

TP 5 : Tableaux et chaînes de caractères

Jusqu'à présent nous avons manipulé au plus 3 nombres en même temps. Pour ce dernier, nous avons déclaré au plus 3 variables pour stocker les valeurs lues (c'est le cas de l'ordonnement de 3 valeurs). Supposant maintenant qu'on veut ordonner N nombres ($N > 3$), comment peut-on stocker ces nombres ? En déclarant N variables ? Comment faire si $N > 100$, est-il possible de déclarer 100 variables ?

En algorithmique, des structures de données sont offertes pour stocker et manipuler plusieurs nombres en même temps c'est le cas des tableaux, des matrices et chaînes de caractères. Dans ce TP, nous abordons ces structures en langage C.

1 Les tableaux

Un tableau (un vecteur) est une variable permettant de stocker plusieurs valeurs de même type dans une même structure. En langage C, la création des tableaux peut être statique ou dynamique (la création dynamique ne sera pas abordée dans ce TP). La taille d'un tableau statique est invariable dans le temps ; c'est pour cela qu'il faut créer dès le début un tableau avec la taille nécessaire. À la création d'un tableau d'une taille N , le système réserve N cases où chaque case est de taille adéquate au type du tableau. Chaque élément du tableau est stocké dans une case du tableau.

1.1 Déclaration

```
type_du_tableau nom_du_tableau [taille_du_tableau] ;
```

Exemple :

```
int tab[100] ; // déclaration d'un tableau de 100 entiers.
```

```
int max=10 ; float t[max] ; // déclaration d'un tableau de max réels.
```

1.2 Initialisation

```
type_du_tableau nom_du_tableau [taille_du_tableau] = {val1, val2, val3, ..., valn} ;  
ou :  
type_du_tableau nom_du_tableau [] = {val1, val2, val3, ..., valn} ;
```

Remarques :

- Dans la première initialisation **n** doit être inférieur ou égal à **taille_du_tableau**.

- Dans le 2^e cas, la taille du tableau est égale à **n**.

Exemple :

```
int tab[5]={1,2,3} ;//déclaration d'un tableau de taille 5, initialisé par 3 éléments.
```

```
int tab[] = {1,2,3} ;//déclaration d'un tableau de 3 éléments (la taille du tableau est égale à 3).
```

1.3 Manipulation des éléments

L'accès à une case d'un tableau se fait via un indice qui prend des valeurs de 0 à N-1 (N est la taille du tableau). Le premier élément a 0 comme indice et le dernier élément a (N-1) comme indice. Une case du tableau est considérée comme une variable : toutes les opérations applicables sur les variables sont applicables sur les cases du tableau.

```
nom_du_tableau[indice]
```

Exemples :

```
soit : int tab[10]= {1,2,3} ;
```

- `tab[0] += tab[1]+tab[2] ; // tab[0]=1+2+3=6.`
- `printf("%d", tab[2]) ; // affichage de la valeur 2.`
- `scanf("%d", &tab[0]) ; // lecture de tab[0].`

2 Les matrices

En langage C, une matrice est un tableau de tableau. Une matrice est un tableau à 2 dimensions qui est constitué de lignes et de colonnes. L'intersection d'une ligne avec une colonne est une case où un élément de la matrice peut être stocké.

2.1 Déclaration

```
type_de_la_matrice nom_de_la_matrice [nombre_de_lignes][nombre_de_colonnes] ;
```

Exemples :

```
int mat[2][3] ; // création de la matrice mat de 2 lignes et 3 colonnes (de 2*3=6 éléments).
```

```
int mat[4][4] ; // création d'une matrice carrée de 16 éléments.
```

2.2 Initialisation

```
type_de_la_matrice nom_de_la_matrice [nombre_de_lignes][nombre_de_colonnes] =  
    {{val_11, val_12, ...,val_1m}, ..., {val_n1, val_n2, ..., val_nm}} ;
```

ou :
 type_de_la_matrice nom_de_la_matrice [][] =
 {{val_11, val_12, ..., val_1m}, ..., {val_n1, val_n2, ..., val_nm}} ;

Exemples :

```
int mat[][]={{1,2},{4,8}} ;// ici la taille de mat est 2*2
```

```
int mat[3][4]={{1,2},{4,8}} ;// ici la taille de mat est 3*4
```

2.3 Manipulation des éléments

Soit une matrice de N lignes et M colonnes :

- la taille de la matrice est égale à $N \times M$,
- l'indice de la première ligne ainsi que la première colonne est 0,
- l'indice de la dernière ligne ainsi que la dernière ligne est $N-1$ et $M-1$ respectivement.

L'accès à un élément (une case) se fait comme suit :

nom_matrice[indice_ligne][indice_colonne]

Une case d'une matrice est considérée comme une variable : toutes les opérations applicables sur les variables sont applicables sur les cases d'une matrice.

Exemples :

```
int mat[][]={{1,0,0},{0,1,0},{0,0,1}} ; // déclaration.
```

```
printf("%d", mat[0][0]) ;
```

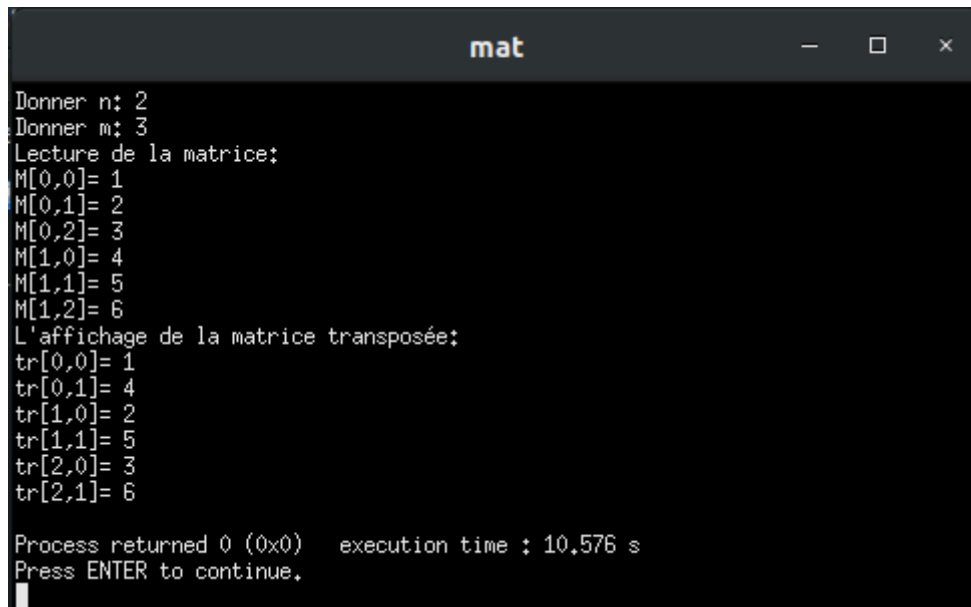
```
scanf("%d",& mat[1][1]) ;
```

```
mat[2][2]=mat[1][1]+mat[0][0] ;
```

2.4 Exemple : Calcul de transposée d'une matrice :

```
main.c ✕
3
4 int main()
5 {
6     int mat[30][40], tr[40][30], n, m, i, j;
7     // Faites le controle sur n et m:
8     printf("Donner n: ");
9     scanf("%d", &n);
10    printf("Donner m: ");
11    scanf("%d", &m);
12
13    printf("Lecture de la matrice:\n");
14    for(i=0; i<n; i++){
15        for(j=0; j<m; j++){
16            printf("M[%d,%d]= ", i, j);
17            scanf("%d", &mat[i][j]);
18        }
19    }
20
21    for(i=0; i<n; i++){
22        for(j=0; j<m; j++){
23            tr[j][i]=mat[i][j];
24        }
25    }
26
27    printf("L'affichage de la matrice transposée:\n");
28    for(i=0; i<m; i++){
29        for(j=0; j<n; j++){
30            printf("tr[%d,%d]= %d\n", i, j, tr[i][j]);
31        }
32    }
33    return 0;
34 }
```

Exécution



```

mat
Donner n: 2
Donner m: 3
Lecture de la matrice:
M[0,0]= 1
M[0,1]= 2
M[0,2]= 3
M[1,0]= 4
M[1,1]= 5
M[1,2]= 6
L'affichage de la matrice transposée:
tr[0,0]= 1
tr[0,1]= 4
tr[1,0]= 2
tr[1,1]= 5
tr[2,0]= 3
tr[2,1]= 6

Process returned 0 (0x0)   execution time : 10.576 s
Press ENTER to continue.

```

3 Les chaînes de caractères

En langage C, une chaîne de caractères est une séquence de caractères qui se termine par un caractère nul : `'\0'`. Cette séquence est représentée sous forme d'un tableau de caractères avec le dernier élément le caractère `'\0'`.

3.1 Déclaration

```
char nom_de_la_chaine_de_caracteres[taille_de_la_chaine_de_caracteres];
```

Une chaîne de caractère d'une taille N comprend (N-1) caractères suivi de `'\0'`.

Exemple : `char ch[20];` //déclaration d'une chaîne de caractères de taille 20 (19 caractères au maximum + `'\0'`).

3.2 Initialisation

```
char nom_de_la_chaine_de_caractères[] = "la chaine de caractères";
```

Exemples :

`char c[] = "abcd";` //la taille de c est égale à 5.

`char c[10] = "abdc";` //la taille de c est égale à 10.

`char c[] = {'a', 'b', 'c', 'd', '\0'};` //la taille de c est égale à 5.

```
char c[10]={ 'a','b','c','d','\0' } ; //la taille de c est égale à 10.
```

3.3 Manipulation des éléments

Comme les chaînes sont des tableaux, elles sont manipulées de la même façon que les tableaux, sauf dans la lecture et l'écriture. La lecture et l'écriture d'une chaîne de caractères se font comme suit :

Lecture :

Pour lire une chaîne de caractères, on utilise le spécificateur de format '%s'. La fonction 'scanf' lit les caractères jusqu'à elle rencontre un caractère blanc.

```
char c[20] ;  
scanf("%s", c) ; //sans '&'.
```

Pour lire une ligne on utilise la fonction 'fgets' comme suit :

```
char c[20] ;  
fgets(c, sizeof(c), stdin) ; //stdin est le fichier du clavier.
```

La fonction '**fgets**' lit depuis le fichier '**stdin**' une séquence de caractère de taille (**sizeof(c)-1**) jusqu'à la rencontre d'un saut de ligne '\n' ou la fin du fichier.

Écriture

```
char c[20] = "Hello world!" ;  
printf("%s\n", c) ;
```

Exemples :

```
char c[20] = "hello world ?" ;  
c[0]='H' ; // "Hello world ?"  
c[20]='\0' ; // "Hello world !"
```

3.4 Bibliothèque 'string.h' :

'string.h' est une bibliothèque implémentée en langage C qui comporte des fonctions prédéfinies permettant de manipuler les chaînes de caractères à savoir la concaténation, la comparaison, etc.

Exemple : la fonction `strlen(<chaîne_de_caractères>)` renvoie la longueur (la taille) de `<chaîne_de_caractères>` :

```
char c[20] = "hello world ?" ;  
int l = strlen(c) ; // la variable l est égale à 13 (la taille de la chaîne de caractères c.
```

Exemple 1 :

```
main.c ✕
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      char c[500];
7      int i;
8      printf("Donner un mot c: ");
9      fgets(c, sizeof(c), stdin);
10     i=0;
11     while(c[i] != '\0'){
12         if(c[i]>='a' && c[i]<='z'){
13             c[i]= c[i]-'a'+'A';
14         }
15         i++;
16     }
17     printf("c est devenu: %s\n",c);
18     return 0;
19 }
20
```

Exécution :

```
tableaux
Donner un mot c: Hello world!
c est devenu: HELLO WORLD!

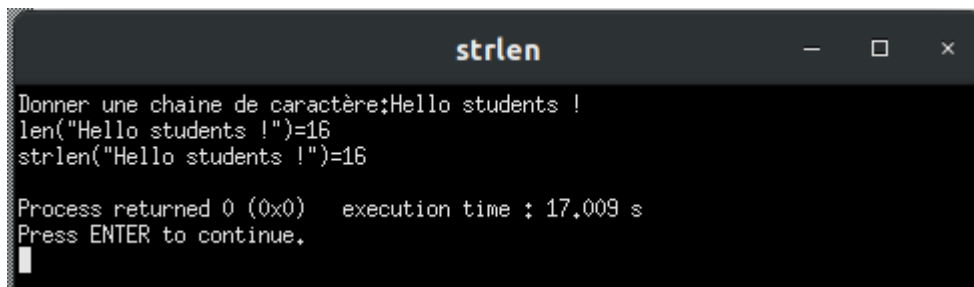
Process returned 0 (0x0)   execution time : 4.999 s
Press ENTER to continue.

```

Exemple 2 : Longueur d'une ligne :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main()
6  {
7      char ch[200], i;
8      printf("Donner une chaine de caractère:");
9      gets(ch);
10
11     for(i=0; ch[i] != '\0'; i++);
12
13     printf("len(\"%s\")=%d\n",ch, i);
14     printf("strlen(\"%s\")=%d\n", ch, strlen(ch));
15
16     return 0;
17 }
18
```

Exécution



```
strlen
Donner une chaine de caractère:Hello students !
len("Hello students !")=16
strlen("Hello students !")=16

Process returned 0 (0x0)   execution time : 17.009 s
Press ENTER to continue.
```