

## Chapitre trois

### LES STRUCTURES

Nous continuons notre étude des structures de données qui sont prédéfinies dans la plupart des langages informatiques. La structure de tableau permet de regrouper un certain nombre de données de *même* type, et de plus le type doit être assez simple (pour l'instant). Les *types structurés* vont nous permettre de manipuler des données structurées dont les types des éléments peuvent être différents.

Cela correspond à la notion de *masque* d'un formulaire à remplir : on demande le nom ; il y a un certain nombre de cases, indiquant par là le nombre maximum de lettres permises ; on demande l'année de naissance et on s'attend à voir un entier.

# 1 Définition des types structurés

## 1.1 Un exemple introductif

Introduction.- Le problème du *répertoire téléphonique* se pose à chacun d'entre nous. Au début, nous connaissons par cœur les numéros de téléphone des gens que nous connaissons. Peu à peu nous en avons trop et il faut prendre quelques notes. Les notes s'entassent et on aimerait bien mettre un peu d'ordre. On pense à un répertoire téléphonique.

Un répertoire réaliste comprendrait le nom, le prénom, le numéro de téléphone fixe, le numéro du mobile, le numéro du bureau, l'adresse e-mail et bien plus. Considérons un cas très simple avec juste un nom et un numéro de téléphone par item.

Une telle structure de données s'appelle *enregistrement* (en anglais *record*) en PASCAL et tout simplement **structure** en langage C. Intéressons-nous à l'exemple que nous venons d'évoquer que nous étudierons au fur et à mesure dans les différentes divisions.

Le problème.- Établissons une fiche pour chacun des items de notre répertoire. Simplifions le contenu de cette fiche, comme nous l'avons déjà dit ; vous pourrez l'améliorer plus tard pour obtenir quelque chose de plus réaliste si vous le désirez. Disons qu'une fiche contiendra un nom et un numéro de téléphone.

Exercice 1.- Pourquoi le masque de ce formulaire est-il peu réaliste ?

Précisions.- Précisons un peu la nature des données :

- le nom sera un mot de longueur inférieure à 29 ;
- le numéro de téléphone sera un mot de longueur inférieure à 19.

Un premier exemple de déclaration de type structuré.- Ces décisions nous permettent de considérer la fiche comme un élément du type structuré suivant :

```
struct ITEM
{
    char NOM[30];
    char TEL[20];
};
```

## 1.2 Syntaxe de la définition d'un type structuré

Introduction.- Nous venons de rencontrer un nouveau constructeur de types, permettant d'obtenir des types plus complexes à partir d'autres types, simples ou eux-mêmes déjà complexes. Ce type correspond à la définition de ce qu'on appelle un **masque** pour une fiche. Les éléments de ce type correspondent aux fiches proprement dites.

Définition d'un type structuré.- La syntaxe suit la règle suivante :

```
struct NOM
{
  type_1 champ_1;
  type_2 champ_2;
  - - - - -
  type_n champ_n;
};
```

où *NOM*, *champ\_1*, ... , *champ\_n* sont des identificateurs (non utilisés pour autre chose) et *type\_1*, ... , *type\_n* des types.

Chacun des identificateurs s'appelle un **champ** ou un **membre**.

Exercice.- Considérons l'exemple précédent. Combien y a-t-il de champs ? Quels sont leurs noms ? Quel est le type de chacun d'eux ?

## 2 Déclaration des variables structurées

### 2.1 Un exemple

Exemple.- Pour le type structuré ci-dessus on peut déclarer une variable, disons `personne`, de la façon suivante :

```
struct ITEM personne;
```

On remarquera la nécessité de répéter le mot clé `struct`.

Remarque.- On peut ne pas donner de nom au type structuré si on déclare en même le type et les variables de ce type comme dans l'exemple suivant :

```
struct
{
    char NOM [30];
    char TEL [20];
} personne;
```

mais ceci est à éviter.

### 2.2 Syntaxe de la déclaration d'une variable structurée

Une variable d'un type structuré (appelée parfois **variable structurée**) se déclare de la façon suivante :

```
struct NOM VAR;
```

où `VAR` est un identificateur et `NOM` le nom d'un type structuré.

## 3 Accès aux champs d'une structure

### 3.1 Syntaxe

Les variables d'un type structuré peuvent être manipulées globalement dans une affectation ou champ par champ. L'accès à un champ se fait grâce à l'**opérateur de champ** `'.'`. Par exemple `personne.NOM` permet d'accéder au champ `NOM` de la variable structurée `personne`.

### 3.2 Continuation de l'exemple

Le problème.- Nous voulons initialiser notre répertoire. On voudrait un programme qui permette d'initialiser les fiches (avec le nom, le prénom et l'adresse), puis d'afficher le répertoire ainsi obtenu.

Exercice.- Expliquer pourquoi ce peu de manipulations est assez irréaliste. Énumérer d'autres manipulations qui rendraient le cahier des charges plus réaliste.

Précisions.- Nous allons demander le nom, puis le numéro de téléphone. Nous considérerons qu'un nom commençant par `'#'` est une valeur sentinelle de fin d'initialisation. Notre fichier comprendra au plus 100 fiches.

Un programme.- Donnons un programme réalisant ce que nous voulons :

```
/* init.c */

#include <stdio.h>
#include <string.h>

struct ITEM
{
    char NOM[30];
    char TEL[20];
};

struct ITEM FICHIER[100];
int    n;
char   name[30];

void INITIALISER(void)
{
    n = 0;
    printf("Nom : ");
    gets(name);
    while (name[0] != '#')
    {
        strcpy(FICHIER[n].NOM,name);
        printf("Numero de telephone : ");
        gets(FICHIER[n].TEL);
        n++;
    }
}
```

```
        printf("Nom : ");
        gets(name);
    }
}

void AFFICHER(void)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%30s : %20s\n", FICHER[i].NOM,
              FICHER[i].TEL);
}

void main(void)
{
    INITIALISER();
    AFFICHER();
}
```

Remarque.- Ce programme doit fonctionner en continu. Dès qu'on éteint l'ordinateur toutes les données sont perdues. Nous verrons dans le chapitre suivant l'intérêt des fichiers (au sens informatique) permettant de résoudre ce problème.

## 4 Particularités des structures en C

Nous venons de voir comment on met en place en langage C les notions sur les structures que l'on rencontre dans la plupart des langages de programmation (implémentant cette notion). Nous allons voir maintenant quelques particularités du langage C.

### 4.1 Possibilité de l'affectation globale

Dans le langage C (de la norme ANSI seulement) on peut utiliser l'affectation globale du genre :

```
ITEM_1 = ITEM_2;
```

alors que ceci n'est pas possible dans un langage tel que PASCAL.

### 4.2 Initialisation lors de la déclaration

En langage C on peut initialiser une variable structurée lors de sa déclaration, comme pour les tableaux avec des accolades et des virgules. Par exemple :

```
struct PERSONNE
{
    char NOM [30];
    char PRENOM [20];
    int AGE;
};

struct PERSONNE ITEM = {"DUBOIS", "Paul", 30};
```

### 4.3 Type pointeur de structure

Introduction.- Il n'y a pas de notion nouvelle à voir mais le langage C permet, outre la notation classique, une notation souvent utile.

Exemple.- Voulant créer un répertoire on considère des structures comportant des champs pour le nom et le numéro de téléphone :

```
#include <string.h>
struct DONNEE
{
    char NOM [30];
    char TEL [20];
};
struct DONNEE *PTR, PERSONNE;
PTR = &PERSONNE;
strcpy((*PTR).NOM, "Paul");
strcpy((*PTR).TEL, "01 24 34 44 54");
```

Opérateur flèche.- Les parenthèses dans l'accès au champ `(*PTR).NOM` sont obligatoires. Pour ne pas alourdir les notations, on peut utiliser l'*opérateur flèche* équivalent qui donne :

```
strcpy(PTR->NOM, "Paul");
```

la flèche étant tout simplement obtenue, au clavier, par la concaténation des symboles tiret et strictement plus grand.

## 5 Structure comme paramètre de fonction

Introduction.- Une structure peut servir de paramètre à une fonction, avec passage par valeur ou par adresse, et peut être le résultat d'une fonction, sans qu'il n'y ait rien de plus à en dire.

Exercice.- Écrire des fonctions AFFICHE et SAISIE pour la structure précédente.

[ Les déclarations seront :

```
void AFFICHE( struct DONNEE PERSONNE);  
void SAISIE( struct DONNEE *PERSONNE);
```

les appels se faisant sous la forme :

```
AFFICHE(PERSONNE);  
SAISIE(&PERSONNE);
```

de façon habituelle. ]

Exercice.- (**Opérations sur les rationnels**)

Définir un type `rationnel` comportant les deux champs `num` (pour numérateur) et `den` (pour dénominateur) de types entier puis des fonctions permettant la saisie d'un rationnel, son affichage, la somme, la multiplication, la soustraction et la division de deux rationnels.

Les incorporer dans un programme complet.