Langage C

Les tableaux

L2 Mathématique et Informatique

Université de Marne-la-Vallée

Tableau en C

Un tableau C est constitué d'un nombre fixé d'éléments du même type, accessibles par un indice. La déclaration d'un tableau

```
1 typeElement nomTableau[taille];
```

- `nomTableau` est le nom du tableau;
- `taille` est un entier positif. En C `ansi` taille doit être une constante définie à la compilation;
- la déclaration réserve une zone permettant de ranger consécutivement `taille` entiers de type `typeElement`
- les indices sont des entiers et commencent avec 0
- les indices valides vont de 0 à `taille` −1
- `t[i]` désigne l'élément à l'indice `i` du tableau `t`

Taille fixée à la compilation

Ne pas utiliser de valeurs numérique pour la taille d'un tableau.

```
1  /* Remplir un tableau avec les 5 premiers multiples de 5
2  int truc[5];
3  for(i=0;i<5;i++)
4      truc[i]=5*i;
5  for(i=0;i<5;i++)
6      printf("%d ",truc[i]);
```

Comment faire si on veut maintenant les 20 premiers multiples de 5.
On ne peut utiliser sans risque la substitution de l'éditeur de texte (*changer partout 5 en 20*)

Définition de constante

- Utiliser une constante symbolique

```
1 #define TAILLE 42
```

le préprocesseur remplace avant compilation toute apparition de la première suite de lettres par la deuxième (aucune analyse). L'utilisation de majuscule permet de reconnaitre dans le code une directive pour le préprocesseur (très utile en cas d'erreur dans la définition).

- Utiliser une variable *constante*
  On spécifie qu'une variable désigne une constante en préfixant le type par `const` lors de la déclaration.

```
1 const int taille=42;
```

Une tentative d'affectation directe (`taille=12;`) est une erreur detectée à la compilation
Une écriture indirecte (via l'adresse)(`scanf("%d",&taille);` ne provoque qu'un warning, et des erreurs à l'exécution.
Pour éviter toute problème déclarer la taille comme globale.

Taille définie à l'exécution

On peut définir des tableaux à l'exécution. La déclaration doit être effectuée après avoir déterminé la taille. À utiliser avec concentration.

```
1     do{
2         printf("entrez la taille >0");
3         scanf("%d",&taille);
4       } while(taille <=0);
5
6     int t[taille];
7     for (i=0;i<taille;i++)
8         t[i]=5*i;
```

Lorsque la taille est connue à la compilation, on peut préciser les valeurs des éléments à la déclaration : on donne entre accolades les valeurs consécutives séparées par des virgules.
Si la totalité des valeurs n'est pas donnée, les éléments sont initialisés avec des 0.

```
1  #define TAILLE 10
2  int i,t[TAILLE];
3  int tt[TAILLE]={0,1,2,3,4,5};
4  ...
5   }
```

`t` est un tableau de 10 `int` au contenu non précisé (ce qu'il y a dans la zone reservée).
`tt` est une tableau de 10 entiers. les 6 premiers contiennent les entiers de 0 à 5, les 4 suivants contiennent 0.

Tableaux partiellement remplis

Dans la pratique, on sait rarement à l'avance le nombre d'éléments à manipuler. Suivant les cas

- On déclare des tableaux de taille réelle TAILLE_MAX (constante) que l'on remplit partiellement.
- On alloue dynamiquement la place juste nécessaire (malloc)

Il faut savoir combien d'éléments sont à prendre en compte

- on mémorise la taille effective (le nombre d'éléments à prendre en compte) dans une variable. Le mieux est d' associer le tableau et sa taille effective dans une `structure`.
- on utilise un marqueur de fin (élément n'appartenant pas à l'ensemble des valeurs valides) après le dernier élément valide. Ceci n'est pas toujours possible.

Manipulation

- un tableau c'est l'adresse de son premier élément, le compilateur interprétant `t[i]` comme le `i` $^{\text{ième}}$ élément du type après l'adresse `t`.
- si on transmet un tableau à une fonction, les éléments sont transmis par adresse.
- un tableau ne connait pas sa taille
- c'est au programmeur de vérifier la validité des indices utilisés
- Pas d'opérations globales sur les tableaux! En particulier pas d'affectations ou de copie directe. Il faut utiliser des fonctions qui parcourent le tableau cases par cases.

Fonction et tableau

- comme un paramètre d'une fonction, on déclare le type tableau avec les crochets;
- à l'appel, on précise uniquement le nom du tableau. Pour écrire des fonctions facilement réutilisables, on transmet la taille effective;

```
1  void afficheTableau(int tab[], int taille){
2      int i;
3      for(i=0;i<taille;i++)
4          printf("%d ",tab[i]);
5      printf("\n");
6  }
7  void lireTableau(int tab[], int taille){
8      int i;
9      for(i=0;i<taille;i++)
10         scanf("%d",&tab[i]);
11 }
```

Tableaux à plusieurs dimensions

- `truc tab[N];` est un tableau à une dimension. `tab` est une tableau de `N` éléments de type truc.
  `truc ... [N]` correspond au type *tableau de `N` truc*
- `truc tab[M][N]` tab est donc un tableau de M tableaux de N truc
- un tableau de dimension *n* est un tableau dont les éléments sont des tableaux de dimension $n - 1$

```
1  int mat[2][3];
2   int i,j,val=0;
3   for(i=0;i<2;i++)
4     for(j=0;j<3;j++){
5       mat[i][j]=val;
6       val++;
7     }
8  for(i=0;i<2;i++){
9    for(j=0;j<3;j++)
10     printf("mat[%d][%d]=%d ",i,j,mat[i][j]);
11   printf("\n");
12  }
```

```
mat[0][0]=0 mat[0][1]=1 mat[0][2]=2
mat[1][0]=3 mat[1][1]=4 mat[1][2]=5
```

Tableaux à plusieurs dimensions et appel de fonctions

Lors de la déclaration du paramètre, il faut préciser la taille dans toutes les dimensions sauf la première. Cela permet au compilateur de calculer l'adresse où trouver chaque élément. Ces tailles peuvent être des constantes (C ansi):

```
1  void afficheMat(int t[][M],int li,int col){
2  /* col <=M*/
3     int i,j;
4     for(i=0;i<li;i++){
5        for(j=0;j<col;j++)
6           printf("mat[%d][%d]=%d ",i,j,t[i][j]);
7        printf("\n");
8     }
9  }
```

tailles définies à l'exécution

Le compilateur accepte maintenant dans la déclaration de fonction que la taille soit un paramètre de la fonction, la taille doit alors apparaître avant le tableau dans la liste des paramètres

```
1 void afficheMatrice(int li, int col, int mat[][col]);
```

LES PIEGES

- Renvoyer un tableau
  Un tableau étant une adresse, on pourrait être tenter de renvoyer un tableau déclaré localement dans une fonction.
  Le tableau est alloué sur la pile pendant l'exécution de la fonction.
  Après l'exécution l'emplacement du tableau sur la pile est au dessus du sommet et est donc libre. Il est réutilisé des appels de fonctions suivants. Le contenu du tableau varie au fur et à mesure de l'exécution.

- Ordre de parcours de tableau à plusieurs dimensions
  Les tableaux sont rangés par ligne, colonnes, ... Un parcours dans un autre ordre dégradera fortement les performances:

```
1  #define N 20000
2  char T[N][N];
3  int main(void){
4  /*par ligne*/
5   for (i=0 ; i<N ; i++)
6       for (j=0 ; j<N ; j++)
7         T[i][j] = 1;
8  /*dure 1 seconde*/
9  /* par colonnes*/
10  for (j=0 ; j<N ; j++)
11      for (i=0 ; i<N ; i++)
12        T[i][j] = 1;
13 /*dure 3 secondes */
14 return 0;
15 }
```

- les “gros” tableaux ne peuvent être alloués sur la pile. L’essai précédent entraîne une erreur de segmentation si T est déclaré en local.