RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ ECHAHID HAMMA LAKHDAR D'EL OUED
FACULTÉ DE TECHNOLOGIE
1ÈRE ANNÉE LMD SCIENCES ET TECHNIQUES

Chapter II
Les enregistrements et fichiers

Semester: 2

Matière : Informatique2

Mr. BERHOUM Adel

A - Les enregistrements

1. Les tableaux

Les tableaux sont :

- Des variables structurées,
- Leur traitement en C ne diffère pas de celui dans d'autres langages.
- Le C permet un accès encore plus direct et rapide aux données d'un tableau.

1.2. Tableaux à 1 dimension

1.2.1. Déclaration et mémorisation

- **Déclaration** :

type Nom_du_tableau[dimension] ;

Remarque: Nom_du_tableau est un identificateur

Exemples:

int A[25]; tableau de 25 entiers de type **int**

float B[10]; tableau de 10 décimaux de type **float**

char C[30]; tableau de 30 caractères (entiers de type **char**)

1.2. Tableaux à 1 dimension

1.2.1. Déclaration et mémorisation

- **Mémorisation**:
  - **En C,** Le nom d'un tableau est le représentant de l'adresse du premier élément du tableau, les adresses des autres composantes sont calculées automatiquement, relativement à cette adresse.
  - Si un tableau possède N composantes et si le type déclaré des composantes requiert M octets, la mémoire réservée pour ce tableau est de N×M octets.

1.2. Tableaux à 1 dimension

1.2.1. Déclaration et mémorisation

- **Mémorisation**:

Exemples:

int *A[6];*

Un entier **int** requiert **2 octets**, il y a 6 éléments, la mémoire réservée pour le tableau A est donc de 2×6=12 octets.

1.2. Tableaux à 1 dimension

- Initialisation et réservation automatique

❑ Il est possible d'initialiser un tableau lors de sa déclaration, en indiquant la liste des valeurs

Exemples:

int *A[5]={100,200,300,400,500};*

Le premier élément du tableau, A[0] contiendra la valeur 100. le second élément, A[1],contiendra la valeur 200, le dernier élément, A[4], contiendra la valeur 500.

1.2. Tableaux à 1 dimension

- Initialisation et réservation automatique

❑ Lors de sa déclaration, on peut initialiser le tableau, si le nombre de valeur dans la liste est inférieur à la dimension du tableau, les composantes restantes sont mises à 0.

Exemples:

```
int  A[4]={10,20};

int A[]={10,20,30};
```

1.2. Tableaux à 1 dimension

- Initialisation et réservation automatique

Remarque :

- Il faut bien évidemment que le nombre de valeurs dans la liste soit inférieur ou égal à la dimension du tableau !

Exemples:

int *A[**4**] < ou = {**10,20**};*

1.2. Tableaux à 1 dimension

- Accès aux composantes

❑ Lorsque l'on déclare un tableau, (par exemple **int** A[5] ;), on définit un tableau avec 5 composantes, auxquelles on peut accéder par :

A[0], A[1], A[2], A[3], A[4]

Remarque :

❑ Le premier élément du tableau est l'élément 0, donc, pour un tableau de dimension N, le premier sera l'élément 0, le dernier l'élément N-1.

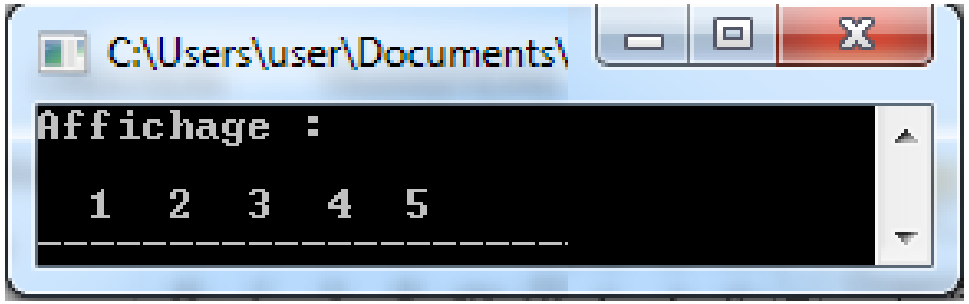
1.2. Tableaux à 1 dimension

- Affichage et affectation

❑ La structure for se prête particulièrement bien au travail avec les tableaux.

Affichage du contenu d'un tableau

```
1   #include <stdio.h>
2   main()
3   {
4       int A[5]={1,2,3,4,5} ;
5       int i ;
6       printf("Affichage :\n\n");
7       for(i=0 ;i<5 ;i++)
8           printf("%3d",A[i]);
9   }
10
```



❑ Avant de pouvoir afficher les composantes d'un tableau, il faut bien sûr leur affecter des valeurs!

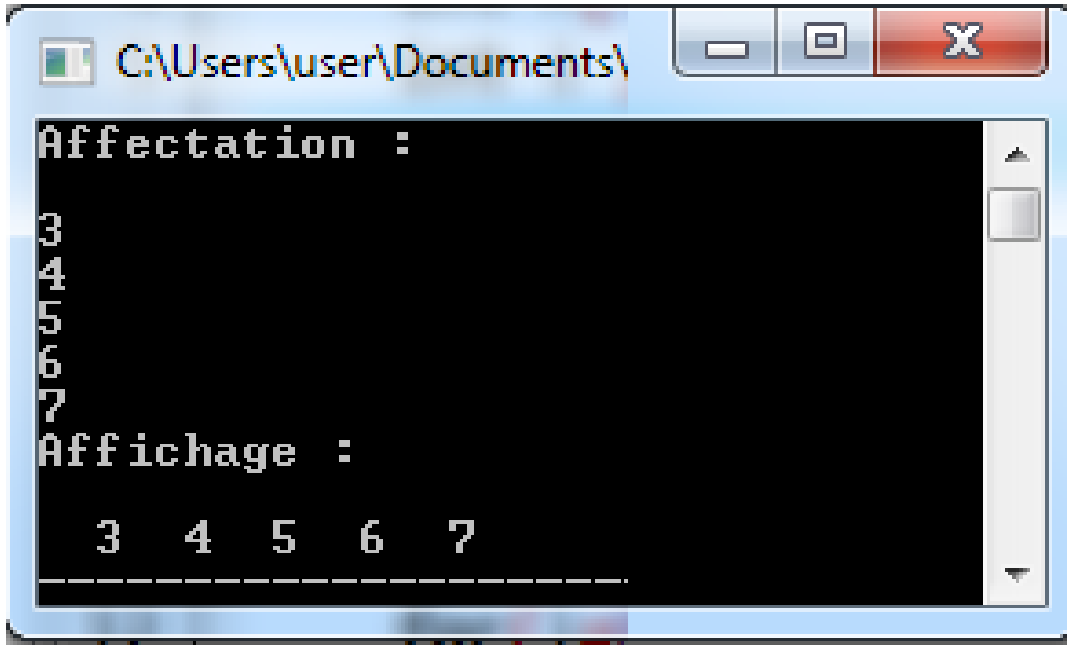
1.2. Tableaux à 1 dimension

- Affichage et affectation

❑ Avant de pouvoir afficher les composantes d'un tableau, il faut bien sûr leur affecter des valeurs!

Affectation

```
1   #include <stdio.h>
2   main()
3   {
4       int A[5];
5       int i ;
6
7       printf("Affectation :\n\n");
8       for(i=0 ;i<5 ;i++)
9           scanf("%d",&A[i]);
10
11      printf("Affichage :\n\n");
12      for(i=0 ;i<5 ;i++)
13          printf("%3d",A[i]);
14  }
```



1.3. Tableaux à 2 dimensions

En C,

- Un tableau A à deux dimensions est à interpréter comme un tableau (à une dimension) de dimension L dont chaque composante est un tableau (unidimensionnel) de dimension C.

- On appelle L le nombre de lignes, C le nombre de colonnes. L et C sont les dimensions du tableau.

1.3. Tableaux à 2 dimensions

En C,

- Un tableau à deux dimensions contient donc L×C composantes.
- Rapprochement avec les maths: "A est un vecteur de L vecteurs de dimension C", ou encore "A est une matrice de dimensions L et C".

1.3. Tableaux à 2 dimensions

1.3.1. Déclaration, mémorisation, réservation automatique

- **Déclaration** :

type Nom_du_tableau[ligne][colonne] ;

Remarque: Nom_du_tableau est un identificateur

Exemples:

int A[10][10]; tableau de 10×10 entiers de type int

float B[5][4]; tableau de 5×4 décimaux de type float

char C[2][25]; tableau de 2×25 caractères

1.3. Tableaux à 2 dimensions

1.3.1. Déclaration, mémorisation, réservation automatique

- **Mémorisation** :

  ❑ Comme pour les tableaux à une dimension, le nom d'un tableau est le représentant de l'adresse du 1er élément du tableau, c'est-à-dire l'adresse de la première ligne. Les composantes d'un tableau sont stockées ligne par ligne

1.3. Tableaux à 2 dimensions

1.3.1. Déclaration, mémorisation, réservation automatique

- **Mémorisation** :

Exemple :

int A[3][2]={{1,2},{10,20},{100,200}}; **tableau de 3×2 entiers**

Si le nombre de lignes ou de colonnes n'est pas déclaré explicitement, l'ordinateur réserve automatiquement le nombre d'octets nécessaires.

1.3. Tableaux à 2 dimensions

1.3.1. Déclaration, mémorisation, réservation automatique

- **Accès aux composantes** :

Comme pour les tableaux à une dimension, les indices des tableaux varient de 0 à N-1. Soit un tableau à deux dimensions A[n][m]:

Premier élément → **A[0][0]** ..... **A[0][m-1]**

: :

: :

A[n-1][0] ..... **A[n-1][m-1]** ← **dernier élément**

1.3. Tableaux à 2 dimensions

1.3.1. Déclaration, mémorisation, réservation automatique

- **Affichage et affectation**:

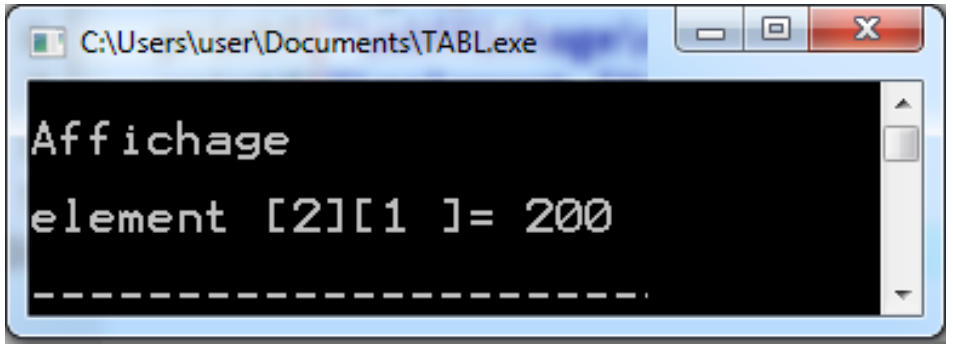
  - Lors du travail avec les tableaux à deux dimensions, on utilisera (souvent imbriqués) deux indices (par exemple i et j), et la structure for, pour parcourir les lignes et les colonnes.

1.3. Tableaux à 2 dimensions

1.3.1. Déclaration, mémorisation, réservation automatique

- **Affichage et affectation**:
- ***Affichage*** :

```
1  #include <stdio.h>
2  main()
3  {
4    int A[3][2]={{1,2},{10,20},{100,200}};
5    int i=2, j=1;
6    printf("\nAffichage\n");
7    printf("\nelement [%d][%d ]= %d\n",i,j,A[i][j]);
8  }
```

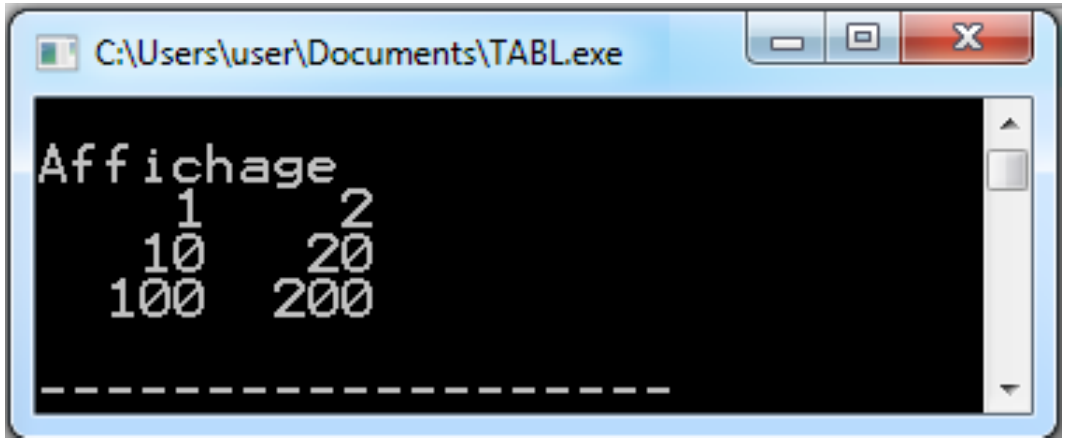


1.3. Tableaux à 2 dimensions

1.3.1. Déclaration, mémorisation, réservation automatique

- **Affichage et affectation**:
- ***Affichage*** :

```
1  #include <stdio.h>
2  main()
3  {
4    printf("\nAffichage\n");
5    int A[3][2]={{1,2},{10,20},{100,200}};
6    int i, j;
7    for(i=0;i<3;i++) /* boucle sur les lignes*/
8      {
9              for(j=0;j<2;j++) /* boucle sur les */
10             { /* colonnes */
11                 printf("%5d",A[i][j]);
12             }
13         printf("\n");
14     }
15 }
```



1.3. Tableaux à 2 dimensions

1.3.1. Déclaration, mémorisation, réservation automatique

- **Affichage et affectation**:
- ***Affectation*** :

```c
#include <stdio.h>
main()
{
  printf("\nAffictation\n");
  int A[5][4];
int i, j;
    for(i=0;i<5;i++) /* boucle sur les lignes*/
        for(j=0;j<4;j++) /* boucle sur les colonnes */
            scanf("%d",&A[i][j]);
}
```

2. Les chaînes de caractères

2.1. Introduction :

- Il n'existe pas de type particulier "chaîne" ou "string" en langage C. Une chaîne de caractères est traitée comme un tableau de caractères à une dimension.
- Il existe tout de même des notations particulières et une bonne quantité de fonctions spéciales pour le traitement des tableaux de caractères.

2. Les chaînes de caractères

2.1. Déclaration, mémorisation,

- **Déclaration** :

char Nom_du_tableau[longueur] ;

❑ Lors de la déclaration (comme pour les tableaux de chiffres), on doit indiquer l'espace à réserver.

❑ La représentation interne d'une chaîne de caractères est terminée par le symbole '\0' (NUL), donc pour un texte de n caractères, il faudra réserver n+1 emplacements.

NB: Il n'y a pas de contrôle de cela à la compilation, s'il y a un problème, celui-ci ne se verra qu'à l'exécution.

2. Les chaînes de caractères

2.1. Déclaration, mémorisation,

- **Mémorisation** :

❑ De même que pour les tableaux de chiffres, le nom de la chaîne est le représentant de son adresse mémoire, et plus précisément le représentant de l'adresse du premier caractère.

Exemple :

char TXT[9]="BONJOUR!";

'B'	'O'	'N'	'J'	'O'	'R'	'!'	/0

↑
TXT
TXT[0]

2. Les chaînes de caractères

2.2. Les chaînes de caractères constantes,

- **Mémorisation** :
  - ❑ Les chaînes de caractères sont indiquées entre guillemets. La chaîne de caractères vide est alors:""
  - ❑ Dans les chaînes de caractères, on peut utiliser les séquences d'échappement, définies comme caractères constants

2. Les chaînes de caractères

2.2. Les chaînes de caractères constantes,

- **Exemples**:

1. **char C[]="Ce texte\n s'écrira sur \n 3 lignes";**
2. **char txt[]={'L','\'' ','a','s','t','u','c','e','\0'};**

❑ Plusieurs chaînes de caractères constantes qui sont séparées par des signes d'espacement dans le texte du programme seront réunies en une seule chaîne lors de la compilation:

3. **char A[]="un ""deux ""trois";**

❑ La chaîne contenue dans A sera évaluée "un deux trois". Il est ainsi possible de définir de très longues chaînes de caractères constantes utilisant plusieurs lignes dans le texte du programme.

2. Les chaînes de caractères

2.2. Les chaînes de caractères constantes,

- **NB**:
  - 'x' est un caractère constant (qui a une valeur numérique 'x'=120) (codé sur 1 octet)
  - "x" est une chaîne de caractères qui contient 2 caractères 'x' et '\0' (codé sur 2 octets)

2. Les chaînes de caractères

2.3. Initialisation de chaînes de caractères,

- En général, on initialise les tableaux en indiquant la liste de ses éléments entre accolades:

```
char chaine[]={'h','e','l','l','o','\0'};
```

- Mais, pour une chaîne de caractères, on peut utiliser:

```
char chaine[]= "hello";
```

Remarque : Si on ne spécifie pas la longueur, le bon nombre d'octets est réservé (y compris le '\0').

2. Les chaînes de caractères

2.3. Initialisation de chaînes de caractères, ,

- **Représentation en mémoire** :

char txt[] ="hello";

‘h’	‘e’	‘l’	‘l’	‘o’	\0

char txt[6]="hello";

‘h’	‘e’	‘l’	‘l’	‘o’	\0

char txt[8]="hello";

‘h’	‘e’	‘l’	‘l’	‘o’	\0	0	0

char txt[5]="hello";

‘h’	‘e’	‘l’	‘l’	‘o’

❑ Passe à la compilation, erreur à l'exécution

char txt[4]="hello";

‘h’	‘e’	‘l’	‘l’

❑ Erreur à la compilation

2. Les chaînes de caractères

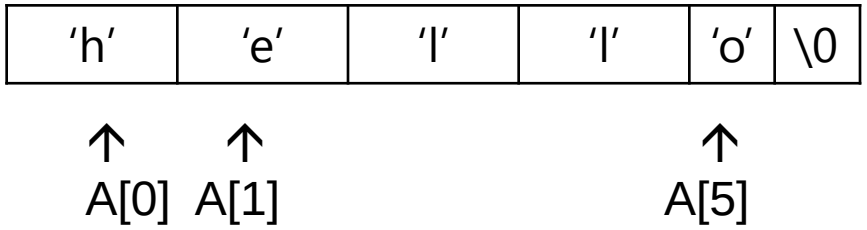
2.3. Accès aux éléments d'une chaîne,

L'accès se fait de la même manière que pour un élément d'un tableau.

En déclarant:

char A[6]="hello";

on a:

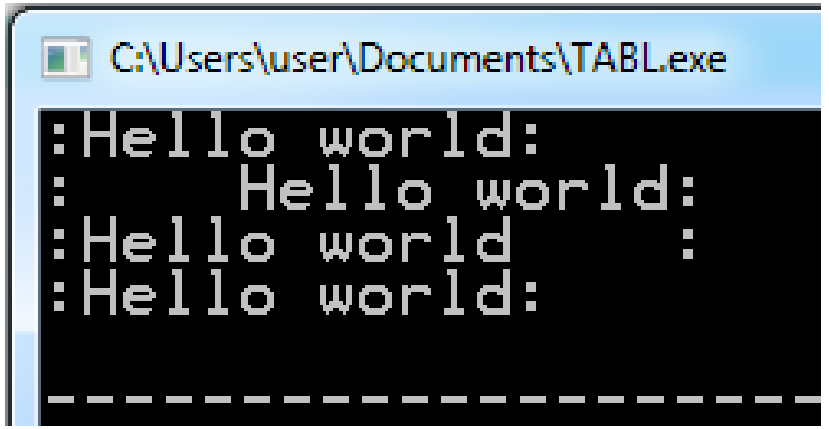


a) Les fonctions de stdio.h

Affichage

- **printf (avec le spécificateur %s)**

```
1  #include <stdio.h>
2  main()
3  {
4    char NOM[]="Hello world";
5  printf(":%s:\n",NOM);
6  printf(":%15s:\n",NOM);
7  printf(":%-15s:\n",NOM);
8  printf(":%5s:\n",NOM);
9  }
```

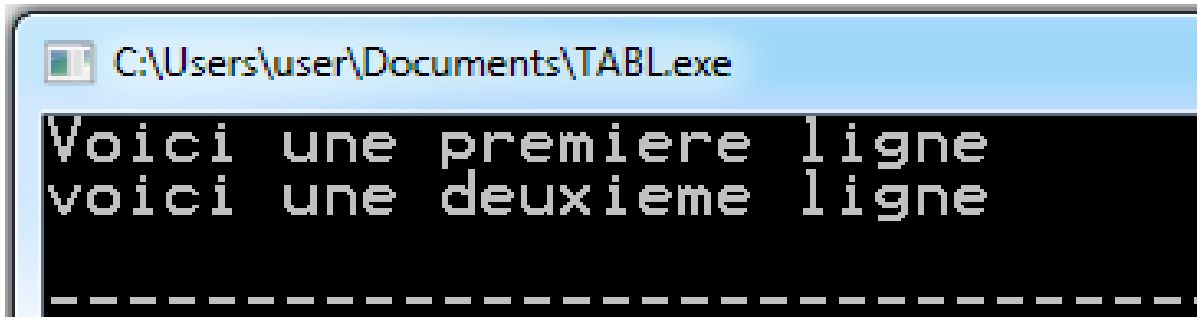


a) Les fonctions de stdio.h

Affichage

- **puts**

```
1  #include <stdio.h>
2  main()
3  {
4    char texte[]="Voici une premiere ligne";
5  puts(texte);
6  puts("voici une deuxieme ligne");
7  }
```



Equivalence: puts(ch) = printf("%s\n",ch)

a) Les fonctions de stdio.h

Lecture

- **scanf (avec le spécificateur %s)**

❑ Le nom de la chaîne est le représentant de l'adresse du premier caractère, il est donc inutile (et interdit!) de mettre &.

```
1  #include <stdio.h>
2  main()
3  {
4    char lieu[30];
5  int jour, mois, annee;
6  printf("Entrez vos lieu et date de naissance:\n");
7  scanf("%s %d %d %d",lieu,&jour,&mois,&annee);
8  }
```

a) Les fonctions de stdio.h

Lecture

- **gets**

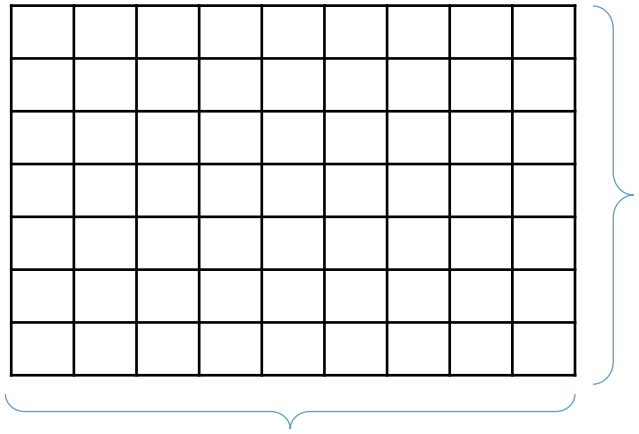
gets(ch): lit une (ou plusieurs) lignes de caractères et la (les) copie à l'adresse indiquée par ch.
Le retour à la ligne final est remplacé par \0&.

```
1  #include <stdio.h>
2  main()
3  {
4    int MAX=1000;
5  char ligne[MAX];
6  gets(ligne);
7  }
```

a) Déclaration, initialisation, mémorisation

type nom[L][C];

Exemple :
char jour[7][9]; réserve l'espace en mémoire pour 7 chaînes de caractères contenant 9 caractères(8 significatifs)



Exemple :
char jour[7][9]={"lundi","mardi","mercredi","jeudi","vendredi","samedi","dimanche"};

l	u	n	d	i	\0	0	0	0
m	a	r	d	i	\0	0	0	0
m	e	r	c	r	e	d	i	\0
j	e	u	d	i	\0	0	0	0
v	e	n	d	r	e	d	i	\0
s	a	m	e	d	i	\0	0	0
d	i	m	a	n	c	h	e	\0

b) Accès aux différentes composantes

- Il est possible d'accéder aux différentes chaînes de caractères d'un tableau en indiquant la ligne correspondante.

Exemple :

```
char jour[7][9]={"lundi","mardi","mercredi","jeudi","vendredi","samedi","dimanche"};
printf("Aujourd'hui, c'est %s!\n",jour[2]);
```

affichage: Aujourd'hui, c'est mercredi!

b) Accès aux différentes composantes

- jour[j] représente l'adresse du premier élément d'une chaîne de caractère, on ne peut donc modifier une telle adresse par une affectation du type:

~~jour[4]="friday";~~

On doit utiliser la fonction **strcpy(jour[4],"friday")**

- Il est possible d'accéder aux différents caractères qui composent le tableau:

```
for(i=0;i<7;i++)
        printf("%c\t", jour[i][0];
```

Affichage:
l m m j v s d

3. Les pointeurs

- Les pointeurs existent dans la plupart des langages de programmation, ils permettent d'accéder à la mémoire de l'ordinateur. Ce sont des variables auxquelles on peut attribuer les adresses d'autres variables.
- En C, ils jouent un rôle primordial dans la définition de fonctions: comme le passage des paramètres en C se fait toujours par la valeur, les pointeurs sont le seul moyen de changer le contenu de variables déclarées dans d'autres fonctions. Ainsi, le traitement de tableaux ou de chaînes de caractères serait impossible sans l'utilisation des pointeurs.

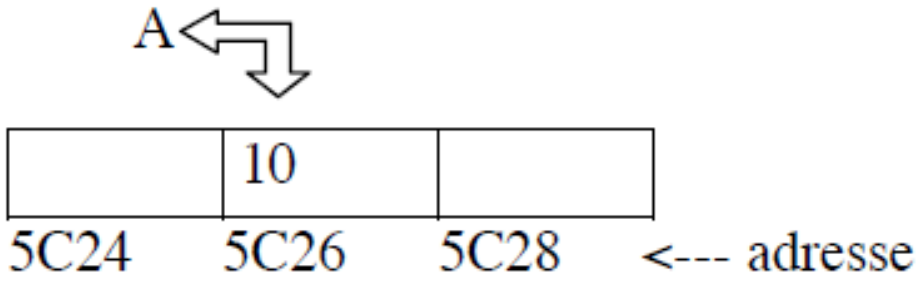
3. Les pointeurs

3.1. Adressage de variables

❑ Il existe deux modes d'adressage principaux: l'adressage direct, et indirect.

a) Adressage direct = Accès au contenu d'une variable par le nom de cette variable

Exemple: int A; On déclare une variable A, A=10; On lui affecte la valeur 10 (son contenu vaut 10)
Représentation en mémoire:



3. Les pointeurs

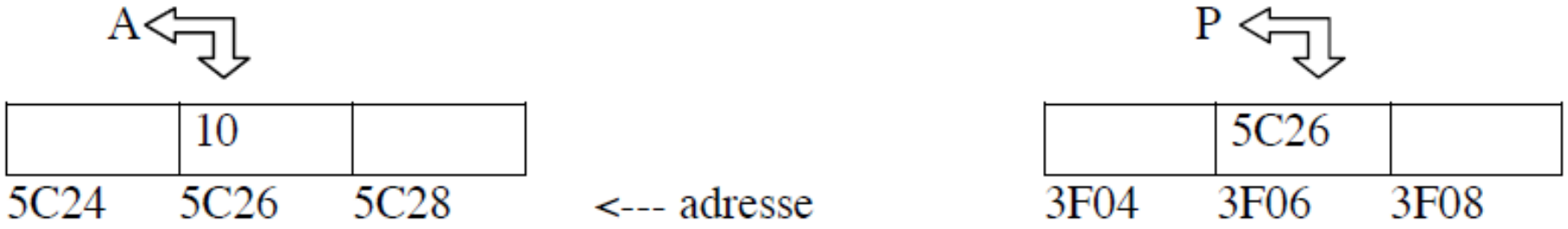
3.1. Adressage de variables

a) Adressage indirect

Si on ne veut, ou on ne peut utiliser le nom d'une variable A, on peut copier l'adresse de cette variable dans une variable spéciale P, appelée pointeur. On peut ainsi, par la suite, retrouver l'information de A en passant par P.

Représentation en mémoire:

A : variable contenant la valeur 10
P : pointeur (variable) contenant l'adresse de A (&A ou 5C26)



3. Les pointeurs

3.2. Les pointeurs

Définition: Un pointeur est une variable spéciale qui peut contenir l'adresse d'une autre variable.

- **En C**, chaque pointeur est limité à un type de données. Il peut contenir l'adresse d'une variable simple de ce type, ou l'adresse d'une composante d'un tableau de ce type. Si un pointeur P contient l'adresse d'une variable A, on dit que P pointe sur A.

Remarques:

Pointeurs et noms de variables ont le même rôle: ils donnent accès à un emplacement dans la mémoire interne de l'ordinateur, il faut tout de même bien faire la différence:

- Un pointeur est une variable qui peut "pointer" sur différentes adresses.
- Le nom d'une variable reste toujours liée à la même adresse.

3. Les pointeurs

3.2. Les pointeurs

a) Les opérateurs de base

- **L'opérateur "adresse de": &** (pour obtenir l'adresse d'une variable)

&var fournit l'adresse de la variable var (déjà vu dans **scanf**)

Exemple: *P=&A;* On affecte à P l'adresse de A

- **L'opérateur "contenu de": ***

***pointeur**: désigne le contenu de l'adresse référencée par pointeur

Exemple:

A: variable contenant la valeur 10 et B: variable contenant la valeur 50

P pointeur non initialisé

P=&A; P pointe sur A

B=***P**; // affecte à B le contenu de l'adresse référencée par P (i.e. le contenu de A) => B=10

***P**=20; // le contenu de l'adresse référencée par P (i.e. le contenu de A) est mis à 20 => A=20

3. Les pointeurs

3.2. Les pointeurs

a) Les opérateurs de base

❑ Déclaration d'un pointeur

Syntaxe:

type *nom_pointeur

Déclare un pointeur (nom_pointeur) qui peut recevoir des adresses de variables de type "type".

Exemple: int *p; Déclare un pointeur p sur int

Lors de la déclaration d'un pointeur en C, ce pointeur est lié explicitement à un type de données. Ainsi, la variable P déclarée comme pointeur sur int, ne peut pas recevoir l'adresse d'une variable d'un autre type que int.

3.2. Les pointeurs

b) Les opérateurs élémentaires sur pointeurs

- Priorités de * (contenu de) et & (adresse de)

***** et **&** ont la même priorité que les opérateurs unaires **(!, ++, --),** dans une même expression,

les opérateurs unaires ***, &, !, ++, --** sont évalués de droite à gauche.

- Si p pointe sur x, alors ***p** peut être utilisé partout où l'on peut écrire x.

Exemple:

```
y=*p+1     ⇔  y=x+1
*p=*p+10   ⇔  x=x+10
*p+=2      ⇔  x+=2
++*p       ⇔  ++x
(*p)++     ⇔  x++
```

3. Les pointeurs

3.3. Pointeurs et tableaux

Il existe une relation très étroite entre tableaux et pointeurs, ainsi, chaque opération avec des indices de tableaux peut être exprimée à l'aide de pointeurs. En général, les versions formulées avec pointeur sont plus compactes et plus efficaces, surtout dans les fonctions .

3. Les pointeurs

3.3. Pointeurs et tableaux

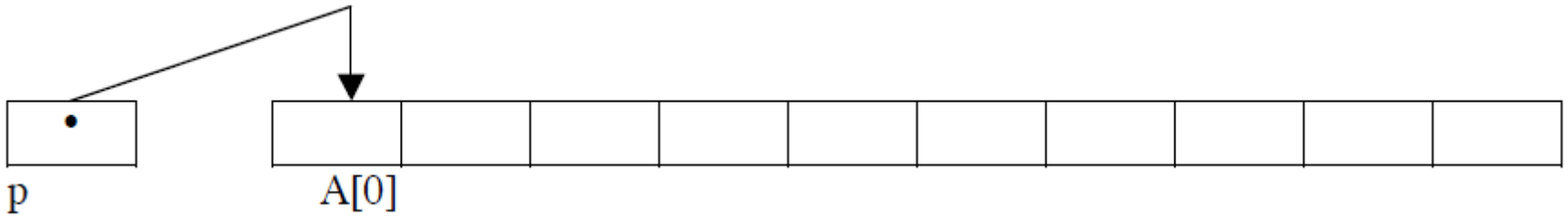
a) Adressage des composantes d'un tableau

On a vu au chapitre VII que le nom d'un tableau représentait l'adresse de son premier élément.

tableau ⇔ &tableau[0]

Exemple:

int A[10];
int *p;
p=A; est équivalente à p=&A[0];



Si p pointe sur une composante quelconque d'un tableau, p+1 pointe sur la composante suivante.

3. Les pointeurs

3.3. Pointeurs et tableaux

a) Adressage des composantes d'un tableau

En résumé:
A: tableau de type quelconque (i: indice)
p: pointeur de même type que A
A désigne l'adresse de A[0]
A+i désigne l'adresse de A[i]
*(A+i) désigne le contenu A[i]
Si p=A
p pointe sur A[0]
p+i pointe sur A[i]

*(p+i) désigne le contenu de A[i]

- On peut donc passer du formalisme "tableau" au formalisme "pointeur", en remplaçant tab[i] par *(tab+i).

b) Arithmétique des pointeurs

- Le C soutient une série d'opérateurs arithmétiques sur les pointeurs que l'on ne rencontre en général que dans les langages machine.
- Toutes les opérations avec les pointeurs tiennent compte du type et de la grandeur des objets pointés.

- **Affectation par un pointeur de même type**

  p1=p2; p2 pointe sur le même objet que p1

- **Addition et soustraction d'un nombre entier**

Si p pointe sur A[i] (élément i d'un tableau A), alors:

p+n pointe sur A[i+n]

p-n pointe sur A[i-n]

b) Arithmétique des pointeurs

- Incrémentation et décrémentation

Si p pointe sur A[i], alors:

p++; p	pointe sur A[i+1]
p+=n;	p pointe sur A[i+n]
p--; p	pointe sur A[i-1]
p-=n;	p pointe sur A[i-n]

MAIS: Attention à ne pas sortir du tableau!!!

- Soustraction de deux pointeurs

p1 et p2 pointent dans le même tableau

p1 - p2 < 0	si	p1 précède p2
p1 - p2 = 0	si	p1 = p2 (p1 pointe au même endroit que p2)
p1 - p2 > 0	si	p2 précède p1
p1 - p2 est indéfini	si	p1 et p2 ne pointent pas dans le même tableau

b) Arithmétique des pointeurs

- De manière générale, la soustraction de deux pointeurs pointant dans le même tableau est équivalente à la soustraction des indices correspondants.

- Comparaison de deux pointeurs (<, >, <=, >=, = =, !=)

La comparaison est équivalente à la comparaison des indices, si les deux pointeurs ne pointent pas dans le même tableau, le résultat est donné par leurs positions relatives dans la mémoire

c) Pointeurs et chaînes de caractères

De la même manière que pour les **int** (ou **float**, ou **double**, ...), un pointeur char peut pointer sur un caractère isolé ou sur les éléments d'un tableau de caractères (une chaîne).

- **Affectation**

On peut attribuer l'adresse d'une chaîne de caractères constante:

char *c; c="Ceci est une **chaîne constante**";

❑ On pourra lire cette chaîne de caractères, mais il est recommandé de ne pas la modifier.

c) Pointeurs et chaînes de caractères

- **Initialisation**

Un pointeur sur **char** peut être initialisé lors de sa déclaration, si on lui affecte l'adresse d'une chaîne de caractères constante.

char *b="Bonjour!!! ";

- **Remarques**:

- ❑ On utilise des tableaux de caractères pour déclarer des chaînes de caractères que nous voulons modifier.
- ❑ On utilise des pointeurs sur char pour manipuler des chaînes de caractères constantes (dont le contenu ne change pas).
- ❑ On utilisera de préférence des pointeurs pour effectuer des manipulations à l'intérieur des tableaux de caractères.

d) Pointeurs et tableaux à deux dimensions

L'arithmétique des pointeurs se laisse élargir avec toutes ses conséquences sur les tableaux à 2 dimensions.

- **Exemple**:

Soit un tableau **int** M[4][10]= {{0,1,2,3,4,5,6,7,8,9,},
{10,11,12,13,14,15,16,17,18,19},
{20,21,22,23,24,25,26,27,28,29},
{30,31,32,33,34,35,36,37,38,39}};

Le nom du tableau M représente l'adresse du premier élément du tableau et pointe sur le tableau M[0] qui a pour valeur: {0,1,2,3,4,5,6,7,8,9,}.

d) Pointeurs et tableaux à deux dimensions

- L'expression M+1 est l'adresse du deuxième élément du tableau et pointe sur M[1] qui a pour valeur {10,11,12,13,14,15,16,17,18,19} ⇒ M+i désigne l'adresse du tableau M[i].
- Comment accéder à l'aide de pointeurs aux éléments de chaque composante du tableau M[0][0], …?

int *p;
p=(int *)M; Conversion forcée (obligatoire pour les pointeurs sur les tableaux à 2D)

- Dû à la mémorisation ligne par ligne des tableaux 2D, on peut maintenant traiter M à l'aide de p comme un tableau unidimensionnel de dimension 4×10=40.
- *(p+i) sera donc le contenu de l'élément de la ligne E(i/10) et de la colonne i%10

d) Pointeurs et tableaux à deux dimensions

- **Exemple**:

```
int A[3][4];
A[0][0]=1;
A[0][1]=2;
A[1][0]=10;
A[1][1]=20;
```

L'adresse de l'élément **A[i][j]** se calcule donc: **A+i×4+j**

Pour pouvoir travailler à l'aide de pointeurs sur un tableau à **2D**, il faut connaître 4 données:

- ❑ l'adresse du premier élément du tableau converti dans le type simple des éléments du tableau **(int *M).**
- ❑ La longueur d'une ligne réservée en mémoire (Cf. déclaration).
- ❑ Le nombre d'éléments effectivement utilisés dans une ligne.
- ❑ Le nombre de lignes effectivement utilisées.

4. Tableaux de pointeurs

- Si l'on a besoin d'un ensemble de pointeurs du même type, on peut les réunir dans un tableau de pointeurs.

- **Déclaration**

type *nom[N];

- **Exemple**:

double *A[10]; Déclare un tableau de 10 pointeurs sur des rationnels de type double dont les adresses et les valeurs ne sont pas encore définies.

- On utilise en général les tableaux de pointeurs pour mémoriser de façon économique des chaînes de caractères de différentes longueurs. On considèrera donc essentiellement des tableaux de pointeurs sur des chaînes de caractères.

4. Tableaux de pointeurs

- **Initialisation**

char *jour={"lundi";"mardi";"mercredi";"jeudi";"vendredi";"samedi";"dimanche"};

- Déclare un tableau jour[] de 7 pointeurs sur char. Chacun des pointeurs est initialisé avec l'adresse de l'une des 7 chaînes de caractères.

Ainsi:

jour[i] désigne l'adresse contenue dans l'élément i de jour = l'adresse de la première composante

- jour[i]+j désigne l'adresse de la j-ième composante
- *(jour[i]+j) désigne le contenu de la j-ième composante

4. Tableaux de pointeurs

- **Initialisation**

char *jour={"lundi";"mardi";"mercredi";"jeudi";"vendredi";"samedi";"dimanche"};

- Déclare un tableau jour[] de 7 pointeurs sur char. Chacun des pointeurs est initialisé avec l'adresse de l'une des 7 chaînes de caractères.

Ainsi:

jour[i] désigne l'adresse contenue dans l'élément i de jour = l'adresse de la première composante

- jour[i]+j désigne l'adresse de la j-ième composante
- *(jour[i]+j) désigne le contenu de la j-ième composante

5. Allocation dynamique de mémoire

a) Déclaration statique de données

Dans un programme, chaque variable a besoin d'un certain nombre d'octets en mémoire. Jusqu'ici, la réservation de la mémoire s'est déroulée automatiquement par l'emploi des déclarations des variables. Dans tous les cas, le nombre d'octets à réserver était déjà connu pendant la compilation. C'est la déclaration statique des variables.

5. Allocation dynamique de mémoire

a) Déclaration statique de données

- **Exemple**:

int a; réservation de 2 octets

char B[10][20]; réservation de 200 octets

float c; réservation de 4 octets

❑ Le nombre p d'octets à réserver pour un pointeur dépend de la machine, mais il est connu à la compilation.

double *g; réservation de p octets

char *H; réservation de p octets

float I[10]; réservation de 10×p octets

5. Allocation dynamique de mémoire

a) Allocation dynamique

- On doit souvent travailler avec des données dont on ne peut pas prévoir le nombre et la grandeur lors de la programmation. Ce serait un gaspillage de réserver toujours l'espace prévisible. Il existe un moyen de gérer la mémoire lors de l'exécution du programme.
- **Exemple**: **char *texte[10];**
- Pour 10 pointeurs: 10×p octets. Mais on ne peut savoir à l'avance le nombre d'octets à réserver pour les phrases elles-mêmes, puisqu'elles ne seront introduites que lors de l'exécution du programme.
- La réservation de la mémoire pour ces 10 phrases peut se faire seulement pendant l'exécution du programme: c'est l'allocation dynamique de la mémoire.

B. Les fichiers

Introduction

En C, les communications d'un programme avec son environnement se font par l'intermédiaire de fichiers. Pour le programmateur, tous les périphériques, y compris le clavier et l'écran, sont des fichiers. Jusqu'ici, on a lu des données dans le fichier d'entrée standard (le clavier) et on les a écrits dans le fichier de sortie standard (l'écran).

On va voir dans ce chapitre que l'on peut créer, lire et modifier nous même des fichiers sur les périphériques disponibles.

B. Les fichiers

1. Définition

- Un fichier (en anglais: file) est un ensemble de données stockées en général sur un support externe (disque dur, disquette, CD, bande magnétique, …). Un fichier structuré contient une suite d'enregistrements homogènes, qui regroupent le plus souvent plusieurs composantes appartenant à un ensemble (champs).
- Dans les fichiers séquentiels, les enregistrements sont mémorisés consécutivement dans l'ordre de leur entrée et peuvent seulement être lus dans cet ordre. Si on a besoin d'un enregistrement précis, il faut lire tous les enregistrements qui le précèdent, en commençant par le premier.

B. Les fichiers

1. Définition

Les fichiers séquentiels que nous allons considérer auront les propriétés suivantes:

- les fichiers se trouvent soit en état d'écriture, soit en état de lecture, on ne pourra pas simultanément lire et écrire dans un fichier.
- à un moment donné, on peut uniquement accéder à un seul enregistrement: celui qui se trouve en face de la tête de lecture.
- après chaque accès, la tête de lecture/écriture est déplacée derrière la donnée lue/écrite en dernier lieu.

B. Les fichiers

1. L'accès aux fichiers séquentiels

- Avant de lire ou d'écrire dans un fichier, l'accès est notifié par fopen. fopen accepte le nom d'un fichier, négocie avec le système d'exploitation et fournit un pointeur spécifique qui sera ensuite utilisé lors de l'écriture/lecture du fichier.
- Après traitement, il faut annuler la liaison entre le nom du fichier et le pointeur à l'aide de la commande fclose.
- On peut dire de manière plus simple qu'entre fopen et fclose le fichier est ouvert

B. Les fichiers

1. La mémoire tampon

Pour des raisons d'efficacité, les accès à un fichier se font par l'intermédiaire d'une mémoire tampon (en anglais: buffer); La mémoire tampon est une zone de la mémoire centrale de la machine réservée à un ou plusieurs enregistrements du fichier. L'utilisation de la mémoire tampon a l'effet de réduire le nombre d'accès à la périphérie d'une part et le nombre des mouvements de la tête de lecture/écriture d'autre part.

B. Les fichiers

1. L'accès aux fichiers séquentiels

- Avant de lire ou d'écrire dans un fichier, l'accès est notifié par fopen. fopen accepte le nom d'un fichier, négocie avec le système d'exploitation et fournit un pointeur spécifique qui sera ensuite utilisé lors de l'écriture/lecture du fichier. Après traitement, il faut annuler la liaison entre le nom du fichier et le pointeur à l'aide de la commande fclose.

- On peut dire de manière plus simple qu'entre fopen et fclose le fichier est ouvert.

B. Les fichiers

1. L'accès aux fichiers séquentiels

a) Le type *FILE

Pour pouvoir travailler avec un fichier, un programme a besoin d'un certain nombre d'informations au sujet de ce fichier:

- l'adresse de la mémoire tampon
- la position actuelle de la tête de lecture/écriture
- le type d'accès au fichier: lecture ou écriture
- l'état d'erreur
- ...

On n'aura pas à s'occuper de ces informations, elles sont rassemblées dans une structure de type spécifique FILE. Lorsque l'on ouvre un fichier avec la commande fopen, le système génère automatiquement un bloc du type FILE et nous fournit son adresse.

B. Les fichiers

1. L'accès aux fichiers séquentiels

a) Le type *FILE

Tout ce que l'on a à faire est:

- ❑ Déclarer un pointeur de type *FILE pour chaque fichier dont on a besoin (ex: FILE *fich;)
- ❑ Affecter l'adresse retournée par fopen à ce pointeur
- ❑ Employer le pointeur à la place du nom du fichier dans toutes les instructions de lecture/écriture
- ❑ Libérer le pointeur à la fin du traitement à l'aide de fclose

1. L'accès aux fichiers séquentiels

b) fopen: ouverture

fopen affecte au pointeur de type FILE l'adresse du fichier.

fich=fopen(nom_de_fichier, mode);

- nom_de_fichier est une chaîne de caractères, soit "fichier.dat", soit chaine[20], si chaine[20] a été déclarée de type char et initialisée
- **mode est le mode de lecture/écriture**
- "w" (write): mode écriture (si le fichier n'existe pas, la fonction le crée, si le fichier existe, la fonction le vide)
- "r" (read): mode lecture (si le fichier n'existe pas, la fonction ne le crée pas)
- "a" (ajout): mode ajout écriture (si le fichier n'existe pas, la fonction le crée, si le fichier existe, les écriture auront lieu à la fin du fichier)

B. Les fichiers

1. L'accès aux fichiers séquentiels

b) fopen: ouverture

Le type d'ouverture peut être agrémenté de deux caractères:

- "b": le fichier est considéré en mode binaire. Il peut donc contenir des données qui sont transférées sans interprétation par les fonctions de la bibliothèque
- "+": le fichier est ouvert dans le mode complémentaire au mode de base.

- **Exemple**:

"r+" le fichier est ouvert en mode lecture et plus, c'est-à-dire lecture + écriture
Si le système ne peut pas ouvrir le fichier, il retourne un pointeur nul (0)

- **Exemple**:

```
fich1=fopen(fichier.dat","r");
iIf(fich1==0)printf("Le fichier n'a pas été ouvert\n");
```

1. L'accès aux fichiers séquentiels

c) Les instructions de lecture et d'écriture

Elles sont analogues à putchar, getchar, printf et scanf. MAIS on ajoute un "f" avant et l'on ajoute en argument le nom du pointeur de type *FILE.

- **Ecriture d'un caractère**

fputc('a',fich) Ecrira le caractère a dans le fichier pointé par fich

- **Lecture d'un caractère**

lettre=fgetc(fich) l it un caractère dans le fichier pointé par fich et le "met" dans lettre

B. Les fichiers

1. L'accès aux fichiers séquentiels

c) Les instructions de lecture et d'écriture

Elles sont analogues à putchar, getchar, printf et scanf. MAIS on ajoute un "f" avant et l'on ajoute en argument le nom du pointeur de type *FILE.

- **Ecriture d'une information**

fprintf("fich,"la valeur est %f\n",expr1)

❑ Ecrira dans le fichier pointé par fich le texte "la valeur est ", suivi de la valeur de expr1 au format float et d'un retour à la ligne

- **Lecture d'une information**

fscanf("fich," %d",&n)

❑ Lira dans le fichier pointé par fich la va

1. L'accès aux fichiers séquentiels

d) Détection de la fin d'un fichier

feof(fich) retourne 0 si la tête de lecture référencée par fich n'est pas à la fin du fichier, retourne 1 si elle est à la fin

- **Exemple**: boucle typique pour lire les enregistrements

```
while(!feof(fich))
{
        fscanf(fich,"%s",NOM);
}
```

1. L'accès aux fichiers séquentiels

e) Résumé sur les fichiers

Résumé sur les fichiers:

• Déclaration du pointeur	FILE FILE *fp;
• Ouverture en écriture	fp=fopen(NOM,"w");
• Ouverture en lecture	fp=fopen(NOM,"r");
• Fermeture	fclose(fp);
• Fin de fichier	feof (fp);
• Ecriture	fprintf("fp,"…",variable);
•	fputc(char,fp);
• Lecture	fscanf("fp,"…",&variable);
	c=fgetc(fp);

- **Exemple**: créer et afficher un fichier séquentiel