Chapitre 7.2: Les pointeurs.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.1. Introduction

- Toute variable manipulée dans un programme est stockée quelque part en mémoire centrale. Cette mémoire est constituée d'octets qui sont identifiés de manière univoque par un numéro qu'on appelle ***adresse.***

- Pour retrouver une variable, il suffit donc de connaître l'adresse de l'octet où elle est stockée.

- Pour des raisons évidentes de lisibilité, on désigne souvent les variables par des identificateurs, et non par leur adresse.

- C'est le compilateur qui fait alors le lien entre l'identificateur d'une variable et son adresse en mémoire. Toutefois, il est parfois très pratique de manipuler directement une variable par son adresse.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.2. Adresse et valeur d'un objet

- On appelle *Lvalue* (left value) tout objet pouvant être placé à gauche d'un opérateur d'affectation. Une *Lvalue* est caractérisée par :
  - son adresse: l'adresse-mémoire à partir de laquelle l'objet est stocké;
  - sa valeur, c'est-à-dire ce qui est stocké à cette adresse.
- Dans l'exemple, int i, j; i = 3; j = i; Si le compilateur a placé la variable i à l'adresse 4831836000 en mémoire, et la variable j à l'adresse 4831836004, on a: Deux variables différentes ont des adresses différentes.

objet	adresse	valeur
i	4831836000	3
j	4831836004	3

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.2. Adresse et valeur d'un objet

- L'affectation i = j; n'opère que sur les valeurs des variables. Les variables i et j étant de type int, elles sont stockées sur 4 octets. Ainsi la valeur de i est stockée sur les octets d'adresse 4831836000 à 4831836003.
- L'opérateur & permet d'accéder à l'adresse d'une variable.
- Toutefois &i n'est pas une Lvalue mais une constante → on ne peut pas faire figurer &i à gauche d'un opérateur d'affectation.
- Pour pouvoir manipuler des adresses, on doit donc recourir un nouveau type d'objets, les pointeurs.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.3. Notion de pointeur

- Un pointeur est un objet dont la valeur est égale à l'adresse d'un autre objet. On déclare un pointeur par l'instruction :

  type **nom-du-pointeur*;

où ***type*** est le type de l'objet pointé.

- Cette déclaration déclare un identificateur, *nom-du-pointeur*, associé à un objet dont la valeur est l'adresse d'un autre objet de type ***type*.**
- L'identificateur *nom-du-pointeur* est donc en quelque sorte un identificateur d'adresse. Comme pour n'importe quelle Lvalue, sa valeur est modifiable.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.3. Notion de pointeur

- Dans l'exemple suivant, on définit un pointeur p qui pointe vers un entier i :

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

- On se trouve dans la configuration

objet	adresse	valeur
i	4831836000	3
p	4831836004	4831836000

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.3. Notion de pointeur

- **L'opérateur unaire d'indirection** * permet d'accéder directement à la valeur de l'objet pointé. Ainsi, si p est un pointeur vers un entier i, *p désigne la valeur de i. Par exemple, le programme

main() { int i = 3; int *p; p = &i; printf("*p = %d \n",*p);} ➔*p = 3. *p=i

- Les objets i et *p sont identiques : ils ont mêmes adresse et valeur:

objet	adresse	valeur
i	4831836000	3
p	4831836004	4831836000
*p	4831836000	3

Cela signifie en particulier que toute modification de *p modifie i. Ainsi, si l'on ajoute l'instruction *p = 0; à la fin du programme précédent, la valeur de i devient nulle.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.3. Notion de pointeur

On peut donc dans un programme manipuler à la fois les objets p et *p. Ces deux manipulations sont très différentes. Comparons par exemple les deux programmes suivants : P_1 ➔

main() { int i = 3, j = 6; int *p1, *p2; p1 = &i; p2 = &j; *p1 = *p2; }

Avant la dernière affectation

objet	adresse	valeur
i	4831836000	3
j	4831836004	6
p1	4831835984	4831836000
p2	4831835992	4831836004

Après *p1 = *p2

objet	adresse	valeur
i	4831836000	6
j	4831836004	6
p1	4831835984	4831836000
p2	4831835992	4831836004

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.3. Notion de pointeur

P_2➔ main() { int i = 3, j = 6; int *p1, *p2; p1 = &i; p2 = &j; p1 = p2; }
Par contre, l'affectation p1 = p2 du second programme, conduit à la situation :

objet	adresse	valeur
i	4831836000	3
j	4831836004	6
p1	4831835984	4831836004
p2	4831835992	4831836004

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.3. Arithmétique des pointeurs

- La valeur d'un pointeur étant un entier, on peut lui appliquer un certain nombre d'opérateurs arithmétiques classiques.

- Les seules opérations arithmétiques valides sur les pointeurs sont :

- L'addition d'un entier à un pointeur. Le résultat est un pointeur de même type que le pointeur de départ ;
- La soustraction d'un entier à un pointeur. Le résultat est un pointeur de même type que le pointeur de départ; La différence de deux pointeurs pointant tous deux vers des objets de même type. Le résultat est un entier.

➔ Notons que la somme de deux pointeurs n'est pas autorisée.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.3. Arithmétique des pointeurs

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

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

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.3. Arithmétique des pointeurs

Exemple:

```
main(){ int i=3, *p1, *p2; p1 = &i;
      p2= p1+ 1;
      printf("p1 = %ld \t p2 = %ld\n",p1,p2); }
```

➔ affiche p1 = 4831835984   p2 = 4831835988.

Par contre, le même programme avec des pointeurs sur des objets de type double :

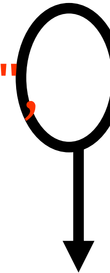
➔ affiche **p1 = 4831835984   p2 = 4831835992.**

7. Tableaux, Chaînes de caractères et pointeurs.

Déclaration de pointeurs

Exemple 2

```
#include <stdio.h>
int main(void)
{
int x=2;
int *p = &x;   /* x et *p deviennent synonymes */
*p = 3;
printf("La nouvelle valeur de x est %d \n",   x  );
return 0;
}
```

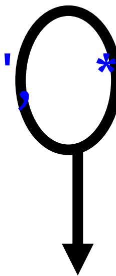


/* doit afficher la valeur 3 */

7. Tableaux, Chaînes de caractères et pointeurs.

Déclaration de pointeurs

```
#include <stdio.h>
int main(void)
{
int x=2;                                    /* affiche 2 */
int *p = &x;  /* x et *p deviennent synonymes */
printf("La valeur de x est %d\n", *p);
x=5;
printf("La nouvelle valeur de x est %d\n",  *p  );
/* doit afficher la valeur 5 */
return 0;
}
```



5

Opérateurs unaires pour manipuler les pointeurs, & (adresse de) et * (contenu)

```
Exemple:  int i = 8;
          printf("VOICI i: %d\n",i);
          printf("VOICI SON ADRESSE EN HEXADECIMAL: %p\n", &i);
```

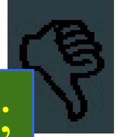
nom_de_Pointeur = &nom_de_variable

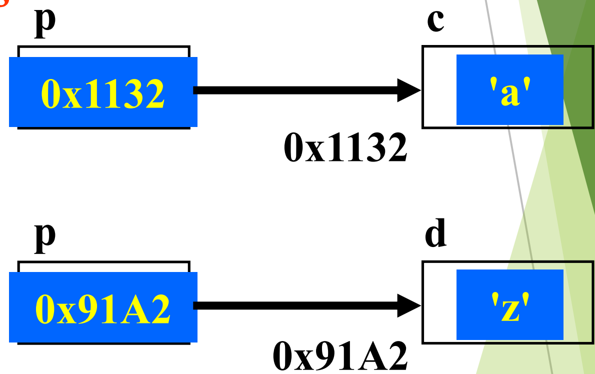
```
void    main(void)
{
        char    c = 'a', d = 'z';
        char    *p;

        p = &c;
    printf("%c\n", *p);
        p = &d;
    printf("%c\n", *p);
}
```

reçoit l'adresse de c; donc pointe sur c.

*p = c;





a
z

L'opérateur * ("valeur pointée par")

* et ++

<u>*p++ signifie:</u>

*p++ trouver la **valeur** pointée

*p++ passer à **l'adresse** suivante

<u>(*p)++ signifie:</u>

(*p)++ trouver la **valeur** pointée

(*p)++ incrémenter cette **valeur** (sans changer le pointeur)

<u>*++p signifie:</u>

*++p incrémenter d'abord le **pointeur**

*++p trouver la **valeur** pointée

```
int a[]={0,1,5};
int*p;
p=a;//&a[0]==a
Int b=*p++;//
```

p	100	104	108
a	0	1	5

```
(*p)++;  //*p=2
 Int c= (*p)++;  //c=2
```

p	100	104	108
a	0	3	5

```
Int d= *++p;  / /d==5
```

p	100	104	108
a	0	3	5

Exercice

```
#include <stdio.h>
void main() {
  int *p, x, y;
    p = &x;                              /* p pointe sur x */
    x = 10;                              /* x vaut 10 */
    y = *p - 1;   printf(" y= *p - 1 =?   = %d\n" , y);
    *p += 1;     printf(" *p += 1 =? *p = x= ?  = %d %d\n" , *p, x);
    (*p)++;     printf(" (*p)++ =? *p = x=  ?  = %d %d  alors y=%d \n" , *p, x, y);

    *p=0;        printf("  *p=0  x=? =  %d\n" , x);

    *p++; *p=20;  printf("  *p++  x=? =  %d\n" , x);
}
```

y vaut 9

x vaut 11

incrémente aussi de 1 la variable pointée par p, donc x vaut 12.
y vaut 9

comme p pointe sur x, maintenant x vaut 0

comme p ne pointe plus sur x, x vaut tjr 9

Utiliser des pointeurs

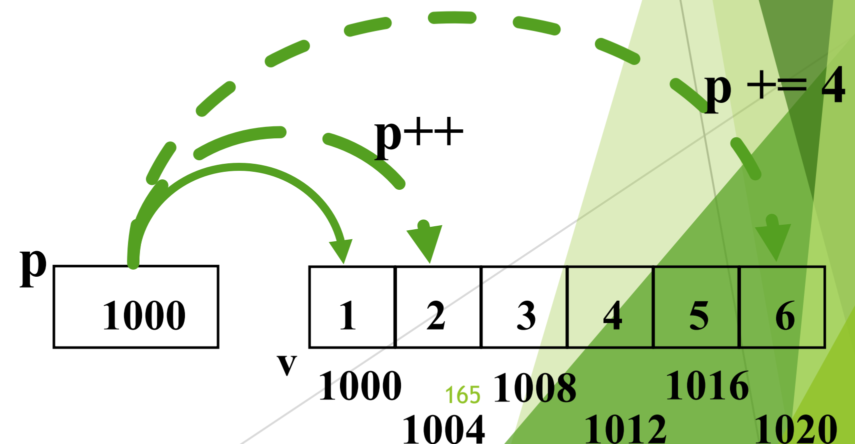
- On peut donc accéder aux éléments par pointeurs en faisant des calculs d’adresses (addition ou soustraction)

```
long v[6] ={1,2, 3,4,5,6 };
long

*p;

p = v;     // v==&v[0]
printf("%ld\n", *p);
p++;
printf("%ld\n", *p);
p += 4;
printf("%ld\n", *p);
```

1
2
6



Exercice

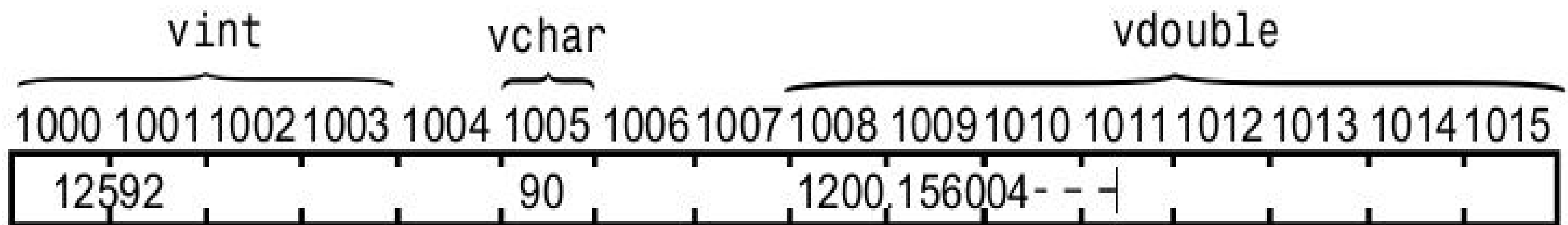
- int t[] = {2,5,7};
- int *p = t;
- int a, b, c ;   //*p=
- **b** = *p++;   // b =
- **b** = *p;   // b =
- **a** = (*p)++; // a=
- **c = *++p**   // c =

- *p=t[0]===2
- b=*p; puis p++; b=2, p=&t[1]
- b=5
- a=5; *p=t[1]=t[1]+1=6
- ++p puis c=*p ➔ c= 7

Pointeurs et types de variables

Les différents types de variables du langage C n'occupent pas tous la même mémoire (int prend quatre octets, une variable double en prend huit, etc)

```
int  vint = 12252;
char  vchar = 90  //asci de Z
double vdouble = 1200.156004
```



La variable **int occupe** **quatre** **octets**,
la variable **char en occupe** **un**,
et **la variable double** **huit**.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.3. Arithmétique des pointeurs

- Les opérateurs de comparaison sont également applicables aux pointeurs, à condition de comparer des pointeurs qui pointent vers des objets de même type.

Exemple:

```
#define N 5
int tab[5] = {1, 2, 6, 0, 7};
 main() { int *p;
   printf("\n ordre croissant:\n");
for (p = &tab[0]; p <= &tab[N-1]; p++) printf(" %d \n",*p);
   printf("\n ordre decroissant:\n");
for (p = &tab[N-1]; p >= &tab[0]; p--) printf(" %d \n",*p);
 }
```

- Si p et q sont deux pointeurs sur des objets de type *type*, l'expression p - q désigne un entier dont la valeur est égale à (p - q)/sizeof(*type*) .

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.4. Allocation dynamique

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: p=&i;
- Soit par l'allocation dynamique d'un nouvel espace-mémoire.

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

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.4. Allocation dynamique

L'allocation dynamique en C se fait par l'intermédiaire de la fonction dans la libraire standard stdlib.h:

```
char * malloc(nombreOctets)
```

- Par défaut, cette fonction retourne un char * pointant vers une espace mémoire de taille **nombreOctets** octets.
- Pour initialiser des pointeurs vers des objets qui ne sont pas de type char, il faut convertir le type de la sortie de la fonction malloc à l'aide d'un cast.
- L'argument **nombreOctets** est souvent donné à l'aide de la fonction sizeof() qui renvoie le nombre d'octets utilisés pour stocker un objet.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.4. Allocation dynamique

Pour initialiser un pointeur vers un entier, on écrit :

```
#include <stdlib.h>
      int *p;
      p = (int*)malloc(sizeof(int));
```

On aurait pu écrire également

```
      p = (int*)malloc(4);
```

Puisqu'un objet de type int est stocké sur 4 octets. Mais on préfèrera la première écriture qui a l'avantage d'être portable.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.4. Allocation dynamique

- Le programme suivant:

```
#include <stdio.h>
#include <stdlib.h>
main(){int i = 3; int *p;
 printf("valeur de p avant initialisation = %ld\n",p);
p = (int*)malloc(sizeof(int));
printf("valeur de p apres initialisation = %ld\n",p);
*p = i; printf("valeur de *p = %d\n",*p);}
```

- Ce programme définit un pointeur p sur un objet *p de type int, et affecte à *p la valeur de la variable i. Il imprime à l'écran (avertissement):
  - valeur de p avant initialisation = 0
  - valeur de p après initialisation = 5368711424
  - valeur de *p = 3

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.4. Allocation dynamique

La fonction malloc permet également d'allouer un espace pour plusieurs objets adjacents en mémoire. On peut écrire par exemple

```
#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));}
```

On a ainsi réservé, à l'adresse donnée par la valeur de p, 8 octets en mémoire, qui permettent de stocker 2 objets de type int. Le programme affiche : p = 5368711424 *p = 3 p+1 = 5368711428 *(p+1) = 6 .

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.4. Allocation dynamique

La fonction **`calloc`** de la librairie **`stdlib.h`** a le même rôle que la fonction **`malloc`** mais elle initialise en plus l'objet pointé *p à zéro. Sa syntaxe est

```
calloc(nb-objets,taille-objets)
```

Ainsi, si p est de type int*, l'instruction

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

est strictement équivalente à

```
p = (int*)malloc(N * sizeof(int));
for (i = 0; i < N; i++)
        *(p + i) = 0;//p[i]=0;
```

➔ L'emploi de **`calloc`** est simplement plus rapide.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.4. Allocation dynamique

- Lorsque l'on n'a plus besoin de l'espace-mémoire alloué dynamiquement (c'est-à-dire quand on n'utilise plus le pointeur p), il faut libérer cette place en mémoire.

- Ceci se fait à l'aide de l'instruction free qui a pour syntaxe:

free(*nom-du-pointeur*);

- A toute instruction de type malloc ou calloc doit être associée une instruction de type free.
- **adr2 = realloc(adr1, taille);**

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.4. Allocation dynamique

void * realloc (void * pointeur, size_t taille)
permet de modifier la taille d'une zone préalablement allouée (par malloc, calloc ou realloc).

Le pointeur doit être l'adresse de début de la zone dont on veut modifier la taille. Quant à taille, de type size_t, elle représente la nouvelle taille souhaitée.

```
adr2 = (int*)realloc(adr1, taille);
if(adr2 != NULL) adr1 = NULL;
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.4. Allocation dynamique : Exemple

```
#include <stdio.h>
 #include <stdlib.h>
 int main () { char *str; /* Initial memory allocation */
str = (char *) malloc(15);
strcpy(str, "tutorialspoint");
printf("String = %s, Address = %u\n", str, str); /* Reallocating memory */
str = (char *) realloc(str,  25);
strcat(str, ".com");
printf("String = %s, Address = %u\n", str, str);
free(str);   return 0 ;    }
```

```
String = tutorialspoint, Address = 355090448
String = tutorialspoint.com, Address =
355090448
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.5. Pointeurs et tableaux

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

A. Pointeurs et tableaux à une dimension

Tout tableau en C est en fait un pointeur constant. Dans la déclaration:

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

- tab est un pointeur constant (non modifiable) dont la valeur est l'adresse du premier élément du tableau.
- Autrement dit, tab a pour valeur &tab[0]. On peut donc utiliser un pointeur initialisé à tab pour parcourir les éléments du tableau.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.5. Pointeurs et tableaux

A. Pointeurs et tableaux à une dimension

- En déclarant un tableau A de type int (int A[N]) et un pointeur P sur des variables entière (int *p),
- l'instruction p = A crée une liaison entre le pointeur P et le tableau A en mettent dans P l'adresse du premier élément de A (de même p = &A[0]). A partir du moment ou p = A;
- la manipulation du tableau A peut se faire par le biais du pointeur P. En effet :

p	**désinge**	**&A[0]**	***p**	**désigne**	**A[0]**
p+1	**désinge**	**& A[1]**	***(p+1)**	**désigne**	**A[1] .**
p+i	**désinge**	**&A[i]**	***(p+i)**	**désigne**	**A[i]**

Ou i ϵ [0 , N-1]

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.5. Pointeurs et tableaux

A. Pointeurs et tableaux à une dimension

```
#define N 5
int tab[5] = {1, 2, 6, 0, 7};
main()
   {int i;
   int *p;
   p = tab; /*p=&tab[0],   p[i] ==tab[i]*/
   for (p = tab ; p<tab+N; p++)/* for (i = 0; i < N; i++)*/

      {     printf(" %d \n",*p); /* p++;/*

      }
   }
```

pour accéder un élément i du Tab : tab[i] ➔

p[i] = *(p + i)

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.5. Pointeurs et tableaux

A. Pointeurs et tableaux à une dimension

Pointeurs et tableaux se manipulent donc exactement de même manière.

Par exemple, le programme précédent peut aussi s'écrire

```
#define N 5
int tab[5] = {1, 2, 6, 0, 7};
 void main() {
 int i;
  for (i = 0; i < N; i++)
      printf(" %d \n", tab[i]); /* p[i] ==tab[i]*/
}
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.5. Pointeurs et tableaux

A. Pointeurs et tableaux à une dimension

la manipulation de tableaux, et non de pointeurs, possède certains inconvénients dus au fait qu'un tableau est un pointeur constant. Ainsi

- ➢ On ne peut pas créer de tableaux dont la taille est une variable du programme,
- ➢ On ne peut pas créer de tableaux bidimensionnels dont les lignes n'ont pas toutes le même nombre d'éléments.

➔ Ces opérations deviennent possibles dès que l'on manipule des pointeurs alloués dynamiquement.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.5. Pointeurs et tableaux

A. Pointeurs et tableaux à une dimension

- Pour créer un tableau d'entiers à n éléments où n est une variable du programme, on écrit

```
#include <stdlib.h>
void main() { int n; int *tab; ...
tab = (int*)malloc(n * sizeof(int)); ... free(tab); }
```

- Si on veut en plus que tous les éléments du tableau tab soient initialisés à zéro, on remplace l'allocation dynamique avec malloc par

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

- Les éléments de tab sont manipulés avec l'opérateur d'indexation [], exactement comme pour les tableaux.

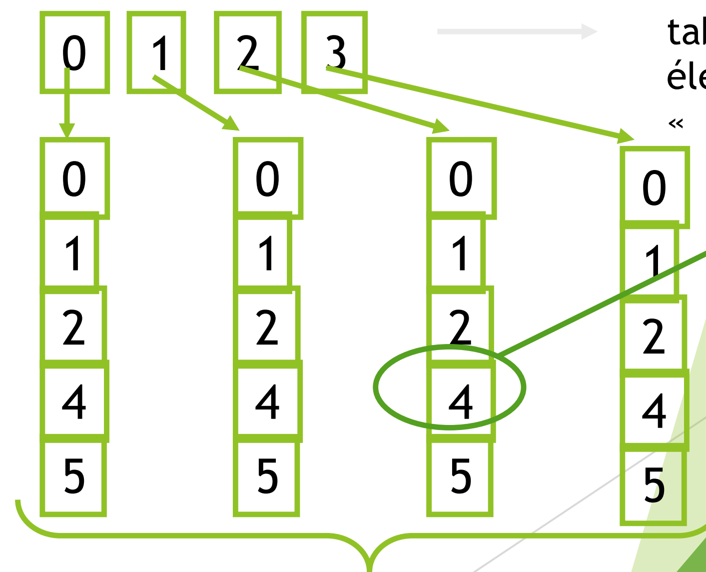
Un nom de tableau est un pointeur constant

Un tableau à plusieurs dimensions est un pointeur de pointeur.

int t[4] [5] ; t désigne un tableau de 4 éléments, chacun de ces éléments étant lui-même un tableau de 5 entiers

les notations **t** et **&t[0][0]** correspondent toujours à **la même adresse**

t[0] représente l'adresse de début du premier bloc (de 5 entiers) de t, t[1], celle du second bloc...



« Pointeurs » vers des v
tableau 1D dont chaque
élément
« pointe » vers un vect

t[2][4],

t[2] est l'*adresse* du début du vecteu d'indice 2, c'est un pointeur vers t[2][

Espace de stockage du tableau

les notations suivantes sont totalement équivalentes :

t[0]	**équiva**	**&t[0][0]**
t[1]	**équiva**	**&t[1][0]**
t[i]	**équiva**	**&t[i][0]**

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.5. Pointeurs et tableaux

tab ou tab[0] ou &tab[0][0]

Tab+1 , tab[1] ou &tab[1][0]

Tab+2 , tab[2] ou &tab[2][0]

1	2	3	4
5	6	7	8
9	10	11	12

B. Pointeurs et tableaux à plusieurs dimensions

- ❑ Un tableau à deux dimensions est, par définition, un tableau de tableaux. Il s'agit donc en fait d'un pointeur vers un pointeur.

Considérons le tableau à deux dimensions défini par : int tab[M][N];

- ➢ tab est un pointeur, qui pointe vers un objet lui-même de type pointeur d'entier. tab a une valeur constante égale à l'adresse du premier élément du tableau, &tab[0][0].

- ➢ De même tab[i], pour i entre 0 et M-1, est un pointeur constant vers un objet de type entier, qui est le premier élément de la ligne d'indice i. tab[i] a donc une valeur constante qui est égale à &tab[i][0].

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.5. Pointeurs et tableaux

B. Pointeurs et tableaux à plusieurs dimensions

- Exactement comme pour les tableaux à une dimension, les pointeurs de pointeurs ont de nombreux avantages sur les tableaux multi-dimensionnés.

- On déclare un pointeur qui pointe sur un objet de type *type* * (deux dimensions) de la même manière qu'un pointeur, c'est-à-dire

  type ***nom-du-pointeur*;

- De même un pointeur qui pointe sur un objet de type *type* ** (équivalent à un tableau à 3 dimensions) se déclare par

  type ****nom-du-pointeur*;

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.5. Pointeurs et tableaux

B. Pointeurs et tableaux à plusieurs dimensions

- Soit la déclaration d'un tableau A et d'un pointeur P : **`int A[M][N], *p;`**
- l'instruction **`p = A[0]`** crée une liaison entre le pointeur p et la matrice A en mettent dans p l'adresse du premier élément de la première ligne de la matrice A (**`p = &A[0][0])`**.
- A partir du moment ou **`p = A[0],`** la manipulation de la matrice A peut se faire par le biais du pointeur P. En effet :

p désigne &A[0][0] et * p désigne A[0][0]
p + 1 désigne &A[0][1] et * (p + 1) désigne A[0][1]
p + N désigne &A[1][0] et * (p + N) désigne A[1][0]
p + N + 1 désigne &A[1][1] et * (p + N + 1) désigne A[1][1]
p + i * N + j désigne &A[i][j] et * (p + i * N + j) désigne A[i][j]

Ou i є [0 , M-1] et j є [0 , N-1]

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.5. Pointeurs et tableaux

B. Pointeurs et tableaux à plusieurs dimensions

Par exemple, pour créer avec un pointeur de pointeur une matrice à k lignes et n colonnes à coefficients entiers, on écrit :

```
main() {
   int k, n; int **tab;
   tab = (int**)malloc(k * sizeof(int*));
   for (i = 0; i < k; i++)
      tab[i] = (int*)malloc(n * sizeof(int));
    ....

   for (i = 0; i < k; i++)
      free(tab[i]);
   free(tab);
}
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.5. Pointeurs et tableaux

B. Pointeurs et tableaux à plusieurs dimensions

- La première allocation dynamique réserve pour l'objet pointé par tab l'espace-mémoire correspondant à k pointeurs sur des entiers. Ces k pointeurs correspondent aux lignes de la matrice.

- Les allocations dynamiques suivantes réservent pour chaque pointeur tab[i] l'espace-mémoire nécessaire pour stocker n entiers.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.5. Pointeurs et tableaux

B. Pointeurs et tableaux à plusieurs dimensions

- Si on désire en plus que tous les éléments du tableau soient initialisés à zéro, il suffit de remplacer l'allocation dynamique dans la boucle for par `tab[i] = (int*)calloc(n, sizeof(int));`

- Contrairement aux tableaux à deux dimensions, on peut choisir des tailles différentes pour chacune des lignes tab[i].

Par exemple, si l'on veut que tab[i]contienne exactement i+1 éléments, on écrit

```
for (i = 0; i < k; i++)
   tab[i] = (int*)malloc((i + 1) * sizeof(int));
```

Chapitre 7.3: Pointeurs et chaînes de caractères .

caractères .

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.6. Pointeurs et chaînes de caractères

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

- On peut donc manipuler toute chaîne de caractères à l’aide d’un pointeur sur un objet de type char. On peut faire subir à une chaîne définie par:

  char *chaine;

- L’affectations : chaine = "ceci est une chaine";

et toute opération valide sur les pointeurs, comme l’instruction chaine++;. Ainsi, le nombre de caractères d’une chaîne sera déterminé(sans compter le caractère nul) par le programme suivant:

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.6. Pointeurs et chaînes de caractères

```
int main() {
   int i; char *chaine;  /* char t[5]; t= "abcd" ; Erreur*/
   chaine = "chaine de caracteres";
   for (i = 0; *chaine != '\0'; i++)
       chaine++;
   printf("nombre de caracteres = %d\n",i);
   return 0;
}
```

- La fonction donnant la longueur d'une chaîne de caractères, définie dans la librairie standard string.h, procédé de manière identique.
- Il s'agit de la fonction strlen dont la syntaxe est strlen(*chaine*); où *chaine* est un pointeur sur un objet de type char.
- Cette fonction renvoie un entier dont la valeur est égale à la longueur de la chaîne passée en argument (moins le caractère '\0').

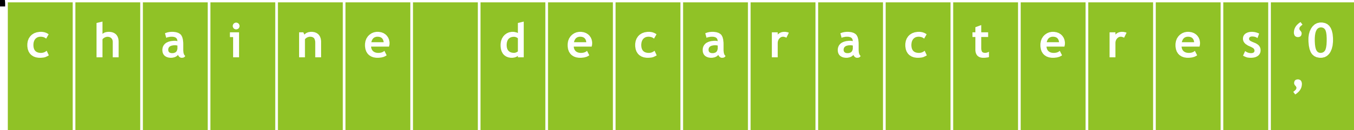
7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.6. Pointeurs et chaînes de caractères

Par exemple: L’utilisation de pointeurs de caractère et non de tableaux permet de créer une chaîne correspondant à la concaténation de deux chaînes de caractères.

```
main(){    int i; char *chaine1, *chaine2, *res, *p;
chaine1 = "chaine ";
chaine2 = "de caracteres";
res = (char*)malloc((strlen(chaine1) + strlen(chaine2)) *
sizeof(char));
p = res;
for (i = 0; i < strlen(chaine1); i++) *p++ = chaine1[i];
for (i = 0; i < strlen(chaine2); i++)*p++  = chaine2[i];
*p='\0'; puts(res);
}
```



Remarque: l’utilisation d’un pointeur intermédiaire p qui est indispensable dès que l’on fait des opérations de type incrémentation.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.6. Pointeurs et chaînes de caractères

En effet, si on avait incrémenté directement la valeur de res, on aurait évidemment “perdu” la référence sur le premier caractère de la chaîne. Par exemple,

```
main(){ int i; char *chaine1, *chaine2, *res;
chaine1 = "chaine ";
chaine2 = "de caracteres";
res = (char*)malloc((strlen(chaine1) + strlen(chaine2)) * sizeof(char));
for (i = 0; i < strlen(chaine1); i++) *res++ = chaine1[i];
printf("\nNombre de caracteres de res=%d et %d\n",strlen(res),strlen(chaine1));
for (i = 0; i < strlen(chaine2); i++) *res++ = chaine2[i];
printf("\nNombre de caracteres de res = %d et %d\n",strlen(res),strlen(chaine2));
free(res);
}
```

1. **Nombre de caracteres de res= 0 et 7**
2. **Nombre de caracteres de res= 10 et 13**

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.6. Pointeurs et chaînes de caractères

- ❑ Recopie d'une chaîne de caractères strcpy, strncpy

➔ Prototype dans <string.h>:

char * strcpy (char * destination, char * source);

- ➢ Copie la chaîne pointée par ***source*** vers la chaîne pointée par ***destination***.
- ➢ Pour éviter des débordements de mémoire il faut s'assurer que la taille du tableau pointé par destination est suffisante pour contenir la chaîne ***source.***
- ➢ Le pointeur ***destination*** est retourné.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.6. Pointeurs et chaînes de caractères

❑ Recopie d'une chaîne de caractères strcpy, strncpy

➔ Prototype dans <string.h>:

char * strncpy (char * *destination*, char * *source*, int N);

- Copie les **N** premiers caractères de **source** vers **destination.**
- Si la fin de la chaîne **source** est rencontré avant que **N** caractères soient copiés **destination** est complétés de caractères **nul ('\0')**.
- Le pointeur **destination** est retourné.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.6. Pointeurs et chaînes de caractères

- ❑ Concaténation de chaînes strcat, strncat (Prototype dans <string.h>)

char * strcat (char * destination, char * source);

- ➢ Ajoute une copie de la chaîne **source** à la chaîne **destination**.
- ➢ La caractère de fin de chaîne de **destination** est remplacé par le premier caractère de **source**, et un nouveau caractère de fin de chaîne est ajouté à la fin de la nouvelle chaîne résultante **destination**.
- ➢ Le tableau de caractères pointé par **destination** doit être suffisamment grand pour accueillir la chaîne concaténée.
- ➢ Le pointeur **destination** est retourné.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.6. Pointeurs et chaînes de caractères

❑ Concaténation de chaînes strcat, strncat (Prototype dans <string.h>)

char * strncat (char * destination, char * source, int N);

➢ Ajoute les **N** premiers caractères de **source** à **destination**, plus un caractère fin de chaîne.

➢ Si la longueur de la chaîne **source** est inférieure à **N**, seul les caractères précédents le caractère fin de chaîne sont recopiés.

➢ Le tableau de caractères pointé par **destination** doit être suffisamment grand pour accueillir la chaîne concaténée.

➢ Le pointeur **destination** est retourné.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.6. Pointeurs et chaînes de caractères

- Comparaison de chaînes strcmp, strncmp (Prototype dans <string.h>)

int strcmp (char * str1, char * str2);

- Compare les chaînes **str1** et **str2**. Cette fonction débute par la comparaison du premier caractère de chaque chaîne. S'ils sont égaux, elle continue avec la paire de caractères suivants tant que les caractères sont égaux et qu'un caractère fin de chaîne n'est pas atteint.
- Cette fonction retourne un entier indiquant la relation d'ordre entre les deux chaînes :
  - ✓ Un zéro signifie que les deux chaînes sont égales.
  - ✓ Une valeur positive indique que le premier caractère différent a une plus grande valeur dans **str1** que dans **str2.**
  - ✓ Une valeur négative indique l'inverse.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.6. Pointeurs et chaînes de caractères

- ❑ Comparaison de chaînes strcmp, strncmp (Prototype dans <string.h>)

int strncmp (char * *str1*, char * *str2*, int N);

- ➢ Cette fonction débute par la comparaison du premier caractère de chaque chaîne. S'ils sont égaux, elle continue avec la paire de caractères suivantes tant que les caractères sont égaux et qu'un caractère fin de chaîne n'est pas atteint et que **N** caractères n'ont pas été comparés.
- ➢ Cette fonction retourne un entier indiquant la relation d'ordre entre les deux chaînes :
- ✓ Un zéro signifie que les **N** premiers caractères des deux chaînes sont égaux.
- ✓ Une valeur positive indique que le premier caractère différent a une plus grande valeur dans **str1** que dans **str2**
- ✓ Une valeur négative indique l'inverse.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.6. Pointeurs et chaînes de caractères

- Longueur d'une chaîne strlen Prototype dans <string.h>

strlen (char * str);

- Retourne un entier égal à la longuer de la chaîne ***str***.

- La longueur d'une chaîne correspond au nombre de caractère entre le début de la chaîne et le caractère fin de chaîne **'\0'**. A ne pas confondre avec la taille du tableau, par exemple : **char MaChaine[100]="chaine test";**

- définir un tableau de **100** caractères mais la chaîne **MaChaine** a une longuer de **11** caractères. Par conséquent **sizeof(MaChaine)** retourne **100**, et **strlen(MaChaine)** retourne **11**.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.6. Pointeurs et chaînes de caractères

- Recherche d'une chaîne dans une autre strstr(Prototype dans <string.h>)

char * strstr (char * *str1*, char * *str2*);

- Retourne un pointeur sur la première occurrence de la chaîne **str2** dans la chaine **str1**, ou un pointeur NULL si **str2** n'est pas une partie de **str1**.

- La comparaison ne porte pas sur le caractère fin de chaîne

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.6. Pointeurs et chaînes de caractères

- Recherche de caractères dans une chaîne strchr ➔ (Prototype dans <string.h>

char * strchr(char * *str1*, int *character*);

- Retourne un pointeur sur la première occurrence de *character* dans la chaîne *str1*.
- Le caractère fin de chaîne est considéré, par conséquent cette fonction peut être utilisée pour localiser la fin d'une chaîne.

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.7. Pointeurs et fonctions: Paramètres d'une fonction

- Les paramètres servent à échanger des informations entre la fonction appelante et la fonction appelée. Ils peuvent recevoir des données et stocker des résultats.
- Il existe deux modes de transmission de paramètres dans les langages de programmation :
  - **La transmission par valeur** : les valeurs des paramètres effectifs sont affectées aux paramètres formels correspondants au moment de l'appel de la fonction ou procédure. Dans ce mode le paramètre effectif ne subit aucune modification.
  - **La transmission par adresse (ou par référence)** : les adresses des paramètres effectifs sont transmises à la fonction appelante. Dans ce mode, le paramètre effectif subit les mêmes modifications que le paramètre formel

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.7. Pointeurs et fonctions: Transmission des paramètres

- La transmission des paramètres en C se fait toujours par valeur.
- Pour effectuer une transmission par adresse en C, on déclare le paramètre formel de type pointeur et lors d'un appel de la fonction, on envoie l'adresse et non la valeur du paramètre effectif
- Exemple :

```
void Incrementer (int x, int *y) {
    x  = x+1;
    *y = *y+1;
    }
int main() { int n = 3, m=3;
   Incrementer (n, &m);
   printf("n = %d et m=%d \n", n,m);
   return 0;}
```


- **Résultat : n=3 et m= 4**

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.7. Pointeurs et fonctions: Transmission des paramètres

❑ Exemple :

```
void Incrementer (int *x, int *y) {
   *x=*x+1;
   *y =*y+1;
   }
int main(){ int n = 3, m=3;
      Incrementer (&n, &m);
      printf(“n = %d et m=%d \n", n,m);
      return 0;}
```

Résultat : n=4 et m= 4

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.7. Pointeurs et fonctions: Transmission des paramètres

Exemple; Une fonction qui échange le contenu de deux variables :

```
void Echange (float *x, float *y){ float z;
      z = *x;
     *x = *y;
     *y = z;
     }
int main() {
   float a=2,b=5;
   printf("a=%f,b=%f\n ",a,b);
   Echange(&a,&b);
   printf("a=%f,b=%f\n ",a,b);
   return 0 ;
}
```

Résultat : a=5 et b= 2

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.7. Pointeurs et fonctions: Transmission des paramètres

- Rappelons qu'un tableau est un pointeur (sur le premier élément du tableau).
- Lorsqu'un tableau est transmis comme paramètre à une fonction secondaire, ses éléments sont donc modifiés par la fonction. Par exemple, le programme

```
void init (int *tab, int n){int i;
   for (i = 0; i < n; i++){
      tab[i] = i;
      printf("tab[%d]=%d\n",i,*(tab+i));
      }}
int main(){int n; int *tab;
   printf("donner la valeur de n \n");
   scanf("%d",&n);
   tab = (int*)malloc(n * sizeof(int));
   init(tab,n); /*initialise les éléments du tableau tab*/
   return 0;
   }
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.7. Pointeurs et fonctions: Récursivité

- Une fonction qui fait appel à elle-même est une fonction **récursive**
- Toute fonction récursive doit posséder un cas limite (cas trivial) qui arrête la récursivité.
- Exemple : Calcul du factorielle: n!= n*(n-1)*….*4*3*2*1* 0!

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

Implémentation itérative

```
int factIterative(int n)
{ int i, result=1;
  for (i=1 ; i<=n ; i++)
            result *= i;
   return result;
}
```

N.B: l'ordre de calcul est l'ordre inverse de l'appel de la fonction

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.7. Pointeurs et fonctions: Récursivité

Fonctions récursives : exercice

- Ecrivez une fonction récursive (puis itérative) qui calcule le terme n de la suite de Fibonacci définie par : U(0)=U(1)=1
  U(n)=U(n-1)+U(n-2)

```
int Fib (int n)
{if (n==0 || n==1) return 1;
   else return (Fib(n-1)+Fib(n-2));
}
int main() {
  int N;
  printf("saisir la valeur de N\n");
  scanf("%d",&N);
  printf("Fib(%d)=%d \n ",N,Fib(N));
  return 0;
 }
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.3. Les Pointeurs

7.3.7. Pointeurs et fonctions: Récursivité

Fonctions récursives : exercice

Une fonction itérative pour le calcul de la suite de Fibonacci :

```
int Fib (int n) {
 int i, AvantDernier, Dernier, Nouveau;
 if (n==0 || n==1) return 1;
 AvantDernier=1; Dernier =1;
 for (i=2; i<=n; i++) {
      Nouveau= Dernier+ AvantDernier;
      AvantDernier = Dernier;
      Dernier = Nouveau;
  }
return Nouveau;
}
```


➔ La solution récursive est plus facile à écrire

7.3. Les Pointeurs
7.3.7. Pointeurs et fonctions: Récursivité
Exemple :