Langage C - Tableaux

Tableaux

Passage de tableaux en paramètre d'une fonction

Chaînes de caractères

Manipulation de chaînes de caractères

Tableaux multidimensionnels

Tableaux de chaînes de caractères

Tableaux (1)

- **Un tableau est une suite ordonnée de valeurs du même type**. On peut accéder à un élément du tableau grâce à une numérotation de ceux-ci.

En langage C, dans un tableau de n éléments, les éléments sont numérotés de 0 à n-1. Ce numéro est appelé **indice**.

- **Déclaration**

type nom_tableau[taille_tableau]; définit un tableau nom_tableau de taille taille_tableau contenant des éléments de type type.

```
int tableau_entier[3]; // déclare un tableau de trois entiers. Les indices vont de 0 à 2.
```

Le compilateur, lorsqu'il voit cette déclaration, fait l'allocation de mémoire contigüe de douze fois la taille d'un entier int.

En C ANSI, la taille du tableau doit être une constante. Pour faire des tableaux de longueur variable, il faut recourir à l'allocation dynamique.

- **Utilisation**

On peut manipuler les éléments d'un tableau un par un grâce à l'opérateur [indice] où indice est le numéro de la case.

```
int a=7;                                  // tableau_entier    ?  ?    ?
tableau_entier[0]=24;                       //                24  ?    ?
tableau_entier[1]=a/2;                    //                24  3    ?
tableau_entier[2]=tableau_entier[0];      //                24  3   24
```

Un élément d'un tableau peut être utilisé comme une variable standard, c'est-à-dire en lecture mais aussi en écriture.

Tableaux (2)

➢ Initialisation

Pour initialiser un tableau, c'est-à-dire donner une valeur initiale à chacun de ses éléments, on utilise les accolades.

```
int nb_jours_dans_mois[]={31,28,31,30,31,30,31,31,30,31,30,31};
```

Dans ce cas, il n'y a pas besoin d'indiquer la taille du tableau. Elle est automatiquement calculée par le compilateur.

➢ Utilisation dans une boucle

On utilise souvent les tableaux dans des boucles. Un compteur sert alors à adresser les éléments successifs du tableau.

```
#define NB_ELEMENTS 15
…
int tableau_entier[NB_ELEMENTS];
int i;
for(i=0;i<NB_ELEMENTS;i++)
        tableau_entier[i]=…;          // initialisation des valeurs du tableau
```

Passage de tableaux en paramètre d'une fonction

- **Un pointeur est une adresse mémoire**.

Si l'on compare la mémoire centrale de l'ordinateur à un énorme tableau, un pointeur sera simplement un indice dans ce tableau.

Qu'une variable représente un pointeur ou un tableau, l'utilisation sera exactement la même pour aller chercher le ième élément du tableau : on utilise l'opérateur [].

```
int tableau_entier[3];
int* p_entier=tableau_entier;
                    tableau_entier[i] ⇔ p_entier[i]
```

- **Passer un tableau en paramètre à une fonction est équivalent à passer l'adresse de son premier élément.**

```
int ma_fonction(int tab_int[]); ⇔ int ma_fonction(int* tab_int);
```

Un tableau est toujours passé par adresse. Cela signifie que **si son contenu est modifié par la fonction appelée, son contenu sera aussi modifié dans le contexte appelant**.

En général, on indique à la fonction **la taille du tableau grâce à un deuxième paramètre**.

```
void tri_tableau(int*,int);

void main(void)
{
    int tableau_entier[3];
    tri_tableau(tableau_entier,3);
}
```

Chaînes de caractères (1)

- **Déclaration**

En C, une chaîne de caractères est un tableau de caractères (char).

char tableau_caractere[50]; // déclare un tableau de cinquante caractères

Les **constantes chaînes de caractères** sont construites grâce aux guillemets doubles.

"Hello !", "Ligne 1\nLigne 2\nLigne 3\n", "IUT d'informatique" sont des chaînes de caractères constantes, c'est-à-dire qu'on ne peut pas les modifier.

- **Codage des chaînes**

En C, une chaîne de caractères est un tableau de caractères qui se **termine par le caractère '\0'**.

Il faut donc toujours **n+1 caractères pour coder une chaîne de longueur n**.

Un tableau peut avoir une taille plus grande que la taille réelle de la chaîne de caractères qu'il contient. En effet, la fin de la chaîne est indiquée par '\0' qui peut donc se trouver à différents positions dans le tableau.

Chaînes de caractères (2)

- **Initialisation**

Les constantes chaînes de caractères peuvent servir à l'initialisation d'une variable de type chaîne de caractères non constante.

```
char tableau_caractere[]="Hello !";
```

Dans ce cas, la taille du tableau est automatiquement calculée et chaque caractère est recopié dans les cases du tableau.

```
// Hello ! ↔ 7 caractères donc tableau_caractere est un tableau de 8 caractères
// La chaîne "Hello !" sera donc codée par le tableau suivant :
// tableau_caractere   'H'   'e'   'l'   'l'   'o'   ' '   '!'   '\0'
// indice               0     1     2     3     4     5     6     7
```

Cette initialisation n'est possible que lors de la déclaration.

On peut aussi faire une initialisation de chaîne en indiquant la taille maximum de la chaîne.

```
char tableau_caractere[50]="Initialisation";
/* Ceci crée une chaîne de caractères qui contiendra au maximum 49 caractères (on enlève un pour '\0') mais qui est initialisée avec une chaîne de 14+1 caractères. */
```

Manipulation de chaînes de caractères (1)

- La **bibliothèque standard <string.h>** offre une multitude d'outils pour manipuler et analyser les chaînes de caractères.

- **Longueur d'une chaîne**

La fonction **int strlen** permet de connaître la longueur d'une chaîne.

```
#include <string.h>
// …
char tableau_caractere[]="Ma chaine";
printf("%s a pour longueur %d.",tableau_caractere,strlen(tableau_caractere));
// affiche   Ma Chaine a pour longueur 9.
```

Le caractère '\0' n'est pas compté par la fonction strlen.

Manipulation de chaînes de caractères (2)

- **Comparaison de chaînes**

La fonction **int strcmp(const char* s1,const char* s2)** permet de savoir si :

- deux chaînes sont égales c'est-à-dire identiques au caractère près ;
- une chaîne est inférieure à une autre au sens de l'ordre alphabétique (ou plutôt A.S.C.I.I.).

Deux chaînes sont données en paramètre et la fonction fournit un résultat entier :

- 0 si les deux chaînes sont égales ;
- positif si s1 est plus grande que s2 ;
- négatif si s1 est plus petite que s2.

```
s1          s2                  strcmp(s1,s2)
"aaaa"      "Aaaa"              32
                    // dans la table A.S.C.I.I., les majuscules sont avant les minuscules
"aaaa"      "aaab"              -1
"aaaa"      "aaaaaaa"           -97
```

Manipulation de chaînes de caractères (3)

Si on veut utiliser la fonction strcmp dans une comparaison de chaînes pour savoir si elles sont identiques, il faut faire une négation car la fonction renvoie 0 qui correspond au booléen faux.

```
if(strcmp(tableau_caractere_1,tableau_caractere_2)==0)
// équivalent à   if(!strcmp(tableau_caractere_1,tableau_caractere_2))
// les deux chaînes sont identiques
{
     // …
}
```

- **Copie de chaînes**

La fonction **char* strcpy(char* dest,const char* src)** permet de copier la chaîne src dans la chaîne dest dont l'ancien contenu est perdu. La fin de chaîne (caractère '\0') est elle aussi recopiée.

La destination doit être un tableau de taille suffisante pour pouvoir contenir la chaîne recopiée. Si ce n'est pas le cas, il risque d'y avoir une erreur de protection de la mémoire.

Tableaux multidimensionnels (1)

En C, on peut construire des tableaux à plusieurs dimensions.

Un **tableau à deux dimensions** correspond à un tableau avec des lignes et des colonnes.

A **trois dimensions**, on peut voir cela comme des tableaux à deux dimensions empilés les uns sur les autres.

Au delà de trois, cela devient plus difficile à imaginer et vous aurez rarement à en manipuler.

➢ Déclaration

La déclaration se fait en mettant plusieurs fois des crochets.

```
float matrice_reel[3][4];
/* déclare un tableau de réels à deux dimensions dont les indices vont de 0 à 2 ou 3. Il ne faut pas
    confondre la dimension du tableau (ici 2) avec la taille (3 fois 4). */
```

➢ Utilisation

Il faut mettre plusieurs fois les crochets avec les indices correspondants.

```
matrice_reel[0][1]=1.6;
matrice_reel[2][2]= matrice_reel[1][3] + 7;
// La notation matrice_reel[0,1] est interdite.
```

Tableaux multidimensionnels (2)

➢ Initialisation

Il y a deux manières équivalentes d'initialiser un tableau multidimensionnel.

```
int matrice_entier_1[3][4]={{0,1,2,3},{4,5,6,7},{8,9,10,11}};
int matrice_entier_2[3][4]={0,1,2,3,4,5,6,7,8,9,10,11};
```

➢ Transformer en un tableau à une dimension

Il est possible de transformer un tableau multi-dimensionnel en tableau à une dimension en mettant toutes les lignes sur une seule, les éléments étant les uns à la suite des autres.

```
float m[3][4];                    m[0][0]  m[0][1]  m[0][2]  m[0][3]
                                  m[1][0]  m[1][1]  m[1][2]  m[1][3]
                                  m[2][0]  m[2][1]  m[2][2]  m[2][3]
float M[12];          M[0] M[1] M[2] … M[9] M[10] M[11]
// Une matrice 3 fois 4 devient un tableau de 12 valeurs et on a :
// m[i][j] => M[i*4+j]
// M[k] => m[k/4][k%4]
```

Cela est généralisable pour toutes les dimensions. Cela permet de faire de l'allocation dynamique de zones contigües plus facilement et augmente les performances.

Tableaux de chaînes de caractères

Il est souvent utile de fabriquer des tableaux dont les éléments sont des chaînes de caractères.

```
char* intitule_mois[]={"janvier","fevrier","mars","avril","mai","juin","juillet","aout",
                                                    "septembre", "octobre","novembre","decembre"};

// intitule_mois[0] correspond à la chaîne de caractères "janvier"

for(i=0;i<12;i++)
{                                                   // facultative car une seule instruction
      switch(intitule_mois[i][0])
      {
          case 'a':
          case 'o': printf("Le mois d'%s comporte %d jours.",intitule_mois[i],
                                                                    nb_jours_dans_mois[i]);
                    break;
          default: printf("Le mois de %s comporte %d jours.",intitule_mois[i],
                                                                    nb_jours_dans_mois[i]);
      }
}                                                   // facultative
```