Programmation C - Structures

Laurent Signac – CC-BY-SA – 07-09-20 1643 9daaee63a16a2d602382

Les structures permettent de créer un nouveau type de données représentant des *agrégats* de plusieurs éléments, nommés, de types éventuellement différents.

1 Définition de structures

Si par exemple nous voulons créer un type `fiche` capable de stocker l'identité, l'âge et la taille de quelqu'un, nous pouvons écrire:

```
struct fiche {
  char nom[255];
  int age;
  float taille;
};
```

Puis nous déclarons une variable ayant ce type et la renseignons:

```
struct fiche fiche1;

strcpy(fiche1.nom,"Dupond");
fiche1.age=35;
fiche1.taille=1.72;
```

Les champs de la structure sont utilisables de la même façon que des variables ordinaires.

2 Affectation de structures

Lors d'une affectation de type :

```
fiche2=fiche1;
```

tous les champs sont recopiés. En fait, c'est toute la zone mémoire contenue dans la structure qui est recopiée, même si celle-ci contient des chaînes de caractères ou des tableaux (qui d'habitude ne sont pas recopiés par une simple affectation).

3 Utilisation de `typedef`

La répétition du mot clé `struct` lors de la déclaration des variable (ou des prototypes des fonctions) est fastidieuse. Aussi, on utilise souvent conjointement avec les structures la possibilité, en C, de donner de nouveaux noms aux types.

Par exemple, `typedef int entier;` permettra d'utiliser le mot clé `entier` à la place de `int`. De même:

```
struct fiche {
  char nom[255];
  int age;
  float taille;
};
typedef struct fiche fiche;
```

ou encore:

```c
typedef struct {
  char nom[255];
  int age;
  float taille;
} fiche ;
```

permettra d'utiliser le mot `fiche` à la place de `struct fiche`.

4 Pointeurs et structures

Nous pouvons bien entendu définir un pointeur vers une structure:

```c
struct fiche * pf;
```

L'accès aux champs peut alors se faire ainsi:

```c
(*pf).age=25;
```

ou ainsi:

```c
pf->taille=1.69;
```