C/C++

Ch 4 – Tableaux et Structures de données

I. Introduction

A. Rappels au sujet des types de données

Les algorithmes manipulent des données : nous avons vu quels étaient les types de données de base que nous pouvions rencontrer :

- des nombres, entiers et réels
- des caractères, des chaînes de caractères
- des valeurs logiques.

Il arrive cependant très souvent que nous ayons à traiter, dans des algorithmes,

- un grand nombre de données de la même manière :
  - effectuer une recherche sur un ensemble de nombres, trier, etc.
- ou bien des données ayant une forme plus complexe :
  - en dessin 2D, la notion de « point » correspond à la définition de 2 ou 3 nombres entiers : l'abscisse (X), l'ordonnée (Y), un code couleur.

B. Notion de TABLEAU

Un exemple de problème à traiter :
Demander la saisie de 20 nombres entiers, trier ces nombres dans l'ordre croissant et afficher les nombres dans le bon ordre.

Une première réflexion autour de la transcription en pseudo-code de la résolution du problème nous permet d'écrire :

```
% Déclaration de 20 nombres entiers  %
VAR Vn1, Vn2, Vn3, . . . , Vn18, Vn19, Vn20 : ENTIER
% Demander la saisie des 20 nombres %
AFFICHER ("Saisir le nombre 1")
```

```
SAISIR(Vn1)
AFFICHER ("Saisir le nombre 2")
SAISIR(Vn2)
AFFICHER ("Saisir le nombre 3")
SAISIR(Vn3)
. . . etc. . . (En tout 40 lignes de code. . . )

% il faudra ensuite trier les nombres . bon courage . . . %
```

Afin de traiter un ensemble de valeurs de même type, la notion de tableau a été mise au point : elle offre la possibilité de **déclarer en une seule fois plusieurs occurrences d'une variable**.
Dans notre exemple :

```
% Déclaration d'un tableau de 20 nombres entiers  %
VAR Tnombre[20] : ENTIER
```

L'intérêt vient ensuite lorsqu'il s'agit d'initialiser la valeur de ces 20 nombres (par exemple : faire 20 fois, « demander la saisie d'un nombre ») :

```
% Demander la saisie des 20 nombres %
POUR i DE 1 A 20 FAIRE
    AFFICHER("Saisir le nombre ",i)
    SAISIR(Tnombre[i])
finPOUR
```

C. Notion de TYPE STRUCTURE, ou structure

Les types de données de base fournis correspondent à des données élémentaires. Pour résoudre des problèmes complexes, nous devrons souvent manipuler des regroupements de ces données élémentaires.
Par exemple, dans le domaine du graphisme, nous devrons gérer la notion de point, en 2D ou 3D. Le type de données d'un point ne sera pas un type de base, mais sera composé de plusieurs types de base :

- Un nombre entier pour l'abscisse X,
- Un nombre entier pour l'ordonnée Y,
- Un nombre entier pour un code couleur.

Pour gérer 4 points nous devrions écrire :

```
% Déclaration des variables permettant de traiter 4 points %
VAR Vabcisse1, Vordonnee1, Vcouleur1 : ENTIER
    Vabcisse2, Vordonnee2, Vcouleur2 : ENTIER
```

Les types structurés vont nous permettre d'étendre la notion de type de donnée en autorisant la création de nouveaux types.
Dans notre exemple :

```
% Définition d'un nouveau type structuré de données : POINT %
TYPE POINT
    Vabcisse, Vordonnee, Vcouleur : ENTIER
finTYPE
% Déclaration des variables de type « POINT » %
VAR Vpoint1, Vpoint2, Vpoint3, Vpoint4: POINT
```

II. Tableaux

A. Une définition

Un **TABLEAU** (ou **VECTEUR)** est une juxtaposition, sous un **NOM UNIQUE**, d'un certain nombre de **VARIABLES DE MEME TYPE**, auxquelles on accède individuellement grâce à un numéro d'ordre, ou rang, qu'on appelle **INDICE.**
En **C++**, les **INDICES** sont numérotés **A PARTIR DE 0**.
ATTENTION : **LE CONTROLE DES INDICES EST A LA CHARGE DU DEVELOPPEUR** (le compilateur ne vérifie pas le dépassement des limites)

Syntaxe 1 : tableau à 1 dimension, non initialisé

```
type_donnees nom_tableau[nbre_elements];
```

- **type_donnees**
  - Correspond au type de données des éléments du tableau
- **nbre_elements**:
  - nombre d'éléments du tableau
  - l'**indice** est alors compris entre **0** et (**nbre_elements – 1)**

Exemple : déclaration d'un tableau d'entiers

```
int Ttemperature[12];
```

int Ttemperature [12] ;

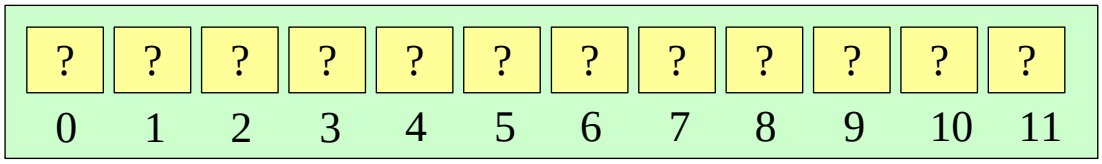


Figure 1 : un tableau de 12 entiers numérotés de 0 à 11

Exemple : initialisation du tableau à 0

```
int i ;
for (i=0 ;i<=11 ;i++)
{
    Ttemperature[i]=0 ;
}
// ou bien :
for (i=1 ;i<=12 ;i++)
{
    Ttemperature[i - 1]=0 ;
}
```

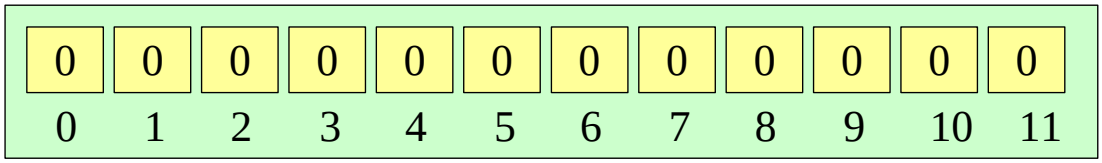


Figure 2 : contenu après initialisation

Exemple : initialisation à partir d'une saisie

```
int i ;
for (i=0 ;i<=11 ;i++)
{
    cout << endl << "temperature " << I << ":";
    cin >> Ttemperature[i] ;
}
```

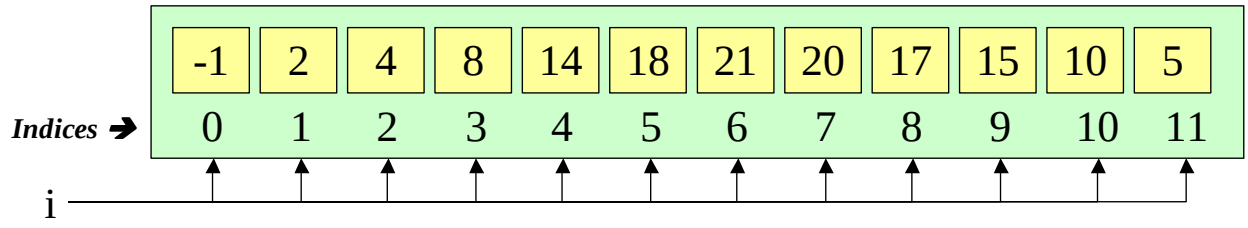


Figure 3 : 'i' va prendre successivement les valeurs 0, 1, 2, ...,10, 11

Exemple : parcourir le tableau et afficher toutes les valeurs

```
// parcours de 0 à 11
for (i=0 ;i<=11 ;i++)
{
    cout << Ttemperature[i] << " / ";
}
```

```
// ou bien en ordre inverse :
for (i=11 ;i>=0 ;i--)
{
    cout << Ttemperature[i] << " / ";
}
```

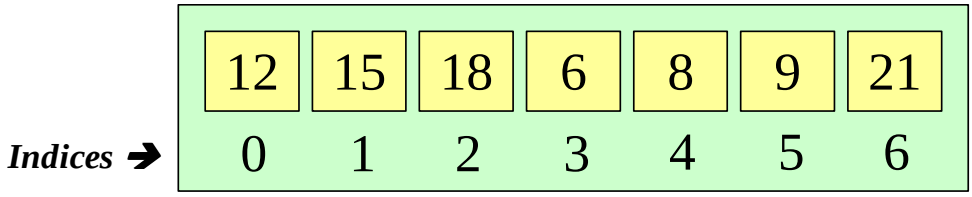
Syntaxe 2 : tableau à 1 dimension, initialisé

```
type_donnees nom_tableau [] = {val1, val2, . . . valN} ;
```

- **type_donnees**
  - Correspond au type de données des éléments du tableau
- **val1, val2, valN**:
  - valeurs d'initialisation du tableau
  - l'**indice** est alors compris entre **0** et le nombre de valeurs entrées **– 1**
- **si un nombre d'éléments est fourni, le nombre de valeurs en doit pas être supérieur au nombre d'éléments**

Exemple : déclaration d'un tableau d'entiers

```
int Ttemperature[] = {12,15,18,6,8,9,21};
```



B. Dimensions, tableaux multi dimensionnels

Syntaxe tableau à N dimensions :

```
type_donnees nom_tableau[elem1][elem2] . . .[elemN];
```

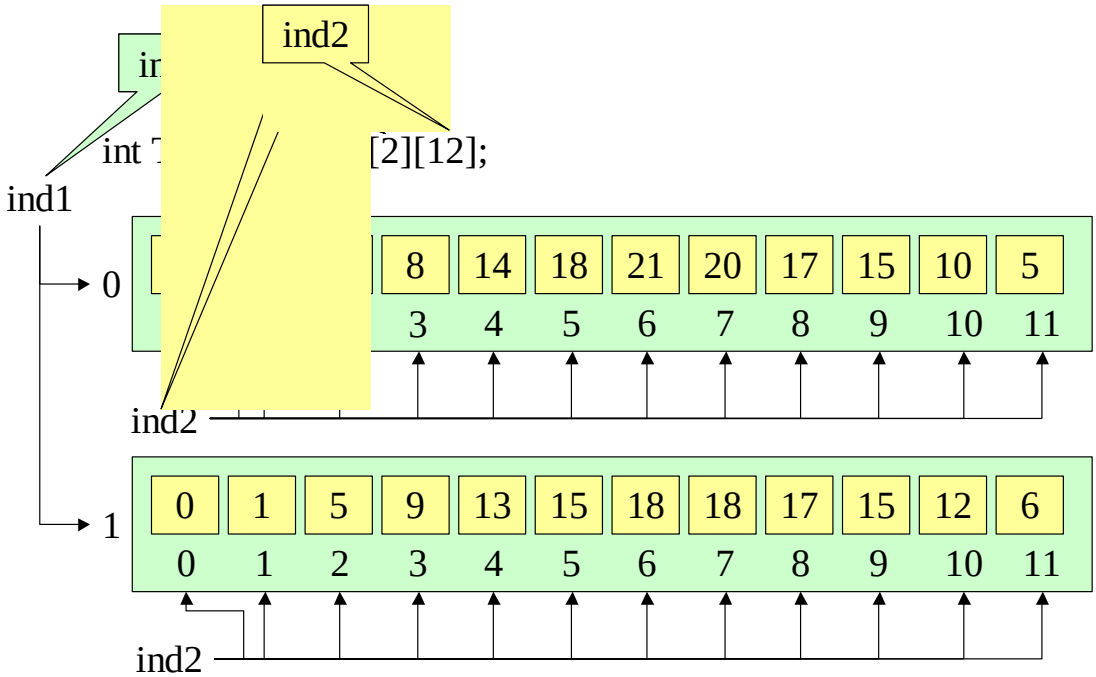
- **nom_tableau**
  - correspond au nom (identificateur) donné au tableau (=aux variables qui le composent)
- **elem1, elem2, elemN** : nombre d'éléments de chacune des dimensions
- **type_données**
  - type de données des éléments du tableau

Exemple : tableau de températures de 2 ans, 12 mois par an

```
int Ttemperature[2][12] ; // sur 2 ans, temp. des 12 mois

    // initialiser les 12 temperatures d'indice 0 %
    Ttemperature[0][0] ← -1
    Ttemperature[0][1] ← 2
    . . . etc. . . .
    Ttemperature[0][11] ← 5

    // initialiser les 12 temperatures d'indice 1
    Ttemperature[1][0] ← 0
    . . . etc. . . .
    Ttemperature[1][10] ← 12
    Ttemperature[1][11] ← 6
. . .
```



Exemple : initialiser les 2 dimensions du tableau

```
    // initialiser le tableau sur 2 ans %
    for (i=0 ;i<=1 ; i++) // boucle sur les années
       for (j=0;j<=11;j++) // boucle sur les temp. de chaque année
           Ttemperature[i][j] = 0;
```

III.Types structurés, structures (langage C)

A. Définition et syntaxe

Un **TYPE DE DONNEES STRUCTURE** permet la constitution d'un regroupement de variables de types de données différents.

Des variables de ce nouveau type pourront ensuite être déclarées et utilisées, tout comme les variables des types de base

Syntaxe de définition d'une structure :

```
struct nom_structure {
    . . . declaration_1 . . . ;
    . . . declaration_2 . . . ;
    . . .
    . . . declaration_N . . . ;
};
```

- **nom_structure**
  - l'identificateur du type de données structuré
- **declaration_1, declaration_1, declaration_N :**
  - déclarations des variables qui composent ce type structuré (variables élémentaires, tableaux, etc.)
    - identificateur de variable, avec éventuellement les dimensions pour un tableau
    - type de données de cette variable.

Exemple :

```
struct salarie {
     char Vnom[20] ;
     char Vprenom[20] ;
     int VanneeNaissance ;
     float TsalaireMois[12] ;
};
```

Syntaxe de déclaration d'une variable du type structure :

```
struct nom_structure nom_var1, nom_var2, . . .,nom_varN ;
```

- **nom_structure** : identificateur du type structure
- **nom_var1, nom_var2, nom_varN** : identificateur des variables déclarées

Remarque : le mot clef **struct** est optionnel

Exemple : Déclaration d'une variable de ce type :

```
struct salarie VunSalarie ;
```

Exemple : Affectation d'une valeur à une des variables élémentaires du type structuré

```
strcpy(VunSalarie.Vnom ,"dupont") ;
strcpy(VunSalarie.Vprenom , "pierre") ;
VunSalarie.VanneeNaissance = 1980 ;
VunSalarie.TsalaireMois[0] = 1350.80 ;
VunSalarie.TsalaireMois[11] = 1842.50 ;
```

Exemple : déclaration et initialisation d'un type structuré

```
salarie VunSalarie = {
     "dupont",
     "pierre",
     1980,
     1350.80,0,0,0,0,0,0,0,0,0,0,1842.50} ;
```

B. Imbrication de types structurés

Un type de données structuré peut être utilisé dans la définition d'un autre type de données structuré.

Exemple :

```
struct date {
     int Vjour ;
     int Vmois ;
     int Vannee ;
} ;
// la définition du type SALARIE utilise le type DATE
struct salarie {
     char Vnom[20] ;
     char Vprenom[20] ;
     struct date VdateNaissance ;
     float TsalaireMois[12] ;
} ;
struct salarie VunSalarie ;

VunSalarie.VdateNaissance.Vjour = 1 ;
VunSalarie.VdateNaissance.Vmois = 12 ;
VunSalarie.VdateNaissance.Vannee = 1980 ;
```

C. Utilisation des types structurés dans les tableaux

La création de tableaux de variables structurés permet le regroupement dans un seul tableau de toutes les variables nécessaires à la gestion des salariés.

Exemple : pour gérer les données d'un ensemble de salariés :

```
. . .
struct salarie {
     char Vnom[20] ;
     char Vprenom[20] ;
     struct date VdateNaissance ;
     float TsalaireMois[12] ;
} ;
const int NB_FICHES = 100 ;
struct salarie TficheSalarie[NB_FICHES] ;
int i,j ;

for ( i=0 ;i<NB_FICHES; i++)
    {
          cin >> TficheSalarie[i].Vnom ;
          cin >> TficheSalarie[i].VdateNaissance.Vjour ;
```

```
        . . .
        cout << endl << "saisie de 12 salaires :" ;
        for ( j=0 ; j<=11 ; j++)
            cin >> TficheSalarie[i].TsalaireMois[j] ;
    } ;
```

D. Typedef pour créer des alias sur les types

Le mot clef **typedef** (langage C/C++) permet de définir des synonymes sur des types de données et ainsi créer des noms personnalisés (intérêt pour redéfinir de manière plus claire des noms de types existants)

Exemple : redéfinir le type int

```
. . .
typedef int ENTIER ; // création d'un synonyme

ENTIER i, j ; // déclaration de variables
```

Exemple : utilisation pour les types structurés

```
. . .
struct fiche_salarie {
     char Vnom[20] ;
     char Vprenom[20] ;
     struct date VdateNaissance ;
     float TsalaireMois[12] ;
} ;

typedef struct fiche_salarie FS ; // création d'un synonyme

FS Vemp1, Vemp2; // déclaration de variables
```

E. Des types structurés C aux classes C++

Les classes, au même type que les types structurés, permettent le groupement de variables de types différents ; elles permettent en plus d'inclure dans leur définition des procédures et fonctions en lien avec ces variables (on parle de données membres et de méthodes membres).

Avec les classes, on entre dans la programmation orientée objets. Le C++ étant un langage de programmation orienté objets, il propose un certain nombre de classes que nous seront amenés à découvrir au cours d'exercices. Nous nous limiterons à l'utilisation de ces classes dans l'objectif de pouvoir utiliser certaines fonctionnalités intéressantes en C++. (***cin***, par exemple, est un objet dont nous utilisons la méthode ***getligne***)