## Exemple introductif

Écrire une procédure qui affiche une table de multiplication.

$$\begin{array}{cccccccc}
1 & 2 & 3 & 4 & \cdots & 9 & 10 \\
2 & 4 & 6 & 8 & \cdots & 18 & 20 \\
3 & 6 & 9 & 12 & \cdots & 27 & 30 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
10 & 20 & 30 & 40 & \cdots & 90 & 100
\end{array}$$

On peut commencer par écrire une procédure auxiliaire qui, pour un entier $n$, affiche la $n$-ième ligne de la table de multiplication.

```
Procédure
  table_multiplication_par(valeur n : entier)
Variable i : entier
Début table_multiplication_par
    Pour i <- 1 à 10 faire
       afficher(i * n)
    FinPour
    afficher(RetourALaLigne)
Fin table_multiplication_par
```





Voici la procédure `table_multiplication` qui utilise la procédure précédente :

```
Procédure table_multiplication()
Variable i : entier
Début table_multiplication
    Pour i <- 1 à 10 faire
        table_multiplication_par(i)
    FinPour
Fin table_multiplication
```

On peut aussi effectuer l'affichage dans une seule procédure en utilisant deux boucles *imbriquées* l'une dans l'autre :

```
Procédure table_multiplication()
Variable i,j : entier
Début table_multiplication
    Pour i <- 1 à 10 faire
       Pour j <- 1 à 10 faire
          afficher(i * j)
       FinPour
       afficher(RetourALaLigne)
    FinPour
Fin table_multiplication
```





### Triangle d'étoiles

Écrivez une procédure qui prend en paramètre un entier $n$ et affiche une ligne de $n$ étoiles.

Écrivez une procédure qui prend en paramètre un entier $h$ et affiche un triangle de hauteur $h$, la pointe vers le haut. On pourra soit se servir de la procédure précédente, soit utiliser des boucles imbriquées

Par exemple, pour $h = 5$ on veut obtenir

```
*
*  *
*  *  *
*  *  *  *
*  *  *  *  *
```

Voici une procédure qui affiche une ligne de $n$ étoiles :

```
Procédure ligne_étoilée(valeur n : entier)
Variable i : entier
Début ligne_étoilée
    Pour i <- 1 à n faire
       afficher(*)
    FinPour
    afficher(RetourALaLigne)
Fin triangle_étoilé
```





Voici une procédure qui affiche le triangle complet :

```
Procédure triangle_étoilé(valeur h : entier)
Variable i,j : entier
Début triangle_étoilé
    Pour i <- 1 à h faire
        Pour j <- 1 à i faire
           afficher(*)
        FinPour
        afficher(RetourALaLigne)
    FinPour
Fin triangle_étoilé
```

Dans des boucles imbriquées, on peut utiliser l'indice de la boucle extérieure comme borne de la boucle intérieure.

Comment pourrait-on modifier cet exemple pour qu'il affiche le triangle la pointe vers le bas ?

```
*  *  *  *  *
*  *  *  *
*  *  *
*  *
*
```

Bien évidemment, on peut imbriquer tous les types de boucles (par exemple une boucle *pour* dans une boucle *tant que*), et il n'y a aucune restriction sur le nombre d'imbrications.

## Exercices

1. Écrire une fonction qui calcule $\sum_{i=1}^{n} \sum_{j=1}^{m} i * j$.

2. Écrire une fonction qui calcule $\sum_{i=1}^{n} \sum_{j=1}^{i} j$.

3. Écrire une fonction qui calcule $\sum_{i=1}^{n} \sum_{j=i}^{n} \sum_{k=i}^{j} \sum_{l=1}^{k} l$.





# Matrices

## Relevé de notes d'une classe

| Matière | Maths | Physique | Info | Anglais |
|---|---|---|---|---|
| Pierre | 12 | 09 | 14 | 13 |
| Marie | 15 | 14 | 14 | 10 |
| Paul | 06 | 10 | 08 | 16 |
| Jacques | 10 | 11 | 12 | 10 |
| Nathalie | 08 | 11 | 07 | 09 |

Chaque ligne correspondant aux notes d'un étudiant est en fait un tableau.

| | [1] | [2] | [3] | [4] |
|---|---|---|---|---|
| Notes de Paul | 06 | 10 | 08 | 16 |

On représente le relevé de notes sous forme d'un tableau de tableaux, qu'on appellera `notes`. Le tableau des notes de Paul (troisième ligne) correspond donc à `notes[3]`.





| | | [1] | [2] | [3] | [4] |
|---|---|---|---|---|---|
| `notes[1]` | Pierre | 12 | 09 | 14 | 13 |
| `notes[2]` | Marie | 15 | 14 | 14 | 10 |
| `notes[3]` | Paul | 06 | 10 | 08 | 16 |
| `notes[4]` | Jacques | 10 | 11 | 12 | 10 |
| `notes[5]` | Nathalie | 08 | 11 | 07 | 09 |

La note d'anglais de Paul (16) est alors `notes[3][4]`. On peut réaliser les opérations habituelles sur cette variable :
- affectation : `notes[3][4]` $\longleftarrow$ `15`
- utilisation : `somme` $\longleftarrow$ `somme + notes[3][4]`





# Syntaxe

Déclaration d'un tableau multidimensionnel

```
Type nom = tableau à n dimensions de nom_type
           indexé par
           Indices_1, Indices_2, ..., Indices_n
```

Les indices sont identiques à ceux des tableaux simples.

**Exemple** Voici un programme qui déclare et stocke le tableau suivant.

| | Maths | Physique |
|---|---|---|
| Pierre | 12 | 09 |
| Paul | 06 | 10 |
| Jacques | 10 | 11 |

```
Type Relevé = tableau d'entiers à 2 dimensions
              indexé par [1..3],[1..2]

Programme Saisie
  Variables notes : Relevé

Début Saisie
  notes[1][1] <- 12
  notes[1][2] <- 09

  notes[2][1] <- 06
  notes[2][2] <- 10

  notes[3][1] <- 10
  notes[3][2] <- 11
  Fin Saisie
Fin Programme
```





## Saisie d'un tableau à deux dimensions

```
Constante NbElèves=3
Constante NbExams=2
Type Relevé = tableau d'entiers à 2 dimensions
              indexé par
              [1..NbElèves], [1..NbExams]
Type Elève = tableau d'entiers
            indexé par [1..NbExams]

/* Procédure qui saisit les notes d'un élève */

procédure Saisie_élève(référence tab: Elève)
variable j: entier
début Saisie_élève
  pour j <- 1 à NbExams faire
    afficher "exam no." j " ? "
    lire tab[j]
  fin pour
fin Saisie_élève
```





```
/* Procédure qui saisit le relevé de notes
 * d'une classe */

procédure Saisie_relevé(référence notes: Relevé)
variable i: entier
début Saisie_relevé
  pour i <- 1 à NbElèves faire
    afficher "Saisie des notes de l'élève " i
    Saisie_élève(notes[i])
  fin pour
fin Saisie_relevé

Programme SaisieBis
  Variables notes : Relevé

Début SaisieBis
  Saisie_relevé(notes)
Fin SaisieBis
```





## Boucles imbriquées sans utiliser de fonction intermédiaire.

On veut augmenter toutes les notes d'un point.

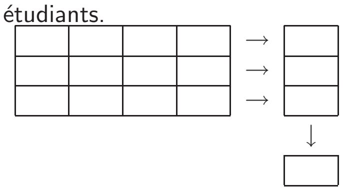
```
procédure Augmentation_Notes (référence
                              notes: Relevé)
Variables i, j: entiers

Début Augmentation_Notes
  Pour i <- 1 à NbElèves faire
     Pour j <- 1 à NbExams faire
        notes[i][j] <- notes[i][j] + 1
     Fin Pour // j
  Fin Pour // i
Fin Augmentation_Notes
```

## Exercices

1. Écrivez un programme qui réalise l'affichage d'un tableau à deux dimensions représentant un relevé de notes.

2. Écrivez une fonction qui calcule la moyenne des notes d'*un* étudiant, données sous laforme d'un tableau à une dimension.
   En déduire une fonction qui calcule l'ensemble des moyennes pour tous les étudiants sous forme d'un tableau, puis une fonction qui calcule la moyenne générale de tous les étudiants.







## Syntaxe en C++

**Déclaration d'un nouveau type de données :**

```
typedef type_éléments nom_type_1 [dim_1];
typedef nom_type_1    nom_type_2 [dim_2];
...
typedef nom_type_n-1  nom_type_n [dim_n];
```

**Déclaration d'une variable de ce type :**

```
nom_type_n nom_variable;
```





### Exemple

```
typedef int Eleve[2];
  // deux notes par élève
typedef Eleve Releve[3];
  // trois élèves dans le relevé

void main(){
  Releve notes;

  notes[0][0] = 12;
  notes[0][1] = 09;
  notes[1][0] = 06;
  notes[1][1] = 10;
  notes[2][0] = 10;
  notes[2][1] = 11;
}
```

**Attention : les indices commencent à 0**

Pour conserver les notations de 1 à `NbEleves` et de 1 à `NbExams`, on peut déclarer une ligne et une colonne de plus et ne pas utiliser l'indice 0 :

```
typedef int Eleve[3];
typedef Eleve Releve[4];
...
notes[1][1]=12;
notes[1][2]=09;
...
```





## Exercice

1. Traduire en C++ les exercices précédents.

## Types structurés

### Exemple

| **NOM :** | **Prénom :** |
|---|---|
| **No. d'étudiant :** | |
| **Moyenne :** | |

### Déclaration en Pseudo

```
Type <nom_type> = Structure
                    champ1 : type_champ1
                    champ2 : type_champ2
                    ...
                  Fin Structure
```

Sur l'exemple :

```
Type Fiche = Structure
                nom, prénom : chaîne de caractères
                num : entier
                moy : réel
             Fin Structure
```

### Utilisation

```
Variables
  etudiant : Fiche
...
etudiant.nom <- "Dupont"
etudiant.moy <- etudiant.moy + 1
afficher etudiant.moy
...
```





## Exercice

1. Définir un type `Polynôme` pour manipuler des polynomes du deuxième degré (de la forme $ax^2 + bx + c$).
   Écrire une fonction qui prend comme paramètre un polynome et calcule son discriminant.





## En C++

### Déclaration

```
typedef struct {
    type_champ1 champ1;
    type_champ2 champ2;
    ...
} nom_type;
```

Sur l'exemple :

```
#include <string>
// (pour utiliser les chaînes de caractères)

typedef struct {
    string nom, prenom;
    int num;
    float moy;
} Fiche;
```

### Utilisation

```
  Fiche etudiant;
  // etudiant est une variable de type Fiche

  etudiant.prenom = "David";
  etudiant.moy = Etudiant.moy + 1;
```





## Enumération

```
Type Mention = Enumération
                   Ajourné, Rattrapage, Passable,
                   ABien, Bien, TBien
               Fin Enumération

Variables  Résultat : Mention

Fonction Résultat(valeur note : entier) : Mention
Début Résultat
    si note<8 alors retourner Ajourné
    sinon si note<10 alors retourner Rattrapage
    sinon si note<12 alors retourner Passable
    sinon si note<14 alors retourner ABien
    sinon si note<16 alors retourner Bien
    sinon retourner TBien
    fin si
Fin Résultat
```

### En C++

```
typedef enum { valeur1, valeur2, ... } nom_type;
```

Sur l'exemple :

```
typedef enum {
    Ajourne, Rattrapage, Passable,
    ABien, Bien, TBien
} Mention;
```