Introduction à la programmation en C

Cours 4
25/01/2013

Structures et nouveaux types .

Samy BLUSSEAU, Miguel COLOM

I. Structures

I. Structures

- Les structures permettent de **créer des types plus complexes** que les ceux vus jusqu'à présent *(int, float, …)*.

- *Exemple :* création d'un nouveau type représentant les nombres complexes :

```
struct complex {

    float re ;
    float im ;

}  ;

typedef struct complex complex ;
```

Avec **struct** on définit ici une structure, qui s 'appelle "*struct complex* ".

Avec typedef on définit un nouveau type, "*complex*".

Les variables de type *complex* ont **deux champs de type float : "re" et "im"**.

I. Structures

Désormais, **on peut utiliser ce nouveau type comme tous les autres**.

```
complex z ;
```

On accède aux **champs du type** défini, comme suit :

```
z.re = 1 ;
z.im = 1.41 ;
```

Ou, si on a un pointeur sur le type, on peut le déréférencer et accéder aux champs gace à l'opérateur " **->** " :

```
complex* pz ;

pz->re = 1 ;  /* Equivalent à (*pz).re = 1 ;*/
pz->im = 1.41 ;
```

I. Structures

Les structures permettent de **regrouper, dans une même variable, plusieurs variables qui caractérisent une même entité**.

Par exemple, pour représenter une matrice, on peut créer une structure **matrice** dont les champs sont les dimensions de la matrice et un tableau.

Cela permet

- d'ajouter de la **cohérence** au code,
- de **remplacer plusieurs paramètres** de fonctions **par un seul**

Exercice :

- Reprendre la bibliothèque algebre_lineaire.c, et y ajouter une structure **matrice** représentant une matrice de float. Ses champs seront les dimensions et un tableau de coefficients.

- Modifier les fonctions implémentées de façon à utiliser ce nouveau type

- On pourra également ajouter des fonctions « new_matrix » et « free_matrix », pour créer une matrice en allouant sa mémoire, et libérer sa mémoire.

I. Structures

Les champs peuvent être de tout type, y compris des pointeurs...

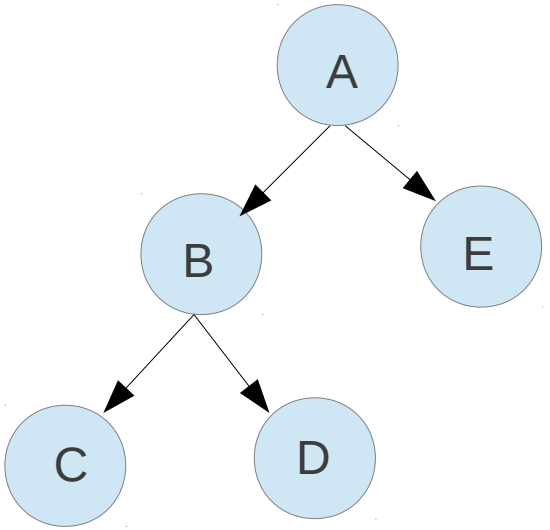
...et notamment des pointeurs vers le type lui-même (**structure récursive**).

Exemple : Nœud d'un arbre

```
struct noeud{

        int contenu  ;
        struct nœud * pere ;
        struct nœud *fils_gauche  ;
        struct nœud *fils_droit  ;

} ;

typedef struct nœud nœud ;
```

Exercice :

- Dans un fichier arbres.h, définir une structure de noeud
- Dans un fichier test_arbres.c, construire l'arbre binaire représenté ci-dessous.
- Dans un fichier arbres.c, écrire une fonction **get_level**, prenant un nœud en paramètre et retournant la profondeur de ce nœud dans l'arbre auquel il appartient.
- get_level pourra appeler d'autres fonctions que vous implémenterez, comme des fonctions **void set_childs (node* pere, node* fg, node* fd)**, **void set_parent(const node* fils, node* pere)**.



profondeur 0

profondeur 1

profondeur 2