

CM4 : Tableaux et pointeurs

ISia2A3 - Algorithmique et programmation structurée

Thibault Liétard

2026-2027

Tableaux

Généralité

Définition
Structure de données permettant de mémoriser un ensemble indexé et de taille fixe d'informations de même type

Dimensions

- **Vecteur** : dimension 1

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

- **Matrice** : dimension 2 ou plus

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

Accès aux informations

index d'un vecteur

	1	2	3	4	5	6	7	8	9	10	11	12
v	10	0	4	5	6	5	1	10	6	10	0	7

v[1]=10, v[2]=0,...

index d'une matrice

		1	2	3	4	5	6	7	8	9	10	11	12
t	1	4	0	3	1	8	4	8	6	0	8	10	6
	2	8	8	1	7	9	6	6	10	2	7	6	2

t[1][1]=4, t[2][3]=1,... t[*ligne*][*colonne*]

Attention

index en C
En C, les index commencent à 0 car il s'agit d'un décalage en mémoire.

Déclaration d'un tableau

Algorithmique
<nom> : vecteur[<taille>] de <type>

C
<type> <nom>[<taille>];

- *int t[5];*
- *int t[]={1,2,3,4,5};*
- *int t[10] = {0}; /*alloue un tableau de 10 cases valant 0*/*

Attention

Pas de vérification de taille
À la compilation, *v[100]* ne déclenche aucune erreur même si *v* est de taille 10.

En pseudo-code comme en C, on fera toujours un test sur la taille.

Notation {} uniquement à la déclaration
Il n'existe pas de valeur ou de type "tableau".

Attention 2

Cases vides
Les cases d'un vecteur sont "vides" à l'initialisation, ce qui veut dire :

- Qu'on ne peut pas faire de test dessus en pseudo-code car rien ne permet de tester le vide en booléen
- Qu'on ne peut pas faire de test dessus en C car elles ne sont pas réellement vides (état aléatoire de la mémoire)

Cas des chaînes de caractères

Pseudo-code
Une chaîne de caractère est un tableau de caractères contenant le texte de la chaîne

en C
Une chaîne de caractère est un tableau de caractères contenant le texte de la chaîne et terminé par le caractère de fin de chaîne \0

Déclaration en C

Tableau
Une chaîne de caractère se déclare comme un tableau de type *char*.
C'est le seul cas où on peut affecter une valeur à un tableau.

Exemple
char s[] = "hello";

h	e	l	l	o	\0
0	1	2	3	4	5

"hello" est un mot de 5 lettres mais le tableau fait 6 cases car s'ajoute le caractère de fin de chaîne.

Attention

Différence
char s1[10] = ”hello”;
char s2[] = ”hello”;

- Dans le premier cas, on précise la taille du tableau :

h	e	l	l	o	\0				
0	1	2	3	4	5	6	7	8	9

- Dans le deuxième cas, on précise la taille du tableau est dicté par la taille de la chaîne :

h	e	l	l	o	\0
0	1	2	3	4	5

On pourrait également déclarer *char* s3 = ”hello”;*

Affichage et lecture

Utilisation du format %s

```
char s[20];
scanf("%s", s);
printf("chaine : %s \n", s);
```

Notes

- Pas de [] dans le passage en paramètre du *printf*
- Pas besoin de & dans le *scanf* car *s* est déjà une adresse, celle de début du vecteur

Tableau et allocation mémoire

Exemple
int t[5];

Alloue 20 octets en mémoire (5x4o).

Généralité
<type> <nom>[<taille>]

Alloue *<taille>* x *sizeof(<type>)* en mémoire, de façon contingüe chaque fois que cela est possible.

Pointeurs

Fonctionnement en mémoire

Exemple
int t[5];

- Déclaration implicite de *t[0]*, *t[1]*, *t[2]*, *t[3]*, *t[4]*
- *t* désigne l'adresse du premier élément du tableau

Principe
int t[5];

- *t* est une adresse typée correspondant à l'adresse de *t[0]*
- *t* désigne l'adresse du premier élément du tableau

Notion de mémoire

Principe
Chaque variable en C est stockée en mémoire est possède **une adresse** (gérée par le système) et **une taille** (dépendante de son type) dans la mémoire.

Pointeur
Un **pointeur** est une variable qui contient une adresse mémoire et son type associé. Un pointeur est toujours typé, et le ”type” pointeur associé à un type s’obtient à l’aide de l’opérateur *.

En C

int i = 0;
int p = &i;*

- ***&i*** désigne l'adresse de la variable *i*, sous la forme d'un code hexadécimal et d'un type (par exemple *<2A15, int>*)
- ***int**** permet de déclarer un pointeur sur un *int*, c'est à dire une variable qui contient l'adresse d'un *int*
- ***p*** est une variable de type "pointeur sur un int"

variable	type	adresse	valeur
i	int	<2A15>	0
p	int *	<2A19>	<2A15>

Pointeurs sur les tableau 1/3

Exemple
int t[5];

- **t** et *&t[0] désignent la même adresse*
- **t* donne la valeur de *t[0]*
- *t+i* est un pointeur sur la case *i*
- *t[i]* revient à faire **(t+i)*

Pointeurs sur les tableau 2/3

Attention
t est une variable de type *int**; en conséquence :

- le contenu de *t* peut être modifié avec *t*[*i*]
- mais la valeur de *t* ne peut pas être modifiée quand on déclare un tableau

Pointeurs sur les tableau 3/3

Arithmétique

Si *t* est un tableau d'entiers, alors :

- *t* est aussi un pointeur valant l'adresse du premier élément
- *t+i* est un pointeur sur la case *i*
- *(t+i) = t + i*sizeof(int)*

Généralités

- $t = < t_{@}$, *<type>* $>$
- *t+i* = $< (t + i)_{@}$, *<type>* $>$
- $(t + i)_{@} = t_{@}$ + *i*sizeof(<type>)*

Pointeur et passage de paramètres 1/3

Problème
En C, quand une fonction s'exécute :

- Les paramètres sont passés par copie. La fonctionne ne travaille pas directement sur les variables passées en paramètre, mais sur une **copie locale**
- Les variables locales à cette fonction n'existent pas en dehors du scope de cette fonction

Conclusion : si on veut modifier la valeur d'un paramètre de façon pérenne, il faut fonctionner autrement

Pointeur et passage de paramètres 2/3

Solution
Passer par des **pointeurs**

- la fonction reçoit une copie de la valeur du pointeur comme paramètre effectif
- cette copie de valeur contient l'adresse de la variable à modifier
- en modifiant la valeur à cette adresse, on modifie une valeur accessible par le programme appelant

Cas pratique
Retourner la somme et la différence de deux entiers

- pseudo-code : paramètre formels *min* et *max* entiers de type Résultat
- C : paramètre *min* et *max* de type *int**

Pointeur et passage de paramètres 3/3

Note
Exemple d'une fonction avec deux paramètres à modifier car sinon on serait passé par le *return*...

Exception
... sauf si le paramètre en question est un tableau car on en peut pas passer de tableau en paramètre en C.

Tableau en paramètre

Principe
On passe un tableau *t* de taille *N* et de type *<type>* en paramètre de deux façons :

- soit par l'adresse de début du tableau avec un pointeur sur le type : *<type>* t*
- soit en le précisant comme tableau dans les paramètres formels : *<type> t[N]*

On peut éventuellement ne pas préciser la taille pour la deuxième méthode.

Exemple

```c
#define N 10

void init(int t[N]){
    for(int i=0; i < N; i++){
        t[i] = 0;
    }
}

int main(){
    int tab[N];

    init(tab);
    for(int i=0; i < N; i++){
        printf("%d\n", tab[i]);
    }

    return 0;
}
```

Remarques

Différence appelant/appelé

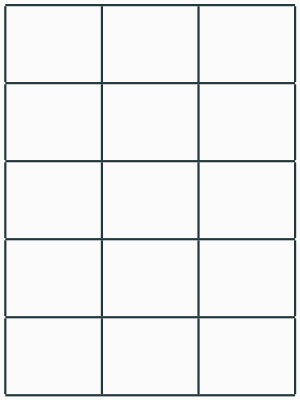
- Pas d'allocation mémoire de vecteurs dans le contexte automatique de l'appelé (paramètres pointeur, donc copie d'adresse)
- Chez l'appelant, dans l'appel à la fonction, *t* est une adresse typée, donc une valeur
- Chez l'appelé, *t* est un pointeur, donc une variable

Matrices en C

Généralités

Principe
Une matrice est un tableau de tableau. On déclare une matrice avec *<type> <nom>[<nb_lignes>][<nb_colonnes>]*

Exemple
int m[5][3]



m[i] désigne la ligne d'indice *i* de la matrice. C'est un vecteur de 3 entiers.

m[i][j] désigne la j^{eme} cases de la i^{eme} ligne.

Fonctionnement en mémoire

Déclaration

- *m* est un vecteur de 5 éléments
- chacun de ces élément est un vecteur de 3 entiers
- *m* est donc un pointeur : <$m_{@}$, ”vecteur de 3 entiers”>
- *m[i]* est donc l’adresse typée du premier élément de la ligne *i* : <$(m[i])_{@}$, *int*>

Contiguïté

- $(m[i][j+1])_{@} = (m[i][j])_{@} + 1 * sizeof(int)$
- $(m[i+1][j])_{@} = (m[i][j])_{@} + 3 * sizeof(int)$

Allocation mémoire

<type> m[nb_l][nb_c]
Les tableau correspondant aux lignes étant stockées de façon contiguë dans la mémoire, un élément est en fait retrouvé en mémoire par la formule suivante :

*m[i][j] = *(m + (i * nb_c + j) * sizeof(<type>))*

Attention

Passage de paramètre en fonction

- Pour retrouver un élément en mémoire, il faut connaître le nombre de colonnes
- Pour passer une matrice en paramètre, il faut préciser au moins *nb_c*

Exemple

void initMat(int l, int m[ ][3])

Tableaux multidimensionnels

De manière générale
void f(<T> t[ ][d2][d3]...[dn], int d1) {

...

} Seule la première dimension est libre, les autres doivent être précisées.

Cas des chaînes de caractères

Différences

char mjours[7][9] = {”lundi”, ”mardi”, ”mercredi”, ”jeudi”, ”vendredi”, ”samedi”, ”dimanche” };

et

char mjours[7] = {”lundi”, ”mardi”, ”mercredi”, ”jeudi”, ”vendredi”, ”samedi”, ”dimanche” };*

Explications

Similarité

Dans les deux cas, *mjours* est un *char***

Différences

- Premier cas : *mjours* est une matrice de *char* de taille 7x9. Les éléments de *mjours* sont des tableaux de *char* de taille 9.
- Deuxième cas : *mjours* est un tableau de *char** de taille 7. Les éléments de *mjours* sont des adresse typées de caractères (*char**), dont chacune est l'adresse de début d'une chaîne de caractères de taille variable.

Paramètres du *main* 1/2

Principe
Possibilité de passer des paramètres au programme principal (*main*) lors de l'appel

Syntaxe
```
int main (int argc, char *argv[ ]) {
    ...
}
```

Détails

- *argc* : nombre de paramètres
- *argv* : tableau contenant les paramètres

Paramètres du *main* 2/2

Explications

- Permet de passer des paramètres au programme quand on le lance dans la console
- tous de type char*
- *argv[0]* est toujours le nom de l'exécutable
- *argv[1]* est le premier paramètre à proprement parler
- *argv[argc-1]* est le dernier paramètre

Exemple

echo.c

```c
int main(int argc, char* argv[]){
    for(int i=0; i < argc; i++){
        printf("%s\n", argv[i]);
    }
    return 0;
}
```

Affichage

```
thibault~/Bureau$ ./echo bienvenue en algo
./echo
bienvenue
en
algo
```