Les types utilisateurs (C)
Corrigé

Résumé

Ce document décrit comment traduire en C les types utilisateur du langage algorithmique.

Table des matières

Liste des exercices

1 Types énumérés

Les types énumérés en C se définissent en utilisant le mot-clé **enum**. Voici un petit programme qui manipule un type énuméré pour les jours de la semaine :

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  enum Jour {LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE };
5
6  int main()
7  {
8      enum Jour un_jour;
9      un_jour = LUNDI;
10     if (un_jour == MARDI) {
11         printf("C'est␣mardi␣!\n");
12     }
13     else {
14         printf("Ce␣n'est␣pas␣mardi␣!\n");
15     }
16
17     return EXIT_SUCCESS;
18 }
```

Remarque : Le type défini n'est pas `Jour` mais **enum** `Jour` qui n'est pas forcément très pratique à écrire. On peut utliser le mot-clé **typedef** pour définir le type Jour.

```
1  typedef enum Jour Jour;
2              /* définition du type Jour équivalent à enum Jour */
3  ...
4  {
5      Jour j;
6      j = LUNDI;
7      ...
8  }
```

typedef permet de définir un nouveau type. Le principe est de faire comme si on définissait une variable du type qui nous intéresse mais, pour le nom de la variable, on donne le nom du type que l'on veut définir et on place un **typedef** devant la déclaration.

```
1          int i;          /* déclaration d'une variable i de type Entier */
2  typedef int Entier      /* définition du type Entier comme étant int */
```

Représentation interne : En interne, le compilateur considère le type enuméré comme un type entier en donnant 0 à la première valeur, 1 à la deuxième, etc. Ainsi, si on essaie d'afficher une valeur d'un type énuméré, c'est cette valeur entière qui sera afficher (sauf à écrire une fonction de transcription).
Attention : En C, les types énumérés sont compatibles entre eux.

2 Les tableaux

2.1 Motivation

2.2 Définition

2.3 Tableaux à une dimension

Les tableaux en C sont beaucoup plus contraignants qu'en algorithme car le type des indices est forcément entier et le premier indice valide est nécessairement 0. Aussi, pour déclarer un tableau en C, il suffit de donner le type des éléments et la capacité.

```
1 double vect[10];        /* vect est un tableau de 10 réels */
```

Les indices valides sont donc 0, 1, 2, ..., 8 et 9.

C'est équivalent à la déclaration suivante en algorithme :

```
1 Variable
2     vect: Tableau [0..9] De Réel
```

Il est possible de donner un nom au type en utilisant **typedef** :

```
1 typedef double Vecteur10[10];   /* Définition du type Vecteur10 */
2 Vecteur10 vect; /* déclaration d'une variable de type Vecteur10 */
```

Remarque : Il est fortement conseillé d'utiliser une constante symbolique à la place de la constante littérale 10.

Accès à un élément : L'accès à un élément se fait comme en algorihtmique en utilisant les crochets :

```
1     vect[0] = 10;
2     vect[1] = vect[0] + 1;
3     vect[0]++;
```

Attention : Aucun contrôle n'est fait sur la validité des indices utilisés. Des indices incorrects correspondent à un programme faux. Malheureusement, ceci ne provoque pas nécessairement un plantage du programme. Ceci rend ces erreurs difficiles à identifier. Il faut donc être vigilant : à chaque fois que vous accéder à un élément d'un tableau, demandez-vous si l'indice est valide.

2.4 Exemples

Exercice 1 : Initialiser un tableau

Initialiser astucieusement un tableau de 10 entiers pour qu'il contienne les valeurs suivantes :

1	2	3	4	5	6	7	8	9	10

Solution : Si on considère un tableau définit par :

```
1 Type
2     Vecteur = Tableau [1..10] De Entier
```

On constate que la valeur de la case d'indice i est justement i. Pour initialiser un vecteur, il suffit donc de parcourir les entiers de 1 à 10 pour remplir la case correspondante.

```
Variable
    tab: Vecteur
    indice: Entier
Début
    Pour indice <- 1 JusquÀ indice = 10 Faire
        tab[indice] <- indice
    FinPour
Fin
```

Le principe est le même en C. Cependant, les indices en C commencent nécessairement à 0. Il faut donc ajuster les bornes de variation de i et initialiser une case avec $i + 1$.

```c
/*****************************************************************
 *  Auteur   : Xavier Crégut <cregut@enseeiht.fr>
 *  Version  : 1.1
 *  Objectif : Initialiser un tableau avec dans l'ordre les valeurs :
 *                       1, 2, 3, 4, 5, 6, 7, 8, 9 et 10.
 *****************************************************************/

#include <stdio.h>
#include <stdlib.h>

int main()
{
    int tab[10];        /* le tableau à initialiser */
    int indice;         /* parcourir les entiers de 0 à 9 */

    for (indice = 0; indice < 10; indice++) {
        tab[indice] = indice + 1;
    }

    return EXIT_SUCCESS;
}
```

Exercice 2 : Afficher un tableau d'entier

Écrire un programme qui affiche tous les éléments d'un tableau de capacité MAX (égale à 10) mais dont la taille effective (c'est-à-dire le nombre d'éléments utiles) est donné par la variable `nb`.

Les éléments du tableau seront écrits entre crochets, dans l'ordre croissant de leur indice et séparés par des points-virgules. Voici quelques exemples :

```
[]              -- un tableau vide (de taille effective nulle)
[ 1; 2; 3]      -- tableau contenant les 3 valeurs 1, 2 et 3.
[ 421 ]         -- tableau contenant la seule valeur 421
```

Solution : On constate que l'on ne peut pas afficher directement chaque élément du tableau car le point-virgule ne doit pas être mis après le dernier élément. le principe est donc d'afficher d'abord le premier élément puis, pour tous les éléments suivants, d'afficher le point-virgule séparateur et l'élément lui-même. Ceci ne peut bien sûr être fait que si le tableau contient au moins un élément.

```c
/*****************************************************************
 *  Auteur   : Xavier Crégut <cregut@enseeiht.fr>
```

```c
 *  Version  : 1.2
 *  Objectif : Afficher un tableau d'entiers tab de taille effective nb.
 ******************************************************************/

#include <stdio.h>
#include <stdlib.h>

#define MAX 10  /* Capacité du tableau */

int main()
{
    int tab[MAX] = { 4, 2, 1 }; /* le tableau à afficher */
            /* manière d'initialiser un tableau en énumérant ses
             * valeurs.
             * Remarque : On pourrait ne pas préciser la capacité MAX.
             * La capacité du tableau serait alors automatiquement
             * mise à 3, nb d'éléments dans les accolades.
             */
    int nb = 3;         /* taille effective de tab */

    /* Afficher le tableau tab de taille nb */
    printf("[");
    if (nb > 0) {       /*{ le tableau n'est pas vide }*/
        int i;  /* pour parcourir les indices du tableau */

        /* afficher le premier élément */
        printf("%i", tab[0]);

        /* afficher les autres éléments */
        for (i=1; i < nb; i++) {
            printf(", %i", tab[i]);
        }
    }
    printf("]");

    return EXIT_SUCCESS;
}
```

2.5 Chaînes de caractères

Les chaînes de caractères sont tout simplement des tableaux de caractères avec comme spécificité que le caractère de code ASCII 0 (`'\0'`) marque la fin de la chaîne. On parle de chaîne de caractères à zéro terminal.

En conséquence, pour représenter une chaîne de 5 caractères, il faut en fait utiliser un tableau de capacité de 6 pour avoir de la place pour le caractère nul en plus des 5 caractères.

Puisque les chaînes sont avant tout des tableaux, toutes les propriétés des tableaux s'appliquent aux chaînes de caractères.

Dans le module `<string.h>` sont définies des opérations pour manipuler les chaînes de caractères :

- `strlen(s1)` : la longueur de s1 (nombre de caractères)
- `strcpy(destination, source)` : initialise la chaîne `destination` à partir de `source`.
  Attention : La chaîne `destination` doit avoir une capacité strictement supérieure à la longueur de `source`.
- `strcat(destination, source)` : ajoute la chaîne `source` à la fin de la chaîne `destination` (concaténation).
  Attention : La chaîne `destination` doit avoir une capacité suffisante :

```
strlen(destination) + strlen(source) + 1 <= capacité de destination
```

- `strcmp(s1, s2)` : renvoie un entier positif si `s1 < s2`, nul si `s1 = s2` et positif si `s1 > s2`.

2.6 Tableaux à plusieurs dimensions

Un tableau à plusieurs dimensions se déclare en ajoutant une paire de crochets et une capacité par dimension. L'accès à un élément se fait par ajout d'un indice entre crochets par dimension.

```
double m[10][20];       /* m est une matrice de réels */
m[0][0];        /* élément à la ligne 0, colonne 0 */
m[10][20];      /* élément à la ligne 10, colonne 20 */
```

Remarque : On peut donner un nom au type matrice en utilisant **`typedef`** :

```
typedef double Matrice[10][20];
        /* définition du type Matrice, tableau de 10x20 de réels */

Matrice m1, m2, m3;     /* 3 matrices ! */
int ligne, colonne;

/* Faire m3 = m1 + m2 */
for (ligne = 0; ligne < 10; ligne++) {
    for (colonne = 0; colonne < 20; colonne++) {
        m3[ligne][colonne] = m1[ligne][colonne] + m2[ligne][colonne];
    }
}
```

3 Les enregistrements

Le principe est strictement le même qu'en algorithmique. C'est le mot-clé **`struct`** qui permet de définir un type énuméré.

```
1  struct Date {
2      int jour;   /* numéro du jour dans le mois */
3      int mois;   /* numéro du mois dans l'année */
4      int annee;  /* numéro de l'année */
5  };
```

C'est le point qui permet d'accéder à un champs d'un enregistrement comme le montre l'exemple suivant.

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  struct Date {
5      int jour;   /* numéro du jour dans le mois */
6      int mois;   /* numéro du mois dans l'année */
7      int annee;  /* numéro de l'année */
8  };
9
10 typedef struct Date Date;
11
12 int main()
13 {
14     Date d1, d2;        /* deux dates */
15     int annee;
16
17     /* initialiser la date d1 */
18     d1.jour = 23;
19     d1.mois = 11;
20     d1.annee = 2000;
21
22     /* initialiser la date d2 à partir de d1 */
23     d2 = d1;
24
25     /* afficher la date d2 */
26     printf("%d/%d/%d\n", d2.jour, d2.mois, d2.annee);
27
28     /* conserver l'année de d2 */
29     annee = d2.annee;
30
31     /* saisir l'année de d2 */
32     scanf("%d", &d2.annee);
33
34     /* afficher la date d2 */
35     printf("%d/%d/%d\n", d2.jour, d2.mois, d2.annee);
36
37     return EXIT_SUCCESS;
38 }
```