


Éléments d'informatique – Cours 10.
Types composés: structures (enregistrements)

Pierre Boudes

29 novembre 2011



This work is licensed under the *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License*.

- *Éléments d'architecture des ordinateurs (+mini-assembleur)* 
- *Éléments de systèmes d'exploitation*
- Programmation structurée impérative (éléments de langage C)
 - *Structure d'un programme C*
 - *Variables : déclaration (et initialisation), affectation*
 - *Évaluation d'expressions*
 - *Instructions de contrôle : if, for, while*
 - *Types de données : entiers, caractères, réels, tableaux, enregistrements*
 - *Fonctions d'entrées/sorties (scanf/printf)*
 - *Écriture et appel de fonctions*
 - *Débogage*
- *Notions de compilation*
 - *Analyse lexicale, analyse syntaxique, analyse sémantique*
 - *préprocesseur du compilateur C (include, define)*
 - *Édition de lien*
- Algorithmes élémentaires
- Méthodologie de résolution, manipulation sous linux

Plan de la séance

Introduction

Déclaration d'un type utilisateur struct

Utilisation d'un type utilisateur struct

Traces avec type utilisateur structure

Intérêt des structures (enregistrements)

Quelques erreurs communes

Introduction

Introduction

Les données déclarées comme ayant un type de base (int, double, char) ne contiennent qu'une seule valeur. Tandis qu'une donnée de type structure **englobe plusieurs valeurs**.

Introduction

Les données déclarées comme ayant un type de base (int, double, char) ne contiennent qu'une seule valeur. Tandis qu'une donnée de type structure englobe plusieurs valeurs.

- Une structure s'utilise comme un type.

Introduction

Les données déclarées comme ayant un type de base (int, double, char) ne contiennent qu'une seule valeur. Tandis qu'une donnée de type structure englobe plusieurs valeurs.

- Une structure s'utilise comme un type.
- Chacune des valeurs est accessible à l'aide d'un nom, fixé à la déclaration de la structure. On parle des champs de la structure.

Introduction

Les données déclarées comme ayant un type de base (int, double, char) ne contiennent qu'une seule valeur. Tandis qu'une donnée de type structure **englobe plusieurs valeurs**.

- Une structure s'utilise comme un **type**.
- Chacune des valeurs est accessible à l'aide d'un nom, fixé à la déclaration de la structure. On parle des **champs** de la structure.
- Un type structure doit être déclaré avant d'être utilisé. Ne pas confondre : **déclaration de variable** (dans une fonction) et **déclaration de type structure** ou **déclaration de fonction**.

Introduction

Les données déclarées comme ayant un type de base (int, double, char) ne contiennent qu'une seule valeur. Tandis qu'une donnée de type structure **englobe plusieurs valeurs**.

- Une structure s'utilise comme un **type**.
- Chacune des valeurs est accessible à l'aide d'un nom, fixé à la déclaration de la structure. On parle des **champs** de la structure.
- Un type structure doit être déclaré avant d'être utilisé. Ne pas confondre : **déclaration de variable** (dans une fonction) et **déclaration de type structure** ou **déclaration de fonction**.
- Les valeurs des champs d'une variable (d'un type structure donné) peuvent être accédées et modifiées individuellement à l'aide de la **notation pointée**.

Introduction

Les données déclarées comme ayant un type de base (int, double, char) ne contiennent qu'une seule valeur. Tandis qu'une donnée de type structure **englobe plusieurs valeurs**.

- Une structure s'utilise comme un **type**.
- Chacune des valeurs est accessible à l'aide d'un nom, fixé à la déclaration de la structure. On parle des **champs** de la structure.
- Un type structure doit être déclaré avant d'être utilisé. Ne pas confondre : **déclaration de variable** (dans une fonction) et **déclaration de type structure** ou **déclaration de fonction**.
- Les valeurs des champs d'une variable (d'un type structure donné) peuvent être accédées et modifiées individuellement à l'aide de la **notation pointée**.
- On peut aussi manipuler globalement la valeur d'une donnée de type structure. Par exemple : faire une affectation entre variables d'un même type structure (copie de tous les champs).

Déclaration d'un type utilisateur struct

Comme avec les variables, comme avec les fonctions, avant de pouvoir utiliser un type structure, il faut déclarer son nom (convention : suffixe `_s`), et la liste de ses champs sous la forme `type nom_champ;`

Déclaration d'un type utilisateur struct

Comme avec les variables, comme avec les fonctions, avant de pouvoir utiliser un type structure, il faut déclarer son nom (convention : suffixe `_s`), et la liste de ses champs sous la forme `type nom_champ;`

```
struct bulletin_s
{
    double temperature; /* temperature de l'air */
    int force;           /* force du vent (Beaufort) */
}; /* <-- attention ';' (cas particulier) */
```

Déclaration d'un type utilisateur struct

Comme avec les variables, comme avec les fonctions, avant de pouvoir utiliser un type structure, il faut déclarer son nom (convention : suffixe `_s`), et la liste de ses champs sous la forme `type nom_champ;`

```
struct bulletin_s
{
    double temperature; /* temperature de l'air */
    int force;           /* force du vent (Beaufort) */
}; /* <-- attention ';' (cas particulier) */
```

- Cette déclaration est placée en dehors de toute fonction, entre les définitions de constantes symboliques (`#define ...`) et les déclarations de fonctions utilisateur.

Déclaration d'un type utilisateur struct

Comme avec les variables, comme avec les fonctions, avant de pouvoir utiliser un type structure, il faut déclarer son nom (convention : suffixe `_s`), et la liste de ses champs sous la forme `type nom_champ;`

```
struct bulletin_s
{
    double temperature; /* temperature de l'air */
    int force;           /* force du vent (Beaufort) */
}; /* <-- attention ';' (cas particulier) */
```

- Cette déclaration est placée en dehors de toute fonction, entre les définitions de constantes symboliques (`#define ...`) et les déclarations de fonctions utilisateur.
- L'effet de cette déclaration est de signaler au compilateur qu'un nouveau type est disponible et comment il peut être utilisé (liste des champs). Aucun espace mémoire n'est réservé à ce moment (ce n'est pas une déclaration de variable).

Déclaration d'un type utilisateur struct

Comme avec les variables, comme avec les fonctions, avant de pouvoir utiliser un type structure, il faut déclarer son nom (convention : suffixe `_s`), et la liste de ses champs sous la forme `type nom_champ;`

```
struct bulletin_s
{
    double temperature; /* temperature de l'air */
    int force;          /* force du vent (Beaufort) */
}; /* <-- attention ';' (cas particulier) */
```

- Cette déclaration est placée en dehors de toute fonction, entre les définitions de constantes symboliques (`#define ...`) et les déclarations de fonctions utilisateur.
- L'effet de cette déclaration est de signaler au compilateur qu'un nouveau type est disponible et comment il peut être utilisé (liste des champs). Aucun espace mémoire n'est réservé à ce moment (ce n'est pas une déclaration de variable).
- **Attention au point-vigule final.**

Utilisation d'un type utilisateur struct : variables

```
int main()
{
    struct bulletin_s x = {0.5, 4};
    struct bulletin_s y;

    y = x; /* copie globale */
    x.temperature = 13.4; /* modif. d'un champ */
    ...
}
```


Utilisation d'un type utilisateur struct : variables

```
int main()
{
    struct bulletin_s x = {0.5, 4};
    struct bulletin_s y;

    y = x; /* copie globale */
    x.temperature = 13.4; /* modif. d'un champ */
    ...
}
```

- Initialisation : syntaxe proche de celle des tableaux.

Utilisation d'un type utilisateur struct : variables

```
int main()
{
    struct bulletin_s x = {0.5, 4};
    struct bulletin_s y;

    y = x; /* copie globale */
    x.temperature = 13.4; /* modif. d'un champ */
    ...
}
```

- Initialisation : syntaxe proche de celle des tableaux.
- la déclaration d'une variable de type structure reprend le mot clé struct et le nom donné à la structure.

Utilisation d'un type utilisateur struct : variables

```
int main()
{
    struct bulletin_s x = {0.5, 4};
    struct bulletin_s y;

    y = x; /* copie globale */
    x.temperature = 13.4; /* modif. d'un champ */
    ...
}
```

- Initialisation : syntaxe proche de celle des tableaux.
- la déclaration d'une variable de type structure reprend le mot clé struct et le nom donné à la structure.
- On accède aux éléments d'une structure à l'aide de la notation pointée : `nom_variable.nom_champ`

Utilisation d'un type utilisateur struct : fonctions

```
/* declaration de fonctions utilisateur */  
struct bm_s moyenne_bm(struct bm_s x, struct bm_s y);
```

Utilisation d'un type utilisateur struct : fonctions

```
/* declaration de fonctions utilisateur */  
struct bm_s moyenne_bm(struct bm_s x, struct bm_s y);
```

On emploie `struct nom_struct`, comme pour une déclaration de variable.

```
/* definitions des fonctions utilisateur */  
struct bm_s moyenne_bm(struct bm_s x, struct bm_s y)  
{  
    struct bm_s nouveau_bm;  
  
    nouveau_bm.temperature = (x.temperature  
                             + y.temperature) / 2.0;  
    nouveau_bm.force = (x.force + y.force) / 2;  
  
    return nouveau_bm;  
}
```

Traces avec type utilisateur structure

```
moyenne_bm({14.1, 2}, {9.5, 4})
```

ligne	x		y		nouveau_bm		A.
	temperature	force	temperature	force	temperature	force	
ini.	14.1	2	9.5	4	?	?	
40					11.8		
42						3	
44	RETOURNE {11.8, 3}						

Intérêt des structures (enregistrements)

Intérêt des structures :

- *lisibilité* : regrouper un ensemble de données dans un même type, nommé de façon explicite, facilite la relecture du code ;

Intérêt des structures (enregistrements)

Intérêt des structures :

- *lisibilité* : regrouper un ensemble de données dans un même type, nommé de façon explicite, facilite la relecture du code ;
- *augmente les possibilités* : les structures permettent d'écrire des fonctions qui retournent plusieurs valeurs, en l'absence de pointeurs.

Intérêt des structures (enregistrements)

Intérêt des structures :

- *lisibilité* : regrouper un ensemble de données dans un même type, nommé de façon explicite, facilite la relecture du code ;
- *augmente les possibilités* : les structures permettent d'écrire des fonctions qui retournent plusieurs valeurs, en l'absence de pointeurs.
- *modularité* : on peut rajouter des champs très facilement, avec très peu de modifications.

Intérêt des structures (enregistrements)

Intérêt des structures :

- *lisibilité* : regrouper un ensemble de données dans un même type, nommé de façon explicite, facilite la relecture du code ;
- *augmente les possibilités* : les structures permettent d'écrire des fonctions qui retournent plusieurs valeurs, en l'absence de pointeurs.
- *modularité* : on peut rajouter des champs très facilement, avec très peu de modifications.
- *Incontournables* : les langages orientés objets généralisent la notion de structure. Un objet est une structure dont les champs peuvent être aussi bien des données que des fonctions (hors programme).

Erreurs communes

- Message d'erreur *étrange* :

```
7  struct bm_s
8  {
9      double temperature; /* temperature de l'air */
10     int force;           /* force du vent (Beaufort)
11 }
12
13 /* Declaration des fonctions utilisateur */
14 void afficher_bm(struct bm_s);
```

14: error: two or more data types in declaration specifiers

Erreurs communes

- Message d'erreur *étrange* :

```
7  struct bm_s
8  {
9      double temperature; /* temperature de l'air */
10     int force;           /* force du vent (Beaufort)
11 }
12
13 /* Declaration des fonctions utilisateur */
14 void afficher_bm(struct bm_s);
```

14: error: two or more data types in declaration specifiers

Oubli du point-virgule!

Erreurs communes

- Message d'erreur *étrange* :

```

7  struct  bm_s
8  {
9      double temperature; /* temperature de l'air */
10     int force;           /* force du vent (Beaufort)
11 }
12
13 /* Declaration des fonctions utilisateur */
14 void afficher_bm(struct bm_s);

```

14: error: two or more data types in declaration specifiers

Oubli du point-virgule!

- Champ inexistant :

```

22     x.toto = 3; /* erreur: pas de champs toto */

```

prog.c: In function 'main':

prog.c:22: error: 'struct bm_s' has no member named 'toto'