Chapitre 2

Les types énumérés

Nous avons vu comment utiliser la boucle *pour* et comment indexer les tableaux en utilisant les entiers. On peut généraliser la nature des index en utilisant ce qu'on appelle les **types ordinaux**. Cette possibilité n'est en fait présente que dans peu de langages informatiques. Elle se trouve sous une forme assez sommaire, sous la seule forme des *types énumérés*, en langage C.

2.1 Les types énumérés

2.1.1 Définition des types énumérés

Un **type énuméré** est un type dont on énumère, c'est-à-dire dont on donne la liste effective, les éléments de ce type. On voit immédiatement des cas où cela peut être intéressant, par exemple pour traiter :

- les mois de l'année (janvier, février...) ;
- les notes de musique ;
- les noms de cartes à jouer (as, roi, dame...) ;
- les marques de voiture ;
- les indications d'état civil (célibataire, marié, divorcé...).

Bien entendu cela ne peut concerner que des types correspondants à un ensemble fini d'éléments.

2.1.2 Mise en place des types énumérés en C

Définition.- La définition d'un type énuméré est très simple : on énumère les éléments de ce type, qui sont des identificateurs, séparés par des virgules et encadrés par des accolades ouvrante et fermante. Par exemple :

```
{lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche}
```

ou :

```
{as, roi, dame, valet, dix, neuf, huit, sept}
```

Déclaration d'un type énuméré.- Pour déclarer un tel type, on commence par le mot clé `enum`, suivi d'un identificateur qui sera le nom du type, suivi de la définition proprement dite, suivie d'un point-virgule. Par exemple :

```
enum jour {lundi, mardi, mercredi, jeudi, vendredi,
samedi, dimanche};
```

Déclaration d'une variable d'un type énuméré.- On commence également par le mot clé `enum`, suivi du nom du type, suivi du nom de la variable (ou des variables), suivi d'un point-virgule. Par exemple :

```
enum jour j;
```

Utilisation simple d'une variable d'un type énuméré.- Pour l'instant on ne peut qu'affecter une constante (du type donné) à une variable d'un type énuméré ou comparer une variable du type à une des valeurs du type. Par exemple :

```
j = dimanche;
if (j == lundi) printf("Monday");
```

Conversion des types énumérés.- Le langage C considère les valeurs des types énumérés comme des constantes entières de type `int`, les convertissant dans l'ordre dans lequel elles ont été énumérées lors de la déclaration à partir de 0. Ceci permet alors de travailler comme avec des types ordinaux.

Remarque.- On peut forcer un élément d'un type à prendre une valeur donnée (grâce au signe = suivi d'une constante entière), les éléments suivants étant alors incrémentés à partir de cette valeur. Par exemple si on déclare :

```
enum jour {lundi, mardi, mercredi = 20, jeudi, vendredi, samedi, dimanche};
```

alors `mardi` aura la valeur 1, `mercredi` la valeur 20 et `jeudi` la valeur 21.

2.1.3 Un exemple d'utilisation d'un type énuméré

Un problème de programmation.- Nous voulons relever la température chaque jour d'une certaine semaine et indiquer les températures minimum et maximum de la semaine en question, ainsi que les jours où celles-ci ont été atteintes.

Le problème de la manipulation des jours de la semaine.- On va, à un certain moment, utiliser une variable parcourant l'ensemble des jours de la semaine ou émulant ce parcours.

Une façon de faire est de coder ces jours par des entiers de 1 à 7, par exemple, ce que l'on fait souvent. Un problème avec les codages est qu'ils ne sont pas uniques et qu'il peut y avoir ambiguïté ; par exemple les jours de la semaine vont-ils être énumérés de dimanche à samedi (l'aspect légal en France) ou de lundi à dimanche (l'aspect anglo-saxon maintenant largement répandu).

Exemple.- Le problème ci-dessus peut être résolu grâce au programme ci-dessous :

```
/* releve.c */

#include <stdio.h>

enum jour  {dimanche, lundi, mardi, mercredi, jeudi,
            vendredi, samedi};

void ecris(enum jour j)
     {
     if (j == dimanche) printf("dimanche");
     if (j == lundi) printf("lundi");
     if (j == mardi) printf("mardi");
     if (j == mercredi) printf("mercredi");
     if (j == jeudi) printf("jeudi");
     if (j == vendredi) printf("vendredi");
     if (j == samedi) printf("samedi");
     }

void main(void)
     {
     enum jour j, min, max;
     float temp[7];

/* Saisie des temperatures*/
     printf("Releve des temperatures de la semaine \n");
     for (j = dimanche ; j <= samedi ; j++)
         {
         printf("Quelle est la temperature du ");
         ecris(j);
         printf(" : ");
         scanf("%f", &temp[j]);
```

```
        }

/* Determination des extremes */
     min = dimanche;
     max = dimanche;
     for (j = lundi ; j <= samedi ; j++)
         {
         if (temp[j] < temp[min])  min = j;
         if (temp[j] > temp[max]) max = j;
         }

/* Affichage des resultats */
     printf("\nLa temperature la plus fraiche ");
     printf("de la semaine a ete constatee le ");
     ecris(min);
     printf(". \nElle etait de %3.1f degres C.", temp[min]);
     printf("\nLa temperature la plus elevee ");
     printf("de la semaine a ete constatee le ");
     ecris(max);
     printf(". \nElle etait de %3.1f degres C.\n", temp[max]);
     }
```

Commentaires.- 1^o) On pourrait utiliser le type `jour` sans utiliser de déclaration de type en écrivant :

```
enum {dimanche, lundi, mardi, mercredi,
jeudi, vendredi, samedi} j, min, max ;
```

mais il aurait fallu le répéter pour la déclaration du tableau.

2^o) Il n'existe pas de fonctions d'entrées-sorties pour les types énumérés, d'où la présence de la fonction `ecris()`.

2.1.4 Règles du choix des identificateurs intervenant dans un type énuméré

On a toute latitude sur le choix des identificateurs mais avec les restrictions (de bon sens) suivantes :

- 1^o) Un identificateur ne peut pas être un mot réservé. Ainsi le type concernant les notes de musique ne peut pas être :

  {`do, re, mi, fa, sol, la, si`}

  puisque '`do`' est un mot réservé; on peut, par exemple, remplacer '`do`' par '`doo`'.

- 2^o) Un même identificateur ne peut pas être utilisé pour plusieurs types différents. Ainsi on ne peut pas à la fois utiliser le type des jours de la semaine :

  {`lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche`}

  et le type des jours ouvrables :

  {`lundi, mardi, mercredi, jeudi, vendredi`}

  puisqu'un identificateur tel que '`lundi`' aurait deux types, ce qui n'est pas permis.

– 3^o) Un identificateur n'est pas une constante. On ne peut donc pas considérer, par exemple, le type des chiffres impairs :

{1, 3, 5, 7, 9}

ou le type des voyelles :

{'a', 'e', 'i', 'o', 'u', 'y'}.

2.2 Renommage des types

Notion.- Jusqu'à il n'y a pas longtemps nous n'avions qu'un nombre limité de types avec des identifications très simples (tels que `int`, `float` et `char`) ; la déclaration d'une variable ne prenait pas trop de place. Avec les constructeurs de types que nous avons vus (pointeur, tableau, énumération), cela risque d'être pénible de déclarer une variable. Heureusement nous pouvons donner un nom à un type donné, aussi compliqué soit-il, et utiliser ce nom au lieu de la description complète du type à chaque fois que nous voudrons déclarer une variable ou un argument de fonction.

Syntaxe.- Ceci se fait en langage C grâce au mot clé `typedef`. Cependant la déclaration de certains types spéciaux en C fait qu'il faut distinguer plusieurs cas.

2.2.1 Cas des types simples

Syntaxe.- On écrit :

`typedef` *type nom1, nom2* `;`

où `type` est la description d'un type et `nom1` et `nom2` des identificateurs, comme d'habitude non utilisés pour autre chose. Cette ligne se place au niveau des déclarations de constantes et des variables.

Nous renommons ainsi le type `type` en lui donnant, dans notre cas, deux noms de substitution. Désormais on pourra déclarer une variable de type `type` de la façon suivante :

```
nom1 i ;
```

ou :

```
nom2 i ;
```

Exemple.- Reprenons notre programme qui demande un entier et affiche le double de cet entier. On peut vouloir utiliser que des entiers naturels et donc utiliser le type `entier` au lieu de `unsigned int` trop long à écrire. On écrira donc le programme de la façon suivante :

```
/* type.c */

#include <stdio.h>

typedef unsigned int entier;

void main(void)
     {
     entier i;
     printf("Entrer un entier naturel : ");
     scanf("%d", &i);
     printf("Le double de cet entier est %d.\n", 2*i);
```

```
}
```

2.2.2 Cas des types énumérés

Il y a quelque chose de gênant avec les types énumérés (et nous verrons qu'il y a quelque chose d'analogue avec les types structurés), c'est qu'il faut utiliser le mot clé `enum` lors de la définition d'un tel type mais aussi répéter ce mot clé lors des déclarations de variables de ces types.

Le renommage d'un tel type, par exemple :

```
typedef enum jour day ;
```

si `jour` a déjà été défini, ou directement :

```
typedef enum jour {dimanche, lundi, mardi, mercredi, jeudi,
vendredi, samedi} day ;
```

sinon (le mot `jour` est même superflu dans ce cas) permet d'avoir des déclarations du style habituel :

```
day d ;
```

2.2.3 Cas des types tableau

Dans ce cas le nom du type doit se trouver entre le type des éléments du tableau et la dimension du tableau. On définira par exemple un type `ligne` qui est une chaîne de 80 caractères de la façon suivante :

```
typedef char ligne [81] ;
ligne L1, L2 ;
```

(sans oublier de réserver une place pour l'indicateur de fin de la chaîne de caractères).

2.3 Test et alternative multiples

2.3.1 Exemple introductif

Exemple.- Reconsidérons la procédure `ecris()` du programme `releve.c` pris en exemple lors de l'étude des types énumérés :

```
void ecris(enum jour j)
     {
     if (j == dimanche) printf("dimanche");
     if (j == lundi) printf("lundi");
     if (j == mardi) printf("mardi");
     if (j == mercredi) printf("mercredi");
     if (j == jeudi) printf("jeudi");
     if (j == vendredi) printf("vendredi");
     if (j == samedi) printf("samedi");
     }
```

On utilise sept tests qui, moralement, n'en forment qu'un seul. Ceci peut effectivement se réécrire en une seule instruction, à savoir un *test multiple*. Ceci donnera :

```
void ecris(enum jour j)
     {
     switch(j)
         {
         case dimanche : printf("dimanche"); break;
         case lundi : printf("lundi"); break;
         case mardi : printf("mardi"); break;
         case mercredi : printf("mercredi"); break;
         case jeudi : printf("jeudi"); break;
         case vendredi : printf("vendredi"); break;
         case samedi : printf("samedi"); break;
         }
     }
```

Le corps de la fonction ne comprend plus qu'une seule instruction, une instruction de test multiple.

Exercice 1.- *Remplacer la version précédente de la fonction* `ecris()` *dans le programme complet et faite-le exécuter.*

Exercice 2.- *Enlever une instruction* `break` *du programme précédent, par exemple le premier, et faite-le exécuter. Que se passe-t-il?*

2.3.2 Syntaxe

Syntaxe.- Nous avons deux formes suivant que l'on utilise `default` ou non :

`switch(`*expression*`)`
{
`case` *const_1* `:` *instruction_1* `; break ;`
`case` *const_2* `:` *instruction_2* `; break ;`
- - - - - - - - - - - - - - - - - -
`case` *const_n* `:` *instruction_n* `; break ;`
}

ou :

`switch(`*expression*`)`
{
`case` *const_1* `:` *instruction_1* `; break ;`
`case` *const_2* `:` *instruction_2* `; break ;`
- - - - - - - - - - - - - - - - - -
`case` *const_n* `:` *instruction_n* `; break ;`
`default :` *instruction* `;`
}

où `expression` est une expression d'un type entier (y compris d'un type énuméré), `const_1`, ... , `const_n` des constantes, toutes différentes, de ce type et `instruction_1`, ... , `instruction_n`, `instruction` des instructions.

Sémantique.- Dans le premier cas si la valeur de l'expression est égale à la constante de numéro i alors l'instruction de numéro i est exécutée. Si elle n'est égale à aucune de ces constantes alors l'instruction vide est exécutée.

Dans le second cas si la valeur de l'expression est égale à la constante de numéro i alors l'instruction de numéro i est exécutée. Si elle n'est égale à aucune de ces constantes alors l'instruction `instruction` est exécutée.

2.3.3 Rôle de l'instruction `break`

Un autre exemple.- Le programme C suivant demande de taper une voyelle (sans signe diacritique) et indique s'il s'agit d'une voyelle minuscule, d'une voyelle majuscule ou d'un autre symbole :

```
/* voyelle.c */

#include <stdio.h>

void main(void)
     {
     char C;
     printf("Entrer un caractere au clavier : ");
     scanf("%c", &C);
     printf("Ce caractere ");
     switch(C)
        {
        case 'a': case 'e': case 'i': case 'o': case 'u':
           printf("est une voyelle minuscule.\n");
           break;
        case 'A': case 'E': case 'I': case 'O': case 'U':
           printf("est une voyelle majuscule.\n");
           break;
        default: printf("n\'est pas une voyelle.\n");
        }
     }
```

Remarque.- On remarquera la façon de faire lorsque à plusieurs constantes correspond la même instruction : on écrit cette instruction après les `case` correspondant. Ceci explique la nécessité de l'instruction `break` : si on ne la met pas alors les instructions des cas suivants sont exécutées.