Les types utilisateurs
Corrigé

Résumé

Ce document décrit les constructions présentes dans le pseudo-langage algorithmique et la langage C pour permettre à l'utilisateur de définir ses propres types, à savoir les types énumérés, les types tableaux et les types enregistrements.

Table des matières

Liste des exercices

1 Types énumérés

1.1 Motivation

Parfois, dans un programme on est obligé de prendre des conventions (des codages) pour représenter certaines informations. Par exemple, pour représenter un mois de l'année on peut décider de prendre des entiers avec 1 pour janvier, 2 pour février, etc. Ce choix est relativement naturel car il est communément admis et utilisé dans la vie courante.

Malheureusement, il n'y a pas toujours concensus sur la manière de représenter une information. Par exemple comment coder les 7 jours de la semaine. Est-ce que l'on prend les entiers et 1 à 7 ou de 0 à 6 ? Le premier numéro (1 ou 0) représente-t-il lundi, dimanche, samedi ou un autre jour ? Il est alors difficile de faire un choix qui soit significatif pour tout le monde.

De la même manière comment représenter des couleurs, les différents états possible d'un système (marche, arrêt, interrompu, etc.).

Dans tous les cas, comprendre la signification des constantes littérales qui se trouvent dans un programme devient alors délicat. Par exemple que représente le chiffre 5 ? Est-ce le mois de mai ? La couleur violet ? Le vendredi ? Tout simplement la valeur 5 ? Autre chose ?

1.2 Définition

Le principe d'un type énuméré est de donner un nom symbolique pour chacune des valeur que peut prendre une variable.

Définition : Les types énumérés permettent à l'utilisateur de définir son propre type en indiquant explicitement l'ensemble de ses valeurs.

```
Type
    Mois = (JANVIER, FÉVRIER, ..., DÉCEMBRE)
        -- Attention : une machine ne comprend pas les « ... ».  Il faut
        -- donc lister explicitement toutes les valeurs possibles.

    Jour = (LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE)

    Couleur = (ROUGE, JAUNE, VERT, BLEU, VIOLET)        -- quelques couleurs
```

C'est le compilateur qui aura la charge de définir un codage pour ces noms symboliques et non le programmeur. Le programme sera plus lisible car le programmeur utilisera le nom symbolique pour désigner une valeur particulière du type :

```
Variable
    m: Mois
    c: Couleur
Début
    m ← MAI
    c ← VIOLET
    m ← c        -- interdit car de types différents !
```

Relation d'ordre : L'ordre dans lequel les valeurs sont listées a une importance (relative) puisqu'on considère qu'il y a une relation d'ordre entre les éléments : ils sont listés du plus petit au plus grand.

```
JANVIER < FÉVRIER < ... < DÉCEMBRE
```

Remarque : Les booléens sont un exemple de type énuméré avec deux valeurs `FAUX` et `VRAI`.

```
Type
    Booléen = (FAUX, VRAI)
```

En C. 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 :

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

enum Jour {LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE };

int main()
{
    enum Jour un_jour;
    un_jour = LUNDI;
    if (un_jour == MARDI) {
        printf("C'est mardi !\n");
    }
    else {
        printf("Ce n'est pas mardi !\n");
    }

    return EXIT_SUCCESS;
}
```

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.

```
typedef enum Jour Jour;
            /* définition du type Jour équivalent à enum Jour */
...
{
    Jour j;
    j = LUNDI;
    ...
}
```

`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.

```
        int i;          /* déclaration d'une variable i de type Entier */
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.

1.3 Types énumérés et constantes

Si on ne disposait pas du type énuméré, on pourrait le simuler en s'appuyant sur les constantes. Ainsi pour définir le type `Mois`, on pourrait faire :

```
Constante
    JANVIER = 1
    FÉVRIER = 2
    ...         -- il faut bien sûr TOUTES les définir !
    DÉCEMBRE = 12

Type
    Mois = Entier       -- ou mieux JANVIER..DÉCEMBRE
```

Cette solution a bien sûr des inconvénients :

– il faut choisir un codage ;
– c'est long à écrire ;
– il faut faire attention de définir une valeur différente pour chacune des constantes ;
– il faut faire bien attention dans le texte à utiliser le nom de la constante et non sa valeur ;
– tous les types sont des entiers pour le compilateur, on pourra donc comparer des mois et des couleurs, les additionner, etc.

2 Les tableaux

2.1 Motivation

2.1.1 Les tableaux comme une facilité d'écriture

Jusqu'à présent, nous avions besoin de déclarer autant de variables que de données à manipuler. Intéressons nous au nom que nous donnons à ces variables. Dans le cas où nous avons deux données entières, nous pouvons les appeler « a » et « b ». Si le nombre de données est plus important, nous serons rapidement en mal d'imagination pour trouver des noms, nous opterons alors certainement pour un préfixe commun et numéro permettant de distinguer les différentes variables. Par exemples, si nous devons utiliser quatre variables entières, nous les appellerons « n1 », « n2 », « n3 » et « n4 ».

Intéressons nous au petit problème suivant.

Exercice 1 : Occurrences des chiffres d'un entier
Intéressons nous aux chiffres qui constituent un nombre.

1.1 Écrire un programme qui compte le nombre d'occurrences des 10 chiffres dans un entier donné.

Par exemple, l'entier 4214 a une occurrence du chiffre 1, une de 2 et deux de 4.

Solution :

En C.

```c
/*********************************************************************
 *  Auteur    : Xavier Crégut <cregut@enseeiht.fr>
 *  Version   : 1.2
 *  Objectif  : Compter le nb d'occurrences des chiffres d'un nombre
 *                ATTENTION, EXEMPLE À NE PAS SUIVRE !!!!!
 *********************************************************************/

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

int main()
{
    int nombre;         /* entier saisi par l'utilisateur */
    int nb0;    /* nb d'occurrences du chiffre 0 dans nombre */
    int nb1;    /*      ''              ''     1   ''     ''  */
    int nb2;    /*      ''              ''     2   ''     ''  */
    int nb3;    /*      ''              ''     3   ''     ''  */
    int nb4;    /*      ''              ''     4   ''     ''  */
    int nb5;    /*      ''              ''     5   ''     ''  */
    int nb6;    /*      ''              ''     6   ''     ''  */
    int nb7;    /*      ''              ''     7   ''     ''  */
    int nb8;    /*      ''              ''     8   ''     ''  */
    int nb9;    /*      ''              ''     9   ''     ''  */

    /* saisir le nombre */
    printf("Nombre : ");
```

```
    scanf("%d", &nombre);

    /* initialiser les compteurs */
    nb0 = 0;
    nb1 = 0;
    nb2 = 0;
    nb3 = 0;
    nb4 = 0;
    nb5 = 0;
    nb6 = 0;
    nb7 = 0;
    nb8 = 0;
    nb9 = 0;

    /* comptabiliser chaque chiffre de nombre */
    do {
        int chiffre;    /* chiffre unité de nombre */

        /* comptabiliser l'unité de nombre */
        chiffre = nombre % 10;
        switch (chiffre) {
            case 0: nb0++; break;
            case 1: nb1++; break;
            case 2: nb2++; break;
            case 3: nb3++; break;
            case 4: nb4++; break;
            case 5: nb5++; break;
            case 6: nb6++; break;
            case 7: nb7++; break;
            case 8: nb8++; break;
            case 9: nb9++; break;
        }

        /* supprimer l'unité de nombre */
        nombre = nombre / 10;

    } while (nombre > 0);

    /* afficher les occurrences */
    printf("nb de 0 : %d\n", nb0);
    printf("nb de 1 : %d\n", nb1);
    printf("nb de 2 : %d\n", nb2);
    printf("nb de 3 : %d\n", nb3);
    printf("nb de 4 : %d\n", nb4);
    printf("nb de 5 : %d\n", nb5);
    printf("nb de 6 : %d\n", nb6);
    printf("nb de 7 : %d\n", nb7);
    printf("nb de 8 : %d\n", nb8);
    printf("nb de 9 : %d\n", nb9);

    return EXIT_SUCCESS;
```

```
}
```

1.2 Comment modifier le programme pour que seuls les chiffres dont le nombre d'occurrences est non nul soient affichés ?

Solution : Il suffit d'ajouter une structure de contrôle **Si** devant chacun des affichages des nombres d'occurrences. C'est faisable mais c'est fastidieux et source d'erreurs.

On peut bien sûr faire du copier/coller mais c'est toujours dangereux d'introduire de la redondance.

1.3 Comment modifier le programme pour que les nombres d'occurrences soient affichés du plus grand vers le plus petit ?

Solution : C'est beaucoup, beaucoup plus difficile. Ce n'est pas infaisable mais bon courrage !

Même s'il est possible de résoudre cet exercice, on constate qu'il serait pratique de pouvoir manipuler de manière uniforme, par exemple en les désignant par leur numéro. Par exemple, pour initialiser les compteurs à zéro il serait plus facile et plus expressif de pouvoir faire :

```
Pour i ← 0 JusquÀ i = 9 Faire
    nb_i ←0
FinPour
```

où nb_i désigne le compteur du nombre d'occurrences du chiffre i.

C'est ce que nous permettra de faire la notion de tableau...

2.1.2 Les tableaux comme une nécessité pour la modélisation des données

Jusqu'à présent, nous avons uniquement utilisé des variables simples qui correspondaient à un unique emplacement en mémoire d'un type particulier. Cependant, il est fréquent de devoir manipuler un grand nombre (par forcément très grand d'ailleurs) de données du même type auxquelles on applique les mêmes traitements.

Par exemple, on peut considérer un ensemble de factures reçues par une entreprise. Pour les représenter, il existe un type de données particulier : le tableau, en l'occurrence, un tableau de facture. Ceci permet de regrouper sous un même nom l'ensemble des factures de l'entreprise. Une facture particulière sera obtenue en utilisant un indice.

Remarquons qu'il faudra certainement commencer par les ordonnées en fonction de leur date d'échéance. Une partie importante de l'informatique et le tri et le classement de l'information.

Exercice 2 : Représentation de l'ensemble des factures

Est-il envisageable de définir autant de variables que de factures à traiter ?

Solution : La solution est évidemment non car on ne connaît pas à priori le nombre de factures. De plus, les trier serait alors pour le moins fastidieux !

2.2 Définition

Un tableau est une variable qui permet de rassembler sous un même nom (celui de la variable) un nombre fini d'éléments ayant **tous le même type**. Un élément particulier d'un tableau est désigné en précisant son **indice**. On parle alors de tableau à une dimension.

Il est peut être nécessaire de préciser plusieurs indices pour accéder à une donnée. On parle alors de tableau à plusieurs dimensions.

Le type qui caractérise une telle variable est également appelé tableau.

Lorsqu'un tableau est défini, le nombre d'éléments qu'il peut contenir doit être précisé (en général au travers de la spécification des valeurs possibles des indices) et il ne pourra pas être changé par la suite. Ce nombre d'éléments est appelé la **capacité** du tableau.

2.3 Tableaux à une dimension

Un tableau à une dimension est un tableau dont on accède aux éléments (pour « lire » ou modifier leur valeur) en utilisant un seul indice (ou index). Le type des indices est généralement un intervalle sur un type scalaire[1] (ou directement un type scalaire). Le type des indices est précisé entre crochets.

Un tableau est donc caractérisé par :
- le type des éléments qu'il contient ;
- les indices (ou index) valides pour accéder à ces éléments.

Définition d'un type tableau : La forme générale de définition d'un type tableau est la suivante :

```
T1 = Tableau [Type_Indice] De Type_Élément;
```

Exemple :

```
Type
    Jour = (LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE)
    Vecteur10 = Tableau [1..10] De Réel
```

Déclaration d'une variable tableau : Les variables de type tableau se déclarent comme les autres variables.

```
Variable
    tab: T1;    -- difficile de donner un nom plus significatif !
```

Opérateurs : Un seul opérateur est défini sur les tableaux. Il s'agit de l'opérateur d'indiçage qui permet de sélectionner un élément du tableau en fonction de son indice. Cet élément se comporte exactement comme une variable de même type que les éléments du tableau.

```
tab[expression]
```

« tab » est une variable de type tableau et « expression » est une expression dont le type est compatible avec celui des indices du tableau.

Exemple : Avec un tableau de réel.

```
Variable
    vect : Vecteur10
    i: Entier;

    tab_bool : Tableau [Booléen] De Réel
    nom_jours : Tableau [Jour] De Chaîne
```

[1]On appelle type scalaire les entiers, les caractères, les booléens et les types énumérés.

```
Début
    i ← 5
    vect[1] ← 1 -- OK   vect[1] = 1
    Lire(vect[2])       -- OK   vect[2] = valeur saisie
    vect[i] ← 3 -- OK   vect[5] = 2
    vect[i+2] ← 4       -- OK   vect[7] = 7
    vect[3] ← vect[5]   -- OK   vect[3] = 2
    vect[-1] ← 5        -- Erreur : -1 ∉ 1..10 !
    vect[11] ← 6        -- Erreur : 11 ∉ 1..10 !

    Pour i ← 1 JusquÀ i = 10 Faire
        vect[i] ← vect[i] + i
    FinPour
Fin
```

Et avec un tableau dont le type des indices n'est pas sur les entiers.

```
Variable
    tab_bool : Tableau [Booléen] De Réel
    nom_jours : Tableau [Jour] De Chaîne
Début
    tab_bool[FAUX] ← 10.0
    tab_bool[VRAI] ← 20.0

    nom_jours[LUNDI] ← "lundi"
    ...
    nom_jours[DIMANCHE] ← "dimanche"
Fin
```

Prenons un exemple concret. On considère des candidats inscrits à une formation diplômante. Si la note obtenue à cette formation est supérieure ou égale à 10 alors le candidat est reçu.

```
Constante
    MAX = 10
Variable
    noms : Tableau [1..MAX] De Chaîne
    notes : Tableau [1..MAX] De Réel
Début
    -- saisir les noms des candidats
    Pour i ← 1 JusquÀ i = MAX Faire
        Écrire("nom␣du␣", i, "ième␣candidat␣:␣")
        Lire(noms[i])
    FinPour

    -- saisir les notes des candidats
    Pour i ← 1 JusquÀ i = MAX Faire
        Écrire("Note␣de␣", noms[i], "␣:␣")
        Lire(notes[i])
    FinPour

    -- afficher les reçus
    Pour i ← 1 JusquÀ i = MAX Faire
```

```
        Si notes[i] >= 10 Alors
            Écrire(noms[i], "␣reçu")
        FinSi
    FinPour
Fin
```

Remarque : Il est important de noter l'utilisation de la constante. On peut ainsi facilement modifier la dimension des différents tableaux et les algorithmes les manipulant en ne modifiant qu'une ligne du programme (la définition de la constante). Si on avait mis directement 10, il serait difficile de changer la dimension (15 par exemple). En effet, il faudrait changer les valeurs de 10 qui correspondent à la taille et seulement celles qui correspondent à la taille (pas le 10 qui correspond à la moyenne). Il peut également y avoir d'autres données qui dépendent de la taille telles que 20 (2*MAX) ou 11 (MAX+1), etc. **La conclusion est donc : utilisez les constantes !**

Capacité et taille effective. La **capacité** d'un tableau est le nombre d'éléments que peut contenir le tableau. Puisque cette capacité ne peut pas être changée au cours de l'exécution d'un algorithme, il est nécessaire de prendre une valeur suffisamment grande. Il faut alors trouver un majorant du nombre d'éléments. Par exemple, pour une promotion de stagiaires, on peut prendre la valeur de 24. Puisque c'est un majorant, c'est donc qu'il y aura généralement moins de stagiaires. On utilise donc une variable qui compte le nombre effectif d'éléments stockés dans le tableau. C'est la **taille effective**.
Attention : Lors de l'accès d'un élément d'un tableau, l'indice fourni doit respecter la contrainte de type définie pour les indices (avec certains langages/environnements, ce non respect peut être détecté à l'exécution du programme et provoquer une erreur d'exécution). Cependant, si seule une partie est réellement utilisée, alors il faut s'assurer que les indices utilisés sont bien dans cette partie utile. Malheureusement, pour vérifier ce point, le compilateur et plus généralement l'environnement de développement ne peuvent pas vous aider. C'est donc à vous d'être vigilant !

En C. 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.

```
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 :

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

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

```
typedef double Vecteur10[10];   /* Définition du type Vecteur10 */
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 :

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

```
Type
    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
```

En C. 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 $i + 1$.

```
/*********************************************************************
 *  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 4 : Afficher un tableau d'entier

Écrire un sous-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.

En C.

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

Attention : Le type chaîne de caractères et les opérations associées sont très dépendant du langage de programmation considéré.

Une chaîne de caractères peut être considérée comme un tableau de caractères avec des opérations supplémentaires. Le premier opérateur est « longueur ». C'est une fonction qui permet d'obtenir le nombre de caractères de la chaîne de caractères.

Par définition les indices valides pour une chaîne de caractères sont des entiers compris entre 1 et la longueur de la chaîne. Chaque caractère peut être obtenu (en accès ou en modification) en utilisant son indice pour le sélectionner.

Un deuxième opérateur défini sur les chaînes de caractères est l'opérateur de concaténation noté « + ».

```
Variable
    ch1, ch2: Chaîne;
Début
    ch1 ← "Chaîne 1";
    ÉcrireLn("Premier caractère : ", ch1[1])
    ÉcrireLn("Dernier caractère : ", ch1[Longueur(ch1)])
    ch2 ← ch1 + ' ' + ch1;
    ÉcrireLn(ch1, " --> ", Longueur(ch1))
    ÉcrireLn(ch2, " --> ", Longueur(ch2))
Fin
```

En C. 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

Les tableaux peuvent avoir plusieurs dimensions. Il suffit de préciser les types qui correspondent aux nouvelles dimensions. Par exemple, on peut définir une matrice comme étant un tableau à deux dimensions, la première correspondant aux lignes et la seconde aux colonnes.

```
Constante
    NB_LIGNES = 5
    NB_COLONNES = 10
Type
    Matrice = Tableau [1..NB_LIGNES, 1..NB_COLONNES] De Réel
Variable
    m1, m2: Matrice;
```

Pour accéder à un élément de ce tableau, il suffit de donner une valeur pour chaque indice, par exemple m1[1,1] ou m1[5,10].

Les différents indices d'un tableau ne sont pas nécessairement de même type.

Remarque : On peut définir un tableau à deux dimensions comme un tableau de tableau.

```
Type
    Matrice = Tableau [1..NB_LIGNES] De Tableau [1..NB_COLONNES] De Réel
```

Dans ce cas, pour accéder à un élément, on écrira : m1[ligne][colonne].

Les deux types Matrice définis sont isomorphes. On préféra cependant la première version (sauf si on veut insister sur la notion de Vecteur et dire qu'une matrice est un tableau de vecteurs).

Remarque : Un tableau peut avoir un nombre quelconque de dimensions. Il est donc possible de définir des tableaux à 3, 4, etc. dimensions. Le nombre de dimensions est nécessairement fixe et ne peut pas changer.

$\boxed{\text{En C.}}$ 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.

```c
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 enregitrements

3.1 Motivation

Un type enregistrement permet de définir des types structurés (ou complexes) qui ne peuvent pas être représentés par les types élémentaires que nous avons déjà vus (Entier, Booléen, Réel, etc.), ni par un tableau.

Par exemple, comment définir une date ? Ce n'est pas un seul entier mais trois entiers correspondant respectivement au numéro du jour dans le mois, au numéro du mois dans l'année, et à l'année.

Une date peut donc être caractérisée par trois valeurs :
- la première correspondant au numéro du jour dans le mois ;
- la deuxième correspondant au numéro du mois dans l'année ;
- la troisième correspondant à l'année.

La valeur d'une date est par exemple (19, 11, 2000) pour le 19 novembre 2000.

Les différentes valeurs sont de natures différentes (même si elles sont de même type dans le cas présent). Elle ne peuvent donc pas être rangées (logiquement) dans un tableau.

Exemples : Voici quelques exemples d'enregistrements :
- Un **complexe** est défini par une partie réelle et une partie imaginaire.
- Une **date** est composée d'un numéro de jour (1..31), d'un numéro de mois (1..12) et d'une année (strictement positive).
- Un **stage** peut être défini comme un intitulé (une chaîne de caractères), une date de début et une date de fin (deux dates), un nombre de places (entier).
- Une **fiche bibliographique** est définie par le titre du livre (Chaîne), les auteurs (noms et prénoms), la date de parution, l'éditeur, le numéro ISBN (Chaîne), etc.

3.2 Définition d'un type enregistrement

Définition : Un enregistrement est un type T qui est le produit cartésien de plusieurs types T_1, T_2, ..., T_n. À chaque composante de type T_i est associé un identificateur choisi par le programmeur. Il permet de sélectionner la composante. Le couple (identificateur, Type) est appelé un *champ* de l'enregistrement.

```
Types
    T_Enregistrement =  -- Le nom du type
        Enregistrement  -- sa définition
            nom_champ1: T1              -- un champ, son type et son rôle
            ...
            nom_champn: Tn
        FinEnregistrement
```

Par exemple, le type correspondant à une date peut être décrit ainsi :

```
Date =
    Enregistrement
        jour: 1..31             -- Le numéro du jour
        mois: 1..12             -- le numéro du mois : 1 janvier...
```

```
        année: Entier   -- année > 0
    FinEnregistrement
```

Intuitivement, un enregistrement permet donc de regrouper dans une même structure un ensemble d'informations ayant entre elles un lien logique.
Remarque : Un enregistrement correspond à un produit cartésien. Par exemple, un complexe est le produit cartésien des réels avec les réels (IR^2).
Intérêt : Les enregistrements permettent de manipuler simultanément un ensemble de données logiquement reliées. Par exemple, pour avoir deux dates, il suffit de déclarer deux variables d1 et d2 du type Date plutôt que j1, m1, a1 et j2, m2, a2 de type Entier.

3.3 Déclaration d'une variable de type enregistrement

Un type enregistrement est avant tout un type. Il permet donc de déclarer des variables de ce type.

```
Variables
    d1: Date  -- d1 est une variable de type Date.
              -- Elle occupe 3 entiers en mémoire.
```

Une valeur du type enregistrement T est un n-uplet $(v_1, v_2, ..., v_n)$ où chaque valeur v_i est du type T_i. Ainsi, la place occupée en mémoire est la somme des places nécessaires pour représenter chacun des champs de types T_i.

$$Place(T) = Place(T_1) + ... + Place(T_n)$$

$$\text{d1.}\left\{\begin{array}{l|c|l} \text{jour} & ? & 1..31 \\ \text{mois} & ? & 1..12 \\ \text{année} & ? & \text{Entier} \end{array}\right.$$

3.4 Manipulation d'une variable de type enregistrement

Sur les variables de type enregistrement, un seul opérateur existe l'affectation. Il consiste à copier toutes les composantes de l'enregistrement.

Tous les autres traitements doivent être explicitement réalisés par le programmeur. Pour ce faire, il peut accéder individuellement à chaque composante grâce à son identifiant. La composante (le champ) se comporte alors exactement comme tout autre variable du même type que cette composante.
Remarque : Il est conseillé d'écrire des sous-programmes réalisant les opérations usuelles sur le type enregistrement : voir sous-programmes et types abstraits de données.

Par exemple, si `d` est une variable de type `Date`, on peut accéder au jour de cette date en écrivant `d.jour`. On peut alors l'écrire, le lire, le comparer... et plus généralement lui appliquer toutes les opérations définies sur le type 1..31.
Attention : Lorsqu'on utilise « . », il est nécessaire que l'expression à gauche soit d'un type **`Enregistrement`** et qu'à droite se trouve un identificateur correspondant à un nom de champ du dit enregistrement. Ces vérifications sont réalisées par le compilateur.

Voici un petit programme qui manipule une variable de type `Date`.

```
Variables
    d1, d2: Date        -- deux dates
    année: Entier;
Début
    -- initialiser la date d1
    d1.jour ← 23
    d1.mois ← 11
    d1.année ← 2000

    -- initialiser la date d2 à partir de d1
    d2 ← d1

    -- afficher la date d2
    Écrire(d2.jour)
    Écrire('/')
    Écrire(d2.mois)
    Écrire('/')
    Écrire(d2.année)

    -- conserver l'année de d2
    année ← d2.année

    -- saisir l'année de d2
    Lire(d2.année)
Fin
```

En C.

3.4.1 Les enregistrements en C

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

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

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

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

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

```
typedef struct Date Date;

int main()
{
    Date d1, d2;        /* deux dates */
    int annee;

    /* initialiser la date d1 */
    d1.jour = 23;
    d1.mois = 11;
    d1.annee = 2000;

    /* initialiser la date d2 à partir de d1 */
    d2 = d1;

    /* afficher la date d2 */
    printf("%d/%d/%d\n", d2.jour, d2.mois, d2.annee);

    /* conserver l'année de d2 */
    annee = d2.annee;

    /* saisir l'année de d2 */
    scanf("%d", &d2.annee);

    /* afficher la date d2 */
    printf("%d/%d/%d\n", d2.jour, d2.mois, d2.annee);

    return EXIT_SUCCESS;
}
```

4 Choix entre type énuméré, tableau et enregistrement

Une différence fondamentale entre le type énuméré et les deux autres est que le type énuméré ne permet de stocker qu'une seule donnée qui correspond à l'une des valeurs listées dans la définition du type. On utilisera donc un type énuméré lorsqu'on modélise une donnée qui peut prendre une seule valeur parmis plusieurs.

Au contraire, pour les tableaux comme pour les enregistrements, nous avons à faire à des types qui regoupent plusieurs données (valeurs) sous un même nom.

– Dans le cas d'un tableau, les valeurs sont *interchangeables*, elles ont donc nécessairement le même type. Dans un tableau, il est logique d'accéder à une variable directement au moyen d'un indice (de type scalaire).
– Dans le cas d'un enregistrement, les différentes valeurs sont de *natures différentes*. Elles ne peuvent donc pas (et elles n'ont pas à) être accédées de manière uniforme comme par exemple au sein d'une boucle. Elles peuvent avoir des types différents.