C : Bases 2 : types structurés

Pierre Karpman

Lycée Champollion MP2I
https://membres-ljk.imag.fr/Pierre.Karpman/CPGE/2025/

Table des matières

Principe

Un type structuré agglomère plusieurs types en un seul

Deux genres de types structurés au programme

- Enregistrements : ensemble d'éléments de types hétérogènes, en nombre fixe
  - À définir nous-même
- Tableaux de taille fixe : ensemble d'éléments d'un même type, en nombre fixe
  - Prédéfinis, pour n'importe quelle type d'éléments et n'importe quelle longueur

Table des matières

Tableaux

Principe

- Stocke plusieurs éléments d'un même type (`int`, genre `struct`, tableau…
- Pour nous : en nombre (de *longueur*) fixe (*fixed-length array* ou *FLA*)
- Le type d'un tableau donne le type des éléments et la longueur, par ex. `int [10]`

Déclaration

Syntaxe

Pour un tableau à une dimension :

`type var[longueur]`

avec `longueur` un *littéral* entier strictement positif (sinon le tableau n'est pas de taille fixe…)

Exemple

```
int t[10];
```

Accès aux éléments

Syntaxe

`var[i]`
pour `i` entre 0 et la longueur du tableau *moins 1*

Warning

Attention à ne pas accéder à des indices invalides : ARR30-C. Do not form or use out-of-bounds pointers or array subscripts

Asan

Quand un programme manipule des tableaux, *toujours* compiler avec l'*address sanitizer* *via* `-fsanitize=address`

Démo.

Initialisation

Syntaxe

Un tableau peut s'initialiser à la déclaration avec la syntaxe :

```
type var[n] = {val0, val1, /*...*/};
```

L'initialiseur n'est *pas* un littéral et l'on s'abstiendra rigoureusement de le transformer en *compound literal*

Exemple

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

(Non-)affectation

On peut lire et écrire les éléments d'une variable de type tableau comme n'importe quelle variable habituelle, mais pas le tableau *lui-même*

```
int t[10];
int s[10];
int a
// ...
a = t[7];
t[3] = 1;
t[2] = t[0] + t[1];
s = t; // impossible
```

Passage d'arguments

Il est (fonctionnellement) *nécessaire* de renseigner la longueur d'un paramètre tableau si elle peut varier d'un argument à un autre

Syntaxe de déclaration de paramètres tableau

```
void fun(int t[10]);
void fun2(size_t tn, int t[tn]);
```

Le passage d'argument est par valeur, mais fonctionnellement comme si par référence

Démo.

Explications.... plus tard

Limitations

On ne peut pas définir des tableaux trop grands

Démo.

Une fonction ne peut pas renvoyer un tableau déclaré dans son corps

Démo.

Explications.... plus tard

Table des matières

Motivation

Pouvoir

- Regrouper ensemble des variables logiquement liées entre elles
- Diminuer le nombre de paramètres d'une fonction
- Permettre à une fonction de renvoyer plus d'une valeur

Syntaxe

Définition

- Il faut définir un type enregistrement pour pouvoir l'utiliser
- Typiquement au niveau d'un fichier
- Syntaxe :

```
struct nom_de_la_struct
{
    type_champ1 nom_champ1;
    type_champ2 nom_champ2;
    // ...
};
```

- Le nom du type est `struct nom_de_la_struct` (en deux mots)

Exemple

```
struct gcof
{
    unsigned d;
    int u;
    int v;
};

struct gcof xgcd(unsigned a, unsigned b)
{
    struct gcof res;
    // ...
    return res;
}
```

Définition avec renommage

Syntaxe

Après définition de `struct ma_struct` :

```
typedef struct ma_struct nouveau_nom_pour_ma_struct;
```

Exemple

```
struct gcof_s
{
    // ...
};
typedef struct gcof_s gcof;

gcof xgcd(unsigned a, unsigned b) {
    gcof res;
    // ...
    return res;
}
```

Affectation, passage par valeur, comparaisons

Les variables et arguments de genre `struct` sont affectés normalement, et passés par valeur ; elles ne sont pas comparables

Exemple

```
void fun(void)
{
    struct s a;
    struct s b;
    // ...
    a = b;
}
```

Accès aux champs

Syntaxe

L'accès (pour lecture, écriture…) utilise la syntaxe :
`var.nom_champ`

Exemple

```
struct gcof xgcd(unsigned a, unsigned b)
{
    struct gcof res;
    res.d = 0;
    res.u = 0;
    res.v = 0;
    // ...
}
```

Initialisation

Syntaxe

Une structure peut s'initialiser à la déclaration avec la syntaxe :

```
struct ma_struct var = {.nom_champ1 = val1,
                        .nom_champ2 = val2 /*...*/};
```

L'initialiseur n'est *pas* un littéral et l'on s'abstiendra rigoureusement de le transformer en *compound literal*

Exemple

```
struct gcof xgcd(unsigned a, unsigned b)
{
    struct gcof res = {.d = 0, .u = 0, .v = 0};
    // ...
}
```