

Programmation avancée

Structures cartésiennes

Walter Rudametkin

Walter.Rudametkin@polytech-lille.fr
<https://rudametw.github.io/teaching/>

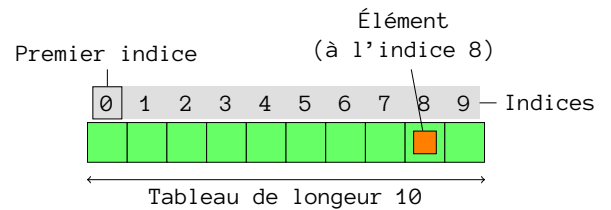
Bureau F011
 Polytech Lille

CM1

1 / 22

Tableaux

- Collections indicées d'informations de même type (homogène)



Types de données

- char, short, int, long, long long, float, double, long double

2 / 22

Structures cartésiennes

- n-uplet d'informations de types quelconque rangées dans des champs
 - Informations complexes (composites)
 - Des n types de variables personnalisés z
- Notation

```
type <ST> = structure
    champ1: <T1>
    champ2: <T2>
    ...
    champn: <Tn>
fin
```

3 / 22

Structures cartésiennes

Domaine des valeurs d'une structure

- Produit cartésien des domaines des champs
 - $\text{Dom}(ST) = \text{Dom}(T_1) \times \text{Dom}(T_2) \times \dots \times \text{Dom}(T_n)$
- Accès aux champs par notation pointée
 - $v : \langle ST \rangle$, accès au champs i $v.\text{champ}_i$

Exemple

```
type Ouvrage = structure
    code: Entier
    titre: Chaine
fin
```

4 / 22

Structures cartésiennes

```
type Complexe = structure
    reelle, imag: Reel
fin
fonction plus(c1,c2) : Complexe
    donnees: c1,c2: Complexe
    locales: c: Complexe
    c.reelle := c1.reelle + c2.reelle
    c.imag := c1.imag + c2.imag
    resultat: c
fin
```

fonction
 Utilisation:

```
c1,c2,c3 : Complexe
c3 := plus(c1,c2)
```

5 / 22

Structures imbriquées

- Le types des champs est quelconque
 - Ils peuvent même être des structures

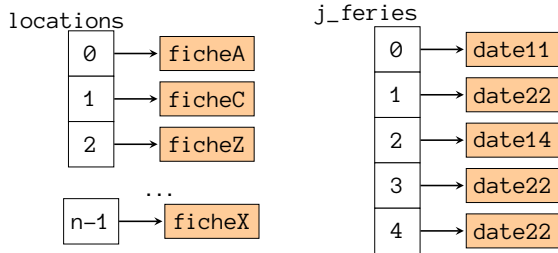
```
type Date = structure
    jour, mois, annee : Entier
fin
```

```
type Fiche = structure
    emprunt : Ouvrage
    date : Date
fin
```

```
//F est une variable de type Fiche
F: Fiche
//Les accès aux champs sont de type
⇒ F.date: Date ⇒ F.date.jour: Entier
⇒ F.emprunt: Ouvrage ⇒ F.emprunt.titre: Chaine
```

6 / 22

Tableaux de structures



```
//Accès
locations : vecteur[N] de Fiche
f: Fiche ; f ← locations[3]
    ⇒ f.date: Date ⇒ f.date.jour: Entier
    ⇒ f.emprunt: Ouvrage ⇒ f.emprunt.titre: Chaine
```

7/22

Déclaration de structures en C

- Le mot clé struct permet de définir des modèles de structures:


```
struct <désignateur> {
    <déclaration de champ1>;
    <déclaration de champ2>;
    ...
    <déclaration de champn>;
}; //Point-virgule obligatoire
```

 où:
 - <désignateur> est le nom (facultatif) du modèle
 - <déclarations de champ> comme des déclarations de var mais sans initialisation

8/22

Exemples de structures en C

```
/* definition de la structure*/
struct date {int j,m,a;};

/*2 variables selon le modèle date*/
struct date d1, d2;

/*définition et utilisation immédiate*/
struct complexe {float reelle, imag;} c1, c2;

/*rappel du même modèle*/
struct complexe c3;
```

9/22

Définitions de synonymes de types (typedef)

- typedef permet de donner des alias (synonymes) à des définitions de types dans toute zone déclarative :


```
typedef <un_type> <synonyme_du_type>
```

 <un_type> a la même syntaxe qu'une déclaration de variable, et <synonyme_du_type> désigne le nouveau nom du type
- Donnez des noms plus simples pour faciliter l'écriture et augmenter la lisibilité
- Exemples :


```
typedef unsigned char octet;
typedef struct ma_structure * ptr_ma_struct;
typedef struct S S;
```

10/22

Tableaux de structures

Exemples :

```
typedef int * PtInt;
typedef int Matrice[10][20];
typedef struct date Date;
typedef struct {
    int numero;
    char titre[50];
} Ouvrage;
```

Conséquences

```
PtInt p; ⇔ int * p;
Matrice m; ⇔ int m[10][20];
Date d; ⇔ struct date d;
Ouvrage o; ⇔ struct {
    int numero;
    char titre[50];
} o;
```

Typedef rend superflu le nom du modèle (sauf dans le cas de structures récursives...).

11/22

Manipulations de structures: Exemple Date

```
typedef struct Date {int jour, mois, annee;} Date; /* option 1 */
typedef struct {int jour, mois, annee;} Date; /* option 2 */
Date d1 = {18,5,2012}; Date d2 = {24,12,2015}; /* variables */
sizeof(Date); /* taille de la structure Date = 3*sizeof(int) */
```

d1	18	05	2012
d2	24	12	2015

- Sélection de champ : opérateur . de plus forte priorité


```
d1.jour = d2.jour;
scanf("%d",&d2.jour);
/* équivalent à scanf("%d", &(d2.jour)); */
```

12/22

Affectation entre structures

- Copie **champs par champ** (contrairement aux tableaux).
 - Attention aux pointeurs, "**shallow copy**"

Avec tableau

```
struct Sarray {
    int p[3];
};
struct Sarray sa1, sa2;

sa1.p[0]=10; sa1.p[1]=20;
sa1.p[2]=30;

sa2 = sa1;
```

Avec pointeur

```
1 struct Spointer {
2     int * p;
3 };
4 struct Spointer sp1, sp2;
5 sp1.p = malloc(3*sizeof(*sp1.p));
6 sp1.p[0] = 10; sp1.p[1] = 20;
7 sp1.p[2] = 30;
8
9 sp2 = sp1;
10 free(sp1.p);
```

Quelles sont les différences ?

13/22

Quelques limites

- Pas de comparaisons (==, !=, >, <, ...)
- Pas d'opérateurs arithmétiques
- Pas de E/S (scanf, printf, ...)
- Pas de support de "deep copy" (pas de copie des valeurs "pointées", seulement les valeurs des pointeurs)
- Attention aux passages des structures dans des fonctions (passage-par-copie des structs, implique "Shallow Copy")

Beaucoup de choses à programmer
à la main !!!

14/22

Tableaux dans les structures

```
typedef struct {
    int numero;
    char titre[50];
} Ouvrage;
Ouvrage x,y; //variables
```

	numéro	titre[0]	titre[1]	titre[49]
x	int	char	char	char

- $y = x$ ✓
 - $y.titre = x.titre$ ⚠ IMPOSSIBLE
- Et si titre était un char * ???

15/22

Tableaux dans les structures

- $y.titre = x.titre$ ⚠ IMPOSSIBLE
 - Copiez les caractères un par un
 - Ou utilisez les fonctions de C dédiées : strcpy, strncpy, strcat, ...

//attention aux caractères de fin de chaîne '\0'
strcpy(y.titre, x.titre);

strncpy(y.titre, x.titre, 50); //donnez la taille
y.titre[50 - 1] = '\0'; //garantir fin de chaîne

/* Ou concaténer avec une chaîne vide: */
*y.titre = '\0'; strcat(y.titre, x.titre, 50-1);

16/22

Structures dans les structures

```
typedef struct {int numero; char titre[50];} Ouvrage;
typedef struct {int jour, mois, annee;} Date;
typedef struct Fiche {
    Ouvrage emprunt ; //struct imbriquée
    Date date ; //struct imbriquée
} Fiche ;
```

//Déclaration et Initialisation en 1:
Fiche f = {{23,"H. Potter"}, {12,5,2006}}; //C99

Accès aux champs

- f.date.jour de type int
- f.emprunt.titre de type char []

17/22

Tableaux de structures

- Utilisation similaire aux tableaux "normaux"

```
1 Fiche tableau_fiches[3];
2 Ouvrage o1; Date d1;
3 //Fiche 1 : initialization des sous structures
4 o1.numero=23; strcpy(o1.titre,"H. Potter");
5 d1.jour=12 ; d1.mois=5; d1.annee=2006;
6 tableau_fiches[0].emprunt=o1; tableau_fiches[0].date=d1;
7 //Fiche 2 et Fiche 3
8 Fiche f2 = (Fiche) {{23,"H. Potter"}, {15,7,2006}}; //C99
9 Fiche f3 = (Fiche) {{30,"Hamlet"}, {12,5,2006}}; //C99
10 tableau_fiches[1]=f2; tableau_fiches[2]=f3;
```

Accès aux champs

- tableau_fiches[2].date.mois de type int
- tableau_fiches[0].emprunt.titre de type char []

18/22

Tableaux de structures: initialisation avancée

- Toujours pareil que pour les tableaux "normaux"

```
1 //Définition et initialisation
2 Fiche tableau_fiches [3] = {
3     {{23,"H. Potter"}, {15,7,2006}},
4     {{30,"Hamlet"}, {12,5,2006}},
5     {{35,"Don Quichotte"}, {12,5,2006}}
6 };
```

- Attention aux accolades, l'initialisation des sous structures et tableaux en nécessite aussi !
- Ça ne marche QUE si on définit et initialise toutes les variables d'un coup

Astuce

- Créer des fonctions utilitaires qui prennent des valeurs en paramètre et renvoient des structures

19/22

Passage de structures en paramètre

- *Passage par valeur* (données)

- L'affectation entre structures étant possible, le passage par valeur ou en tant que résultat de fonction l'est aussi.

```
1 struct complexe {float reelle, imag;};
2 typedef struct complexe Complexe;
3
4 /* Prend deux Complexe en paramètre,
5    renvoi leur addition */
6 Complexe plus (Complexe c1, Complexe c2) {
7     Complexe r;
8     r.reelle = c1.reelle + c2.reelle;
9     r.imag = c1.imag + c2.imag;
10    return r;
11 }
```

20/22

Passage de structures en paramètre

- *Passage par pointeur/référence* (données et résultat)

- Exemple: traduire un point en x

```
1 typedef struct {int x, y;} Point ;
2 void traduire (Point *pp, int dx) {
3     (*pp).x = (*pp).x + dx;
4     /* attention aux priorités */
5 }
6 int main () {
7     Point p;
8     scanf("%d%d", &p.x, &p.y);
9     traduire(&p,10);
10 }
```

21/22

L'opérateur ->

- L'écriture (*pp).x est très courante d'où l'opérateur '->' applicable à tout pointeur de structure:

pointeur->champ \iff (*pointeur).champ

- Exemple

```
void traduire (Point *pp, int dx) {
    pp->x = pp->x + dx;
}
```

22/22