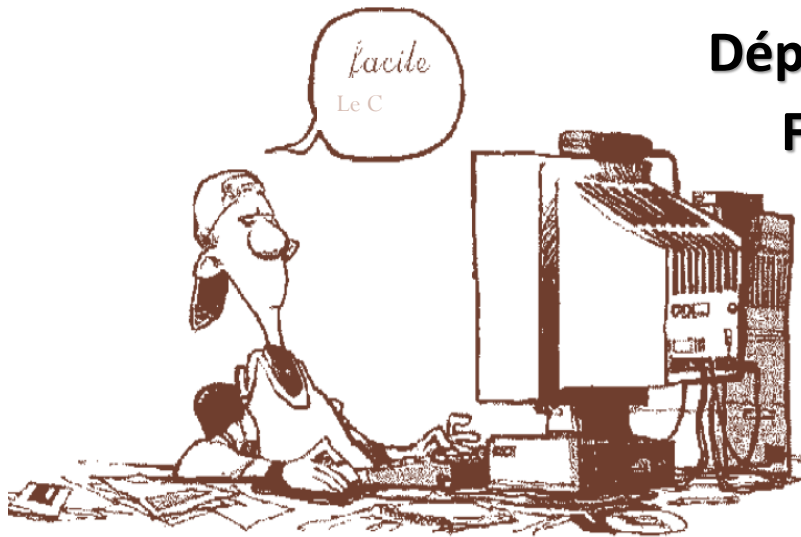




UNIVERSITE ABDELMALEK ESSAADI
ECOLE NATIONALE DES SCIENCES APPLIQUEES
TETOUAN



Module : Programmation Avancée en C



Département Génie Informatique
Filière : Génie Informatique

Pr. Jaber EL BOUHDIDI

Déroulement du Cours et Evaluation

Pré-Requis

- Algorithmique
- Programmation en Langage c

Evaluation

- 1 Contrôle
- Mini projet et Tp noté

NOTE		
Mini-projet (25%)	Examen (60%)	TPs(15%)

- Note Minimale requise pour valider le module est **12/20**

Programme

- **Rappel**
- **Les Pointeurs && Allocation dynamique**
- **Les Structures**
- **Les Fichiers**
- **Les Listes Chainées**
- **Les Piles et les Files**
- **Les Arbres**
- **Complexité**

Chapitre 1 : Les pointeurs

Les pointeurs

Définition de la variable

➡ Une variable est le nom d'un emplacement mémoire. Elle est caractérisée par : un nom, une adresse, un type et un contenu).

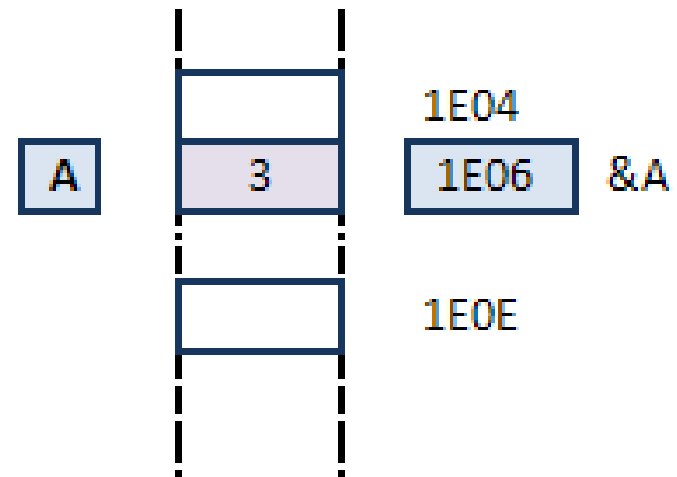
La mémoire d'un ordinateur peut être définie comme une suite de cases mémoire. Chaque case est repérée par un identificateur unique appelé *adresse* (une valeur en hexadécimal).

Les pointeurs

Définition de la variable

Exemple:

- ➡ Un nom : A
- ➡ Une adresse : 1E06
- ➡ Un type : Entier ou réel
- ➡ Un contenu (ou valeur) : 3



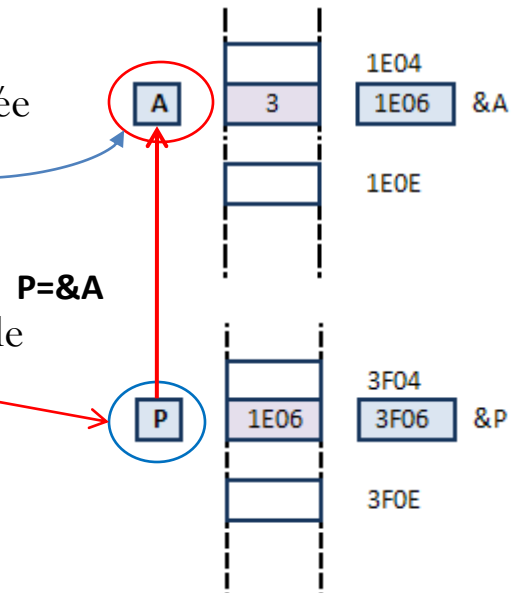
Les pointeurs

L'accès au contenu d'une variable

NB: l'accès au contenu d'une variable se fait directement ou indirectement

1. **Accès direct** : via le nom de la variable concernée

2. **Accès indirect** : via le nom d'un pointeur (variable qui contient l'adresse de la variable concernée)



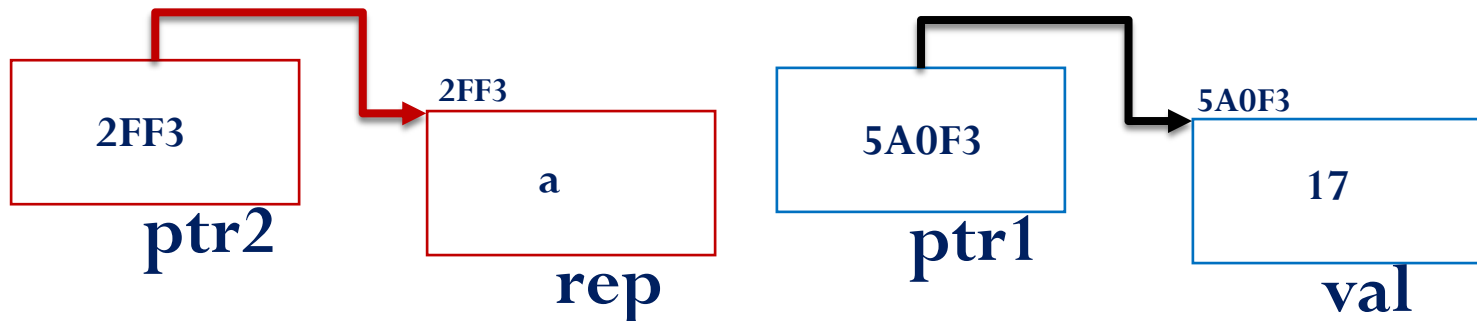
Les pointeurs

- **Définition**

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

- **Exemple :**

```
int val=17;  
char rep='a';  
int *ptr1;      /* déclaration d'un pointeur sur entier */  
char *ptr2;     /* déclaration d'un pointeur sur caractère */  
ptr1=&val;      /* ptr1 contiendra l'adresse de val */  
ptr2=&rep;      /* ptr2 contiendra l'adresse de rep */
```




Les pointeurs

Exemple de gestion d'un pointeur

➤ Déclaration	<code>int *P;</code>	P pointe sur un entier → pas de variable pointée car le pointeur n'est pas encore initialisé
➤ Initialisation	<code>int x=3 ; P=&x;</code>	P pointe sur la variable x → ✓ P contient l'adresse de la variable x ✓ *P contient la valeur de la variable x
➤ Déclaration et initialisation	<code>int *m=&x ;</code>	✓ m est un pointeur sur une variable entière x ✓ x est une variable entière pointée par m
<ul style="list-style-type: none">- L'opérateur & : adresse de → utilisé pour obtenir l'adresse d'une variable en mémoire- L'opérateur * : contenu de → utilisé pour obtenir le contenu de la variable pointée		

```
#include <stdio.h>
main() {
int x=3;
int *Pi;
Pi=&x;
printf("%d %d %X %X %X",x,*Pi,Pi,&x,&Pi);
}
```

Exécution:



```
3 3 12FF50 12FF50 12FF4C
```

Les pointeurs

Exemple :

```
int X=4,Y;
```

```
int *P,*q;
```

```
P=&X ; /* P contient l'adresse de X*/
```

printf("%d",*P) : affiche le contenu de la variable pointée par **P**(c.-à-d. la valeur de **X**).

```
Y=*P-1 ; /* Y vaut 3 */
```

```
*P= *P +1 ; /* incrémente X de 1 , X vaut 5*/
```

```
*P++; /* incrémente aussi de 1 la variable pointée par P , X vaut 6*/
```

N.B : les parenthèses sont importantes. Car *P++ incrémente le pointeur P (l'adresse) et non pas la valeur de X.

Les pointeurs

Exercice 1:

Qu'affiche le programme suivant :

```
main()
{
    int n=5,m=3,*p;
    p=&n;           printf("n=%d   m=%d   \n",n,m);

    (*p)++;         printf("n=%d   m=%d   \n",n,m);

    n++;            printf("n=%d   m=%d   \n",n,m);

    p=&m;            printf("n=%d   m=%d   \n",n,m);

    m+=n;           printf("n=%d   m=%d   \n",n,m);

    n+=(*p)++;      printf("n=%d   m=%d   \n",n,m);

    *p=7;           printf("n=%d   m=%d   \n",n,m);
}
```

Les pointeurs

Exercice 2 :

On considère un programme contenant les instructions suivantes, complétez le tableau suivant:

	A	B	C	P1	P2
Init.	1	2	3	/	/
P1=&A	1	2	3	&A	/
P2=&C					
*P1=(*P2)++					
P1=P2					
P2=&B					
*P1-=*P2					
++*P2					
P1=*P2					
A=++*P2**P1					
P1=&A					
*P2=*P1/=*P2					

Les Données Constantes

Déclaration d'une constante :

```
const int a=5;
```

Important !!

- Toute tentative de modification du contenu d'une variable déclarée **const** génère une erreur à la compilation .
- Cette restriction entraîne qu'il n'est pas possible de définir un pointeur normal sur une constante
- De même si on souhaite que la valeur d'un pointeur ne soit pas modifiée au cours d'un programme, il faut le définir constant.

Les Données Constantes

Exemples :

```
const int taille=1000 ;
```

```
int compteur ;
```

```
int *p1=&taille ;
```

```
const int *p2=&taille ;
```

```
const int *const p3=&taille ;
```

```
*p2=1010 ;
```

```
p2=&compteur ;
```

```
*p2=10 ;
```

```
compteur=10 ;
```

```
p3=&compteur ;
```

Les pointeurs & les tableaux

Les pointeurs ont une relation très étroite avec les tableaux ils facilitent beaucoup plus leur manipulation.

Exemple :

Soit A un tableau contenant des éléments du type **float**
float A[20], X;

Comme A représente l'adresse de A[0],

***(A+1)** désigne le contenu de A[1]

***(A+2)** désigne le contenu de A[2]

...

***(A+i)** désigne le contenu de A[i]

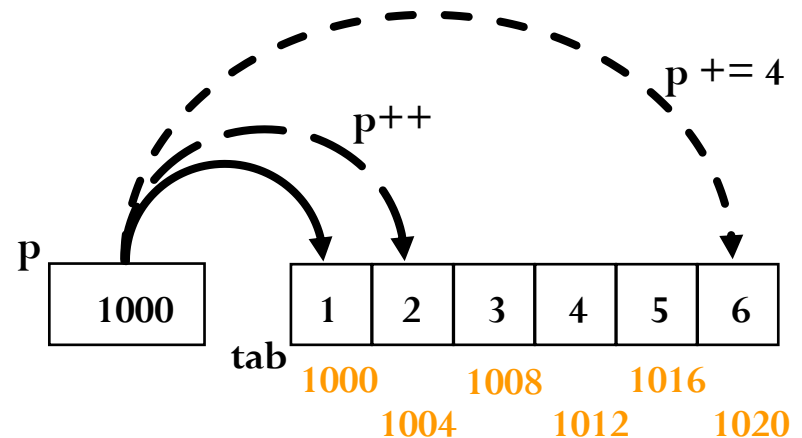
Alors, les écritures suivantes sont équivalentes :

*tab	tab[0]
tab	&tab[0]
*(tab+i)	tab[i]
(tab+i)	&tab[i]

Les pointeurs & les tableaux

Exemple:

```
int    tab[6] = { 1,2,3,4,5,6 };  
int    *p;  
p = tab;  
printf("%d\n", *p);  
p++;  
printf("%d\n", *p);  
p += 4;  
printf("%d\n", *p);
```



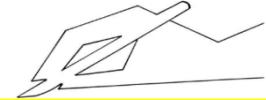
Qu'affiche ce programme?



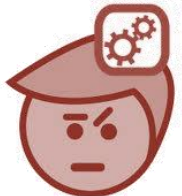
```
1  
2  
6
```


Les pointeurs & les tableaux

EXERCICES



1. En utilisant la notation pointeur, Ecrivez un programme qui range les éléments d'un tableau d'entiers T dans l'ordre inverse.
2. Ecrire un programme qui lit un entier X et un tableau des entiers et élimine toutes les occurrences de X dans T.
→ Utiliser les pointeurs pour parcourir le tableau T.



Les pointeurs & les tableaux

- Allocation dynamique:

La déclaration d'un tableau réserve automatiquement un espace mémoire.

Pour le cas des pointeurs, il faut impérativement utiliser une fonction pour allouer dynamiquement un espace. Comme la fonction **malloc** de la bibliothèque **<malloc.h>**.

- Exemple :

```
int *tab ;  
tab=(int *)malloc(sizeof(int) *10) ; // en C++ int *T=new int[10];
```

Cette instruction réserve un espace mémoire pour dix entiers.

Ou bien

```
int *tab, N;  
tab=(int *)malloc(sizeof(int) *N) ;
```

Avec N la taille du tableau.

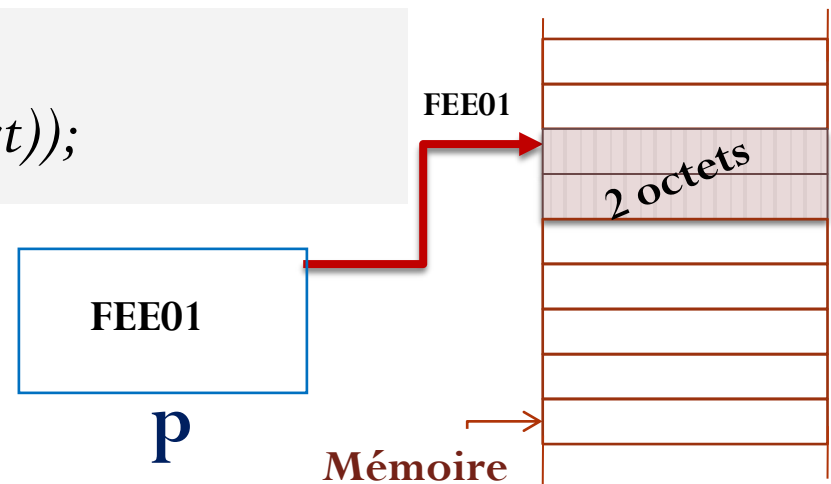
Allocation dynamique

Utilisation de la fonction **malloc** de la bibliothèque **<malloc.h>**.
malloc (nombre_octets)

- Exemple 1:

```
short* p;
```

```
p = (short*)malloc(sizeof(short));
```



Les pointeurs & les tableaux

- Allocation dynamique:

Exemple 2:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
    int i = 3;
    int j = 6;
    int *p;
    p = (int*)malloc(2*sizeof(int));
    *p = i;
    *(p+1) = j;
    printf("p = %ld \t *p = %d \t p+1 = %ld \t *(P+1) = %d\n",p,*p,p+1,*(p+1));
}
```

Qu'affiche ce
programme???

```
p = 4012320      *p = 3          p+1 = 4012324      *(P+1) = 6
Appuyez sur une touche pour continuer...
```

Les pointeurs & les tableaux

- La fonction calloc

La fonction `calloc` fournie par la biblio `stdlib.h` a le même rôle que le `malloc` mais initialise en plus `*p` à zéro :

`calloc(nb-elements, taille-element)`

Exemple

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

est équivalent à :

```
int * p;  
int i;  
p = (int *)malloc(N * sizeof(int));
```

Les pointeurs & les tableaux

- La fonction realloc

```
int * tab;  
tab = (int *)calloc ( 3 , sizeof(int) );  
tab[0] = 1; tab[1] = 2; tab[2] = 3;
```

Augmenter la taille du tableau

```
tab= (int *) realloc (tab, 4 * sizeof(int) );
```

Les pointeurs & les tableaux

- Libération de l'espace alloué

➔ Il faut **libérer** chaque **espace alloué dynamiquement** lorsqu'il **n'est plus utilisé** dans le programme

➔ La libération est effectuée à l'aide de la fonction **free()** de l'en-tête **stdlib.h** :

free(nom-du-pointeur)
en C++ : delete(nom-du-pointeur)

- Exemple :

```
int      *p1 = (int*)malloc(3 *sizeof(int));
int      *p2 = (int*)calloc(3,sizeof(int));
if (p1 == p2)
    free(p1);
else
    {   free(p1); free(p2); }
```

Les pointeurs & chaînes de caractères

- Les chaînes de caractères (Déclaration):

- ➔ Les chaînes de caractères sont des **tableaux** de **caractères**.
- ➔ Leur manipulation est donc analogue à celle d'un tableau à une dimension:

Déclaration:

```
char nom[dim];
```

ou bien

```
char *nom;  
nom = (char*)malloc(dim*sizeof(char));
```

Exemple:

```
char texte[10];
```

ou bien

```
char *texte;  
texte = (char*)malloc(10*sizeof(char));
```


Les pointeurs & chaînes de caractères

- Les chaînes de caractères (Affichage):

AFFICHAGE

➔ On peut utiliser la fonction **printf** et le format %s:

```
char texte[10] = "BONJOUR";  
printf("VOICI LE TEXTE: %s\n", texte);
```

➔ On utilisera aussi la fonction **puts** non formatée:

puts(texte);		printf("%s\n", texte);
putchar(c);		printf("%c\n", c);

Les pointeurs & chaînes de caractères

- Les chaînes de caractères (Lecture):

LECTURE

➔ On peut utiliser la fonction **scanf** et le format %s:

```
char texte[10] ;  
scanf("%s ",texte);
```

➔ On utilisera aussi la fonction **gets** non formatée:

gets(texte);	↔	scanf(" %s ",texte);
c=getchar();	↔	scanf(" %c",&c);

Les pointeurs & chaînes de caractères

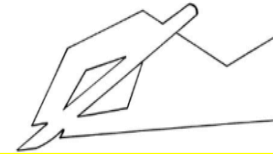
Exemple:

```
#include<stdio.h>
#include<malloc.h>
int main()
{
    int ln=0;
    char *ch;
    char *p;
    ch=(char *)malloc(sizeof(char)*20);
    printf("entrer votre chaine ");
    gets(ch);
    for(p=ch;*p;p++,ln++);

    printf("la longueur de la chaine est %d",ln);
    getchar();
    return 0;
}
```

Les pointeurs & chaînes de caractères

EXERCICES



- Les chaînes de caractères:

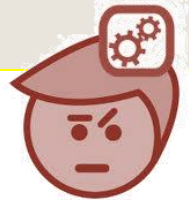
Exercice 1 :

Ecrire un programme qui compte le nombre de lettres majuscules dans une chaîne de caractères entrée par l'utilisateur, en utilisant le formalisme pointeur.

Exercice 2:

Ecrire un programme qui teste si une chaîne de caractères entrée par l'utilisateur est palindrome (une chaîne de caractère est dite palindrome si elle se lit de droite comme de gauche), en utilisant le formalisme pointeur.

Exemple : NON, RESSASSER.



Les pointeurs & chaînes de caractères

• Traitement des chaînes de caractères:

La bibliothèque `<string>` fournit une multitude de fonctions pratiques pour le **traitement** de **chaînes** de **caractères**.

strlen(<s>)	fournit la longueur de la chaîne <i>sans</i> compter le '\0' final
strcpy(<s>, <t>)	copie <t> vers <s>
strcat(<s>, <t>)	ajoute <t> à la fin de <s>
strcmp(<s>, <t>)	renvoie un nombre: - <i>positif</i> si la chaîne1 est <i>supérieure</i> à la chaîne2 (au sens de l'ordre alphabétique) - <i>négatif</i> si la chaîne1 est <i>inférieure</i> à la chaîne2 - <i>nul</i> si les chaînes sont <i>identiques</i> .
strncpy(<s>, <t>, <n>)	copie <n> premiers caractères de <t> vers <s>
strncat(<s>, <t>, <n>)	ajoute au plus <n> caractères de <t> à la fin de <s>

Les pointeurs & chaînes de caractères

- Les chaînes de caractères (Conversion):

La bibliothèque `<stdlib>` contient des déclarations de fonctions pour la **conversion de nombres en chaînes de caractères** et **vice-versa**.

Chaîne → Nombre

atoi(<s>)	retourne la valeur numérique représentée par <s> comme int
atol(<s>)	retourne la valeur numérique représentée par <s> comme long
atof(<s>)	retourne la valeur numérique représentée par <s> comme double

Nombre → Chaîne

itoa (<n_int>, <s>,)	convertit son premier argument en une chaîne de caractères qui sera ensuite attribuée à <s>. La conversion se fait dans la base .
---	--

Les pointeurs & chaînes de caractères

EXERCICES

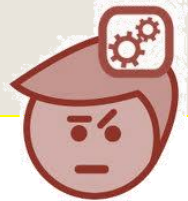
- Les chaînes de caractères:

1. Ecrire un programme qui demande le nom et le prénom de l'utilisateur et qui affiche alors la longueur totale du nom sans compter les espaces. Utiliser la fonction strlen et deux variables.

Exemple:

```
Introduisez votre nom et votre prénom: Ahmed Jalali
Bonjour Ahmed Jalali !
Votre nom est composé de 11 lettres
```

2. Ecrire un programme qui lit deux chaînes de caractères CH1 et CH2 et qui copie la première moitié de CH1 et la première moitié de CH2 dans une troisième chaîne CH3. Afficher le résultat.



Passage des arguments d'une fonction

Exercice :

Écrire une fonction de type **void** permettant d'échanger la valeur de deux entiers.

Ecrire un programme principal qui fait appel à cette fonction.

Est-ce que les deux valeurs ont été vraiment permutées?

Solution :

```
void echanger(int x,int y)
{int z;
  z=x;
  x=y;
  y=z;
  printf("x= %d y=%d ",x,y);
}
int main()
{
  int x=5,y=7;
  printf("x= %d y=%d ",x,y);
  echanger(x,y);
  printf("x= %d y=%d ",x,y);
  getch();
  return(0);
}
```

Mais le résultat est toujours: x=5, y=7

Passage des arguments d'une fonction

Passage par valeur :

Les paramètres sont passés par copie de valeur et ne seront pas modifiés en dehors de la fonction(fonction main par exemple).

Passage par adresse :

Pour pouvoir échanger les paramètres x et y de la fonction principale, il faut faire le *passage par adresse*. Ce sont les pointeurs qui offrent la possibilité de modifier les contenus des variables de la fonction principale (*fonction main()*).

Exemple :

```
void echanger(int *x, int *y)
{
    int z; z=*x; *x=*y; *y=z;
    printf("x= %d y=%d ",*x,*y);
}

int main()
{
    int x=5,y=7;
    echanger(&x,&y);
    printf("x= %d y=%d ",x,y);
    getch();
    return 0;
}
```

Fonctions récursives

Itération

```
procédure Itération( )  
Tantque (condition) faire  
    <Instructions>  
fin tantque
```

```
fonction S(n) : entier  
    S := 0  
    Tant que (n > 0) faire  
        S := S + n  
        n := n - 1  
    fin tant que  
    retourner S
```

Récursivité

```
procédure Itération( )  
Si (condition) alors  
    <Instructions>  
    Itération()  
fin si
```

```
fonction Somme(n) : entier  
Si (n = 0) alors  
    retourner 0  
sinon  
    retourner Somme(n-1) + n  
fin si
```

Fonctions récursives

▶ Exemple 1 : calcul de la factorielle

Équations de récurrences :

▶ $0! = 1$ (base)

▶ $n! = n(n-1)!$ (récurrence)

```
int fact( int n){  
    if (n == 0) || (n == 1)  
        return 1;  
    else  
        return n * fact(n-1);  
}
```

* Que se passe-t-il si $n < 0$?
⇒ *récursivité infinie*

Fonctions récursives

▶ Exemple 2 : suite de fibonacci

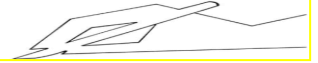
Équations de récurrences :

- ▶ $u(0) = 0, u(1) = 1$ (Base)
- ▶ $u(n) = u(n-1) + u(n-2), n > 1$ (récurrence)

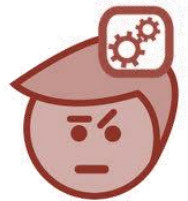
```
int fibo(int n) {  
    if((n==0) || (n==1))  
        return n;  
    else  
        return fibo(n-1) + fibo(n-2);  
}
```

Fonctions récursives

EXERCICES



Ecrire une fonction récursive qui cherche une lettre dans une chaîne de caractères .



Chapitre 2: Les structures

Les Structures

- ➔ Une **structure** est une collection de plusieurs variables (**champs**) groupées dans un ensemble pour un traitement commode,
- ➔ Les variables d'une structure sont appelées **membres** et peuvent être de n'importe quel type (tableaux, pointeurs, entiers)

```
struct Membre
{
    char    nom[80];
    char    adresse[200];
    int     *numero;
    float    amende[10];
};
```

Les Structures

Déclaration de Structures

→ Déclaration de la structure

```
struct produit
{
    int code;
    int qte ;
    float prix ;
} ;
```

→ Déclaration des variables de la structure

```
struct produit prd1 ;
struct produit prd1, prd2 ;
```

→ Déclaration de structure et variables de la même structure :

```
struct produit
{
    int code;
    int qte ;
    float prix ;
} prd1,prd2 ;
```


Les Structures

Déclaration de Structures

➔ Déclaration de la structure

```
typedef struct
{
    int code ;
    int qte ;
    float prix ;
} Enregistrement ;
Enregistrement prd1, prd2 ;
```

METHODE 3

Les Structures

Utilisation des champs d'une structure

```
struct produit
{
    int code;
    int qte ;
    float prix ;
} ;
struct produit prd1, prd2 ;
```

```
prd1.code = 2015 ;
```

affecte la valeur 2015 au champ code de la structure prd1.

```
printf ("%f",prd1.prix) ;
```

affiche la valeur du champ prix de la structure prd1.

```
scanf ("%f",&prd2.prix) ;
```

Lit une valeur qui sera affectée au champ prix de la structure prd2.

➔ *Notez bien la présence de l'opérateur &.*

Les Structures

Utilisation des champs d'une structure

```
struct produit
{
    int code;
    int qte ;
    float prix ;
} ;
struct produit prd1, prd2 ;
```

`prd1=prd2;`

➔ Possible pour le cas de structures de même type.

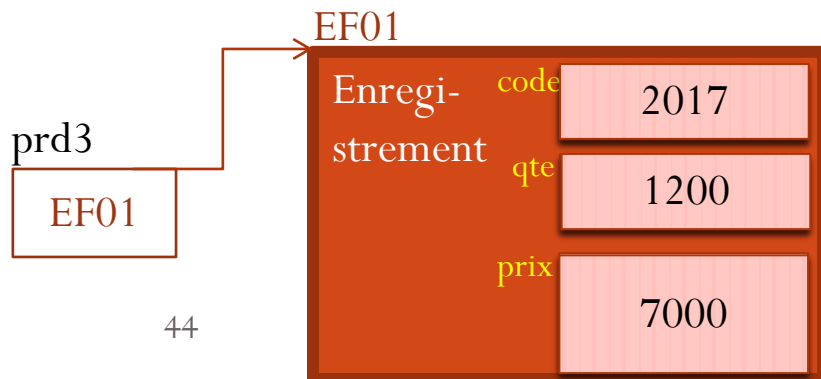
➔ Ça vient a remplacer :

```
prd1.code = prd2.code;
prd1.qte  = prd2.qte ;
prd1.prix = prd2.prix ;
```

Les Structures

Structure et Pointeur

```
typedef struct
{
    int code ;
    int qte ;
    float prix ;
} Enregistrement ;
Enregistrement *prd3 ;
```



→ L'accès aux membres de la structure pointée par `prd3` se fait de la manière suivante:

```
prd3→code=2017;
prd3→qte=1200;
```

→ L'affichage :

```
printf(“%d \n”,Prd3→qte);
```

→ La Lecture :

```
scanf(“%f”,&Prd3→prix);
```

1200

7000

Les Structures

- Structures contenant des tableaux:

```
struct personne
{
    char nom[30] ;
    char prenom [20] ;
    double heures [31] ;
} employe;
```

employe

nom



prenom



heures



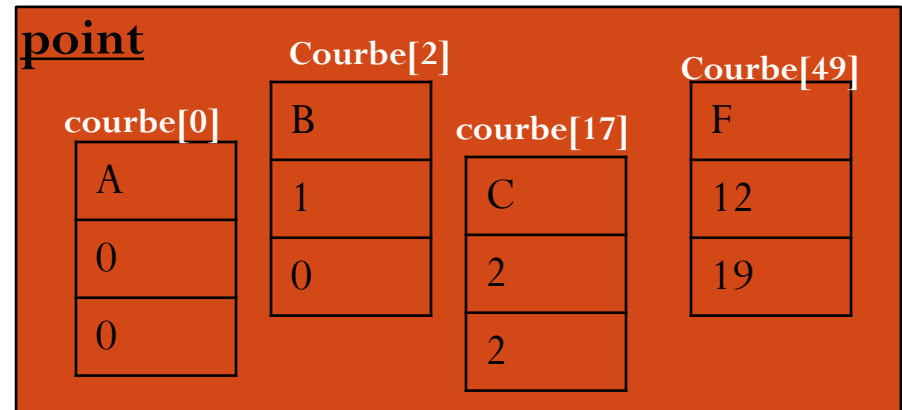
Réserve les emplacements pour une structure nommée **employe** .Ces derniers comportent trois champs:

- **nom** qui est un **tableau** de **30 caractères**,
- **prenom** qui est un **tableau** de **20 caractères**,
- **heures** qui est un **tableau** de **31 flottants**.

Les Structures

- Tableaux de Structures:

```
struct point {  
    char nom ;  
    int x ;  
    int y ;  
};  
struct point courbe[50];
```



➔ La structure point pourrait, par exemple, servir à représenter un point d'un plan, point qui serait défini par son nom (caractère) et ses deux coordonnées.

➔ Le tableau courbe, pourrait servir à représenter un ensemble de 50 points du type ainsi défini.

Les Structures

- Structures imbriquées:

```
struct Date
```

```
{  
    int    jour;  
    int    mois;  
    int    an;  
};
```

```
struct Membre
```

```
{  
    char    nom[80];  
    char    adresse[200];  
    int     numero;  
    float   amende[10];  
    struct  Date  emprunt;  
    struct  Date  creation;  
};
```

```
struct Livre
```

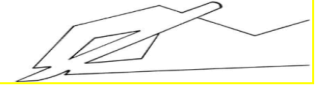
```
{  
    char    titre[80];  
    char    auteur[80];  
    float   prix;  
};
```

```
struct Pret
```

```
{  
    struct  Livre  b;  
    struct  Date   due;  
    struct  Membre *who;  
};
```

Les Structures

EXERCICES

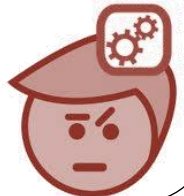


Exercice 1 :

Ecrire un programme qui lit au clavier des informations dans un tableau de structures du type point défini comme suit:

```
typedef struct {  
    char nom;  
    double x ;  
    double y;  
    } point ;
```

→ Le nombre d'éléments du tableau est une constante.



Les Structures

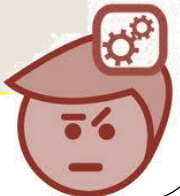
EXERCICES

Exercice 2 : Soit la structure suivante :

```
typedef struct {  
    char Matricule [10];  
    char nom [50];  
    char prenom [50];  
    int Nb_heures_sup;  
    float salaire_fixe ;  
    float salaire ;  
    float prime ;  
} employe;
```

Ecrire un programme permettant de déclarer, de remplir et d'afficher un tableau de N éléments de type employé.

*NB : $\text{salaire} = \text{salaire_fixe} + \text{prime}$ avec $\text{prime} = \text{Nb_heures_sup} * 10$;*



Les Structures

Transmission d'une structure en argument d'une fonction → Transmission par Valeur

```
#include <stdio.h>
struct produit {
    int code ;
    float prix ;
} ;
main()
{
    struct produit prd1 ;
    void fct (struct produit p) ;
    prd1.code = 1055; prd1.prix = 12.5;
    printf ("\navant appel fct : %d %f", prd1.code, prd1.prix);
    fct (prd1) ;
    printf ("\nau retour dans main : %d %f", prd1.code, prd1.prix);
}
```

```
void fct (struct prdoduit prd)
{
    prd.code = 0; prd.prix=1;
    printf ("\ndans fct : %d %f", prd.code, prd.prix);
}
```

Résultat:

avant appel fct : 1055 12,5
dans fct : 0 1
au retour dans main : 1055 12,5

Les Structures

Transmission d'une structure en argument d'une fonction → Transmission par Adresse

```
#include <stdio.h>
struct produit {
    int code ;
    float prix ;
} ;
main()
{
    struct produit prd2 ;
    void fct (struct produit *) ;
    prd2.code = 1055; prd2.prix = 12.5;
    printf ("\navant appel fct : %d %f", prd2.code, prd2.prix);
    fct (&prd2) ;
    printf ("\nau retour dans main : %d %f", prd2.code, prd2.prix);
}
```

```
void fct (struct prdoduit *prd)
{
    prd→code = 0; prd→prix=1;
    printf ("\ndans fct : %d %f", prd→code, prd→prix);
}
```

Résultat:

avant appel fct : 1055 12,5
dans fct : 0 1
au retour dans main : 0 1

Les Structures

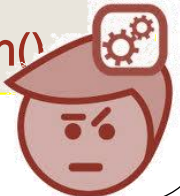
EXERCICES

Exercice 3 : Soit la structure suivante :

```
typedef struct {  
    char *nom;  
    char *prenom ;  
    long CNE ;  
    char filiere [4]; /* GI ou GSTR*/  
    float CC1;  
    float CC2;  
    float Note_projet;  
    float Note_module;  
    char decision[3]; /* V: Valdé , NV: Non Validé ,R: Rattrapage  
    } CYCLE_INGENIEUR ;
```

Ecrire un programme permettant de déclarer, de remplir et d'afficher un tableau de N étudiant .

N.B: Le tableau des étudiants doit être déclaré à l'intérieur de la fonction main()



Les Fichiers

Chapitre3: Les Fichiers

Les Fichiers

- Définition

Un **fichier** est un ensemble **d'informations stocké** sur une mémoire de masse (disque dur, disquette, bande magnétique, CD-ROM).

Types de Fichiers

Fichier Binaire: contient des données non textuelles. Ils ne prennent sens que s'ils sont traités par un programme adapté.

Exemples: code exécutable d'1 prog., fichiers son, vidéo, etc.

Fichier Texte: est formé de caractères ASCII, organisés en lignes, chacune terminée par un caractère de contrôle de fin de lignes.

Les fichiers textes peuvent être édités avec des éditeurs de texte et affichés de manière lisible à l'écran.

Les Fichiers

Types d'accès :

Séquentiel : le fichier est parcouru systématiquement depuis le début jusqu'à l'élément recherché

Direct : la position de l'élément recherché est fournie

Les Fichiers

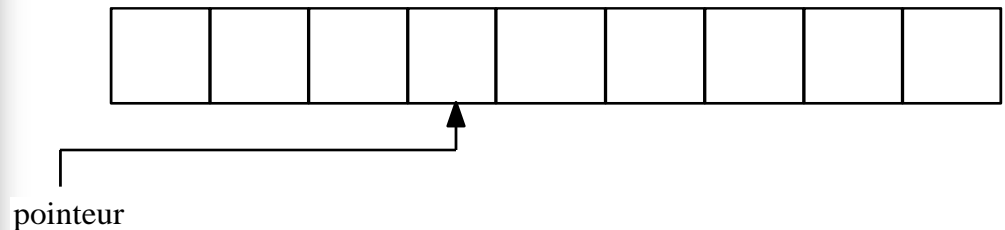
- Opérations et Déclaration

Opérations de Base (Librairie stdlib.h)

- Créer
- Ouvrir
- Fermer
- Lire
- Ecrire
- Détruire
- Renommer

➔ Déclaration: **FILE *fichier;**

➔ On définit un **pointeur** qui fournit l'adresse d'une cellule donnée.



NB: La déclaration des fichiers doit figurer **AVANT** la déclaration des autres variables.

Les Fichiers

- Ouverture et fermeture de fichiers:

➔ L'ouverture se fait à l'aide de la fonction **fopen**:

FILE* f;

f=fopen(const char* name, const char* mode);

➔ La Fermeture se fait à l'aide de la fonction **fclose** :

int fclose (FILE *f);

➔ la fonction **int feof(FILE *fichier)** retourne 0 tant que la fin du fichier n'est pas atteinte.

Exemple:

```
FILE *fichier ;  
fichier = fopen("c : \listes.txt ", "w" ) ;  
/* instructions et traitements*/  
fclose(fichier) ;
```

Les Fichiers

- Modes d'ouverture d'un fichier Texte:

```
FILE *fopen(char *nom, char *mode);
```

mode :

- « **r** » ouverture d'un fichier en **lecture** : le fichier doit exister, autrement la fonction fopen return NULL ;
- « **w** » création et ouverture d'un fichier en **écriture** : si le fichier existe, son contenu est détruit ;
- « **a** » ouverture d'un fichier en **écriture à la fin du fichier** : si le fichier n'existe pas, il est créé ;
- « **r+** » ouverture d'un fichier **en lecture et écriture** : le fichier doit exister, autrement la fonction fopen return NULL ;
- « **w+** » **création et ouverture** d'un fichier **en lecture et écriture** : si le fichier existe, son contenu est détruit ;
- « **a+** » **ouverture d'un fichier en lecture et en écriture à la fin** du fichier : si le fichier n'existe pas, il est créé.

Les Fichiers



Ces modes d'accès ont pour particularités :

- Si le mode contient la **lettre r**, le fichier **doit exister**.
- Si le mode contient la **lettre w**, le fichier **peut ne pas exister**. Dans ce cas, il sera créé. Si le fichier existe déjà, son **ancien contenu sera perdu**.
- Si le mode contient la **lettre a**, le fichier **peut ne pas exister**. Dans ce cas, il sera créé. Si le fichier existe déjà, **les nouvelles données seront ajoutées à la fin du fichier précédent**.

Les Fichiers

- Lecture et Ecriture dans les fichiers:

Fonctions de lecture

```
int    fscanf(FILE* stream, const char* format, ...);  
int    fgetc(FILE* stream);  
char*  fgets(char* buffer, int size, FILE* stream);
```

Fonctions d'écriture

```
int    fprintf(FILE* stream, const char* format, ...);  
int    fputc(int ch, FILE* stream);  
int    fputs(const char* buffer, FILE* stream);
```



Les Fichiers

• Exemple 1: Lecture / Ecriture :

```
#include <stdio.h>
void main(void)
{
    char titre[81];
    float x[10];
    int ind[10], i=0,n=10;
    FILE *f;
    f = fopen("monfichier.txt","w");
    if (f !=NULL) {
        fprintf(f,"%s\n",titre);
        for (i=0; i < n; i++ ) {
            fprintf(f,"%f %d\n", x[i],ind[i]);
        }
    }
    fclose(f);
}
```

Ecriture

```
#include <stdio.h>
void main(void)
{
    char titre[81];
    float x[10];
    int ind[10], i=0;
    FILE *f;
    f = fopen("monfichier.txt","r");
    if (f!= NULL) {
        fgets(titre,80,f);
        while(!feof(f)) {
            fscanf(f,"%f %d",&x[i],&ind[i]);
            i++;
        }
    }
    fclose(f);
}
```

Lecture

Les Fichiers

Exemple 2: lecture caractère par caractère

```
FILE *f;  
    f = fopen("monfichier.txt","r");  
do  
    {  
        caractereActuel = fgetc(f); // On lit le caractère  
        printf("%c", caractereActuel); // On l'affiche  
    } while (caractereActuel != EOF);  
fclose(f);
```

Les Fichiers

Exemple 3: lecture d'une chaîne de caractères

```
FILE *f;
    f = fopen("monfichier.txt","r");
do
    {
        fgets(chaine, TAILLE_MAX, f);
        // On lit maximum TAILLE_MAX caractères du fichier, on stocke le tout dans
        "chaine"
        printf("%s", chaine); // On affiche la chaîne
    } while (!feof(f));
fclose(f);
```

Les Fichiers

Supprimer un fichier :

Pour supprimer un fichier, on utilise la fonction suivante :

```
int remove(const char* fichierASupprimer);
```

Exemple:

```
int main()  
{  
    remove("test.txt");  
    return 0;  
}
```


Les Fichiers

Renommer un fichier :

Pour renommer un fichier, on utilise la fonction suivante :

```
int rename(char* ancienNom, char* nouveauNom);
```

Exemple:

```
int main()  
{  
    rename("test.txt", "test_renomme.txt");  
    return 0;  
}
```

Les Fichiers

Exercice 1

Ecrire un programme qui permet de:

- Créer un fichier texte dont le nom est choisi par l'utilisateur
- remplir le fichier par une liste des étudiants(CNE, Nom, Prénom) : le nombre des enregistrements est déterminé par l'utilisateur(*Enregistrement par ligne*)
- Afficher son contenu

Exercice 2

1) Soit le fichier texte suivant, écrire un programme en c affichant le nombre de mots commençant par une majuscule.

Semestre 2	Module	Programmation Avancée
en C		
	Contrôle continu	Numéro: 1
fichiers et Listes		
simplement chaînées		

2) Comment récupérer directement le dernier mot « chaînées » de ce fichier .

Les Fichiers

Exercice 3

Ecrire un programme qui permet de supprimer la 5ème ligne et la 8ème ligne d'un fichier texte. Chaque ligne comporte un enregistrement de type étudiant.

Les Fichiers Binaires

- Modes d'ouverture d'un fichier binaire:

```
FILE *fopen(char *nom, char *mode);
```

mode :

- « **rb** » ouverture d'un fichier en **lecture** : le fichier doit exister, autrement la fonction fopen return NULL ;
- « **wb** » création et ouverture d'un fichier en **écriture** : si le fichier existe, son contenu est détruit ;
- « **ab** » ouverture d'un fichier en **écriture à la fin du fichier** : si le fichier n'existe pas, il est créé ;
- « **rb+** » ouverture d'un fichier **en lecture et écriture** : le fichier doit exister, autrement la fonction fopen return NULL ;
- « **wb+** » **création et ouverture** d'un fichier **en lecture et écriture** : si le fichier existe, son contenu est détruit ;
- « **ab+** » **ouverture d'un fichier en lecture et en écriture à la fin** du fichier : si le fichier n'existe pas, il est créé.

Les Fichiers Binaires

Fonction de lecture des fichiers binaires

fread(void **pointeur*, size_t *taille*, size_t *nombre*, FILE **flot*);

- *pointeur*: est l'adresse du début des données à transférer,
- *taille*: la taille des objets à transférer,
- *nombre*: leur nombre.
- **Valeur de retour**: la fonction Renvoie le nombre de blocs lus

Fonction d'écriture dans un fichier binaire

fwrite (void **pointeur*, size_t *taille*, size_t *nombre*, FILE **flot*);

- *pointeur*: est l'adresse du début des données à transférer,
- *taille*: la taille des objets à transférer,
- *nombre*: leur nombre.
- **Valeur de retour**: la fonction Renvoie le nombre de blocs écrits

Les Fichiers Binaires

Exemples:

Ecrire dans un fichier binaire :

```
int main(void) {  
    FILE *f_in, *f_out;  
    char F_SORTIE[]="c:\nomfich.bin";  
    int tab1[50], tab2[50]; int i;  
    for (i = 0 ; i < NB; i++) tab1[i] = i;  
    if ((f_out = fopen(F_SORTIE, "wb"))  
        == NULL) { printf("Impossible  
d'écrire dans le fichier");  
        return(-1);}  
    fwrite(tab1, 50 * sizeof(int), 1,  
        f_out);  
    fclose(f_out);  
}
```

Lire à partir d'un fichier binaire :

```
if ((f_in = fopen(F_SORTIE, "rb"))  
    == NULL) { printf("Impossible de  
lire dans le fichier "); return(-1); }  
fread(tab2, 50 * sizeof(int), 1,  
    f_in);  
fclose(f_in);  
for (i = 0 ; i < 50; i++)  
    printf("%d\t", tab2[i]);  
return(0);  
}
```

Les Fichiers Binaires

Positionnement dans un fichier:

Il est possible d'accéder à un fichier en *mode direct*, c'est-à-dire que l'on peut se positionner à n'importe quel endroit du fichier. La fonction **fseek** permet de se positionner à un endroit précis:

```
int fseek(FILE *flot, long deplacement, int origine);
```

La variable *origine* peut prendre trois valeurs :

0 : début du fichier ;

1 : position courante ;

2 : fin du fichier.

Les Fichiers Binaires

Positionnement dans un fichier:

La fonction

int rewind(FILE *fptr);

permet de se positionner au début du fichier.

Elle est équivalente à : ***fseek(fptr, 0, 0);***

La fonction

long ftell(FILE *fptr);

retourne la position courante dans le fichier (*en nombre d'octets depuis l'origine*).

Les Fichiers Binaires

Exemple:

```
/* on se positionne a la fin du fichier */
fseek(f_in, 0, 2); printf("\n position %ld", ftell(f_in));
/* déplacement de 10 int en arriere */
fseek(f_in, -10 * sizeof(int), 2);
printf("\n position %ld", ftell(f_in));
fread(&i, sizeof(int), 1, f_in); printf("\t i = %d", i);
/* retour au début du fichier */
rewind(f_in); printf("\n position %ld", ftell(f_in));
fread(&i, sizeof(int), 1, f_in); printf("\t i = %d", i);
/* déplacement de 5 int en avant */
fseek(f_in, 5 * sizeof(int), 1);
printf("\n position %ld", ftell(f_in));
fread(&i, sizeof(i), 1, f_in); printf("\t i = %d\n", i);
```

Les Fichiers Binaires

Exercice 1:

Développer un programme en C permettant de faire la gestion des *courriers électroniques*, chaque courrier est identifié par: l'adresse de son expéditeur, son sujet, sa date d'envoi, son contenu et son état de lecture (Par défaut l'état des messages non lus prend la valeur 0).

- 1) Donnez la déclaration de(s) (la) structure(s) nécessaire(s) pour gérer ces données.
- 2) Ecrivez une fonction permettant d'ajouter un courrier électronique au fichier binaire ***Mail.bin*** sans écraser son contenu .
- 3) Ecrivez une fonction permettant de numéroter les enregistrements du fichier **Mail.bin**.

Exercice 1(suite):

- 4) Ecrivez une fonction permettant de copier les enregistrements impaires du fichier **Mail.bin** dans le fichier **Mail_Impairs.bin** et ceux paires dans le fichier **Mail_Pairs.bin**.
- 5) Ecrivez une fonction *Chercher_Mail_Exped(char *nom_fich, char *adr_exp)* permettant de rechercher un mail dans le fichier **Mail.bin** à base de l'adresse de son expéditeur. La fonction retournera le numéro de de l'enregistrement de la première occurrence .
- 7) Ecrivez une fonction *MenuPrincipale()* permettant d'afficher à l'utilisateur la liste des actions à faire.
- 8) Ecrivez une fonction *main()* permettant de faire appel aux fonctions développées.

- Pb et difficultés

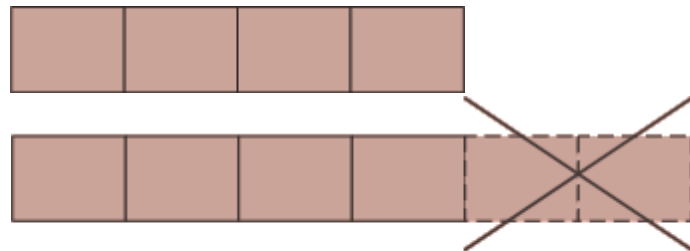
➔ Stocker des données en mémoire, nous avons utilisé des variables simples :

- Type int, double. . . ,
- Des tableaux,
- Des structures personnalisées. Si vous souhaitez

➔ Stocker une série de données, le plus simple est en général d'utiliser des tableaux.

➔ Limitation: (Exemple)

Int tab[4]

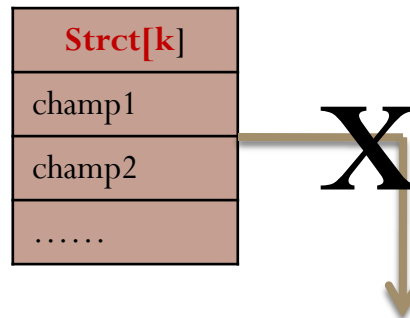


Impossible d'ajouter des cases à un tableau après sa création !

- Pb et difficultés

- ➔ Lors de la manipulation de nombre variable d'instances d'une structure, et on souhaite insérer, supprimer dynamiquement;
- ➔ *Les Tableaux de structures ne suffisent plus*

```
Struct strct{
    champ1;
    Champ2;
};
struct strct strct[n];
```

[illegible]