

Les structures et les tableaux

Gaël Thomas



CSC4103 – Programmation système
2019–2020

1 Du type primitif au type composé

- Jusqu'à maintenant, vous avez vu les types primitifs
 - ◆ `char`, `short`, `int`, `long` `long`
(signés par défaut, non signés si préfixés de `unsigned`)
 - ◆ `float` (4 octets), `double` (8 octets)
- Dans ce cours, nous apprenons à définir de nouveaux types de données
 - ◆ Une **structure** est constituée de sous-types **hétérogènes**
 - ◆ Un **tableau** est constitué de sous-types **homogènes**

1.1 Les structures

- Une structure est une définition d'un nouveau type de données
 - ◆ composé de sous-types nommés (primitifs ou composés)
 - ◆ possédant un nom

Remarque : les sous-types d'une structure s'appellent des champs

Définition d'une nouvelle structure avec :

```
struct nom_de_la_structure {  
    type1 nom_du_champs1;  
    type2 nom_du_champs2;  
    ...  
};
```

Par exemple :

```
struct nombre_complexe {  
    int partie_reelle;  
    int partie_imaginaire;  
};
```

Par convention, les noms de structures commencent par une minuscule en C

1.1.1 Déclaration d'une variable de type structure

- Une déclaration d'une variable de type structure se fait comme avec un type primitif :

```
struct nombre_complexe z1, z2, z3;
```

- On peut aussi initialiser les champs de la structure lors de la déclaration :

```
/* partie_relle de z prend la valeur 0 */
```

```
/* partie_imaginaire de z prend la valeur 1 */
```

```
struct nombre_complexe i = { 0, 1 };
```

```
/* autre solution : */
```

```
struct nombre_complexe j = { .partie_reelle=0, .partie_imaginaire=1 };
```

1.1.2 Accès aux champs d'une variable de type structure

- L'accès aux champs d'une variable de type structure se fait en donnant le nom de la variable, suivi d'un point, suivi du nom du champs :

```
struct point {
    int x;
    int y;
};

struct ligne {
    struct point p1;
    struct point p2;
};

void f() {
    struct point p;
    struct ligne l;

    p.x = 42;
    p.y = 17;
    l.p1.x = 1;
    l.p1.y = 2;
    l.p2 = p; /* copie p.x/p.y dans l.p2.x/l.p2.y */

    printf("[%d %d]\n", p.x, p.y);
}
```

1.2 Les tableaux

- Un tableau est un type de données composé de sous-types homogènes
- Les éléments d'un tableau peuvent être de n'importe quel type (primitif, structure, mais aussi tableau)
- Pour déclarer un tableau :

```
type_des_elements nom_de_la_variable[taille_du_tableau];
```

Par exemple :

```
int          a[5];          /* tableau de 5 entiers */
double       b[12];         /* tableau de 12 nombres flottants */
struct point c[10];         /* tableau de 10 structures points */
int          d[12][10];     /* tableau de 10 tableaux de 12 entiers */
                                     /* => d est une matrice 12x10 */
```

1.2.1 Accès aux éléments d'un tableau

- L'accès à l'élément n du tableau `tab` se fait avec `tab[n]`
- Un tableau est indexé à partir de zéro (éléments vont de 0 à $N - 1$)

```
void f() {
    int x[3];
    int y[3];
    int i;

    /* 0 est le premier élément, 2 est le dernier */
    for(i=0; i<3; i++) {
        x[i] = i;
        y[i] = x[i] * 2;
    }
}
```

1.2.2 Tableaux et structures

- On peut mixer les tableaux et les structures, par exemple :

```
struct point {
    int x;
    int y;
};

struct triangle {
    struct point sommets[3];
};

void f() {
    struct triangle t;

    for(i=0; i<3; i++) {
        t.sommets[i].x = i;
        t.sommets[i].y = i * 2;
    }
}
```


1.2.3 Différences par rapport à Java

- On ne peut pas accéder à la taille d'un tableau
- Lors d'un accès en dehors des bornes du tableau, l'erreur est silencieuse : c'est une erreur, mais elle n'est pas signalée immédiatement
=> parmi les erreurs les plus fréquentes (et les plus difficiles à repérer) en C

```
void f() {  
    int x[3];  
  
    x[4] = 42; /* Erreur silencieuse !!! */  
              /* Écriture à un emplacement aléatoire en mémoire */  
              /* le bug pourra apparaître n'importe quand */  
}
```

1.2.4 Initialisation d'un tableau lors de sa déclaration

- Un tableau peut être initialisé lorsqu'il est déclaré avec

```
type_element nom_variable[taille] = { e0, e1, e2, ... };
```

Par exemple : `int x[6] = { 1, 2, 3, 4, 5, 6 };`

- Comme pour les structures, on peut partiellement initialiser un tableau

Par exemple : `int x[6] = { 1, 1, 1 };`

1.2.5 Initialisation mixte de tableaux et structures

- On peut composer des initialisations de tableaux et de structures

```
struct point {  
    int x;  
    int y;  
};
```

```
struct triangle {  
    struct point sommets[3];  
};
```

```
struct triangle t = {  
    { 1, 1 },  
    { 2, 3 },  
    { 4, 9 }  
};
```

1.2.6 Tableaux et chaînes de caractères

Une chaîne de caractère est simplement un tableau de caractères terminé par le caractère `'\0'` (c'est à dire le nombre zéro)

```
char yes[] = "yes";
```

est équivalent à

```
char yes[] = { 'y', 'e', 's', '\0' };
```

1.3 Passage par valeur et par référence

- En C, il existe deux types de passage d'arguments :
 - ◆ Passage par valeur :
l'argument est copiée de l'appelé vers l'appelant
=> l'argument et sa copie sont deux variables différentes
 - ◆ Passage par référence :
une référence vers l'argument de l'appelant est donné à l'appelé
=> l'appelant et l'appelé partagent la même donnée
- Par défaut :
 - ◆ Les **tableaux** sont passés par **référence**
Un argument de type tableau est déclaré avec `type nom[]`, sans la taille
 - ◆ Les **autres types** sont passés par **copie**

1.3.1 Passage par valeur – les types primitifs

```
/* le x de f et le x du main sont deux variables distinctes */  
/* le fait qu'elles aient le même nom est anecdotique */  
void f(int x) {  
    x = 666;  
    printf("f : x = %d\n", x);           /* f : x = 666 */  
}  
  
int main() {  
    int x = 42;  
    f(x);                               /* x est copié dans f */  
    /* => le x de main n'est donc pas modifié par f */  
    printf("g : x = %d\n", x);         /* g : x = 42 */  
    return 0;  
}
```

1.3.2 Passage par valeur – les structures

```
struct point {
    int x;
    int y;
};

void f(struct point p) {
    p.x = 1;
    printf("(%d, %d)\n", p.x, p.y);          /* => (1, 2) */
}

int main() {
    struct point p = { -2, 2 };
    f_p(p);                                  /* p est copié dans f */
    printf("(%d, %d)\n", p.x, p.y);        /* => (-2, 2) */
    return 0;
}
```

1.3.3 Passage par référence – les tableaux

```
void print(int x[], int n) {
    for(int i=0; i<n; i++) {
        printf("%d ", x[i]);
    }
    printf("\n");
}
```

```
int main() {
    int tab[] = { 1, 2, 3 };

    print(tab, 3); /* => 1 2 3 */
    f(tab);
    print(tab, 3); /* => 1 42 3 */

    return 0;
}
```

```
/* x est une référence vers le tableau original */
void f(int x[]) {
    x[1] = 42; /* => modifie l'original */
}
```


1.4 Notions clés

■ Les structures

- ◆ Une structure définit un nouveau type de donné
- ◆ Définition : `struct nom { type_1 champs_1; type_2 champs_2; ... };`
- ◆ Déclaration : `struct nom var = { v1, v2 };`
- ◆ Utilisation : `var.champs_i`

■ Les tableaux

- ◆ Un tableau est un type de donné
- ◆ Déclaration : `int tab[] = { 1, 2, 3 };`
- ◆ Utilisation : `tab[i]`
- ◆ Une chaîne de caractère est un tableau de caractère terminé par un zéro

■ Passage par valeur ou par référence

- ◆ Les tableaux sont passés par référence
- ◆ Les autres types sont passés par valeur