BTS
BREVET DE TECHNICIEN SUPÉRIEUR
LYCÉE QUALIFIANT MOHAMED V
ISSAOUIRA

المملكة المغربية
وزارة التربية الوطنية
والتعليم الأولي والرياضة

Algorithmique et programmation

2024-2025

Chapitre 3 :
Les tableaux et les fonctions

Les tableaux

Les tableaux

Ensemble de données du même type

Exemple de problème :

Saisir une suite de nombres, puis afficher cette suite après avoir divisé tous les nombres par la valeur maximale de la suite.

Nécessité de conserver les nombres en mémoire

variable contenant une valeur: Val

132

variable contenant une collection de valeurs du même type:

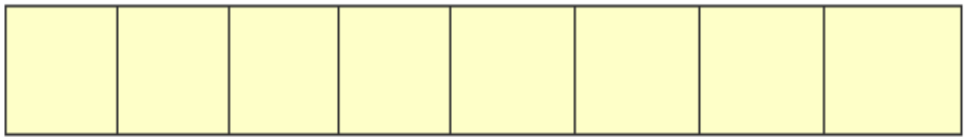
TabVal

132	52	-57	-8902	-841	8100	–641

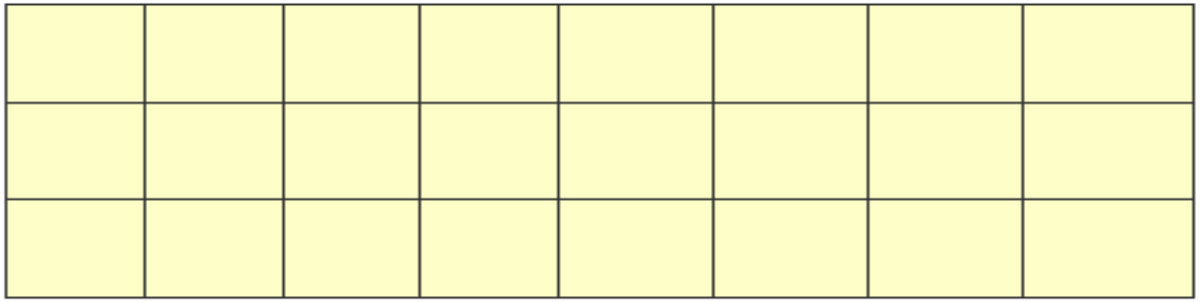
Ensemble de données du même type

- **Structure de données permettant d'effectuer un même traitement sur des données de même nature**

- Tableau à **une** Dimension



- Tableau à **deux** dimensions



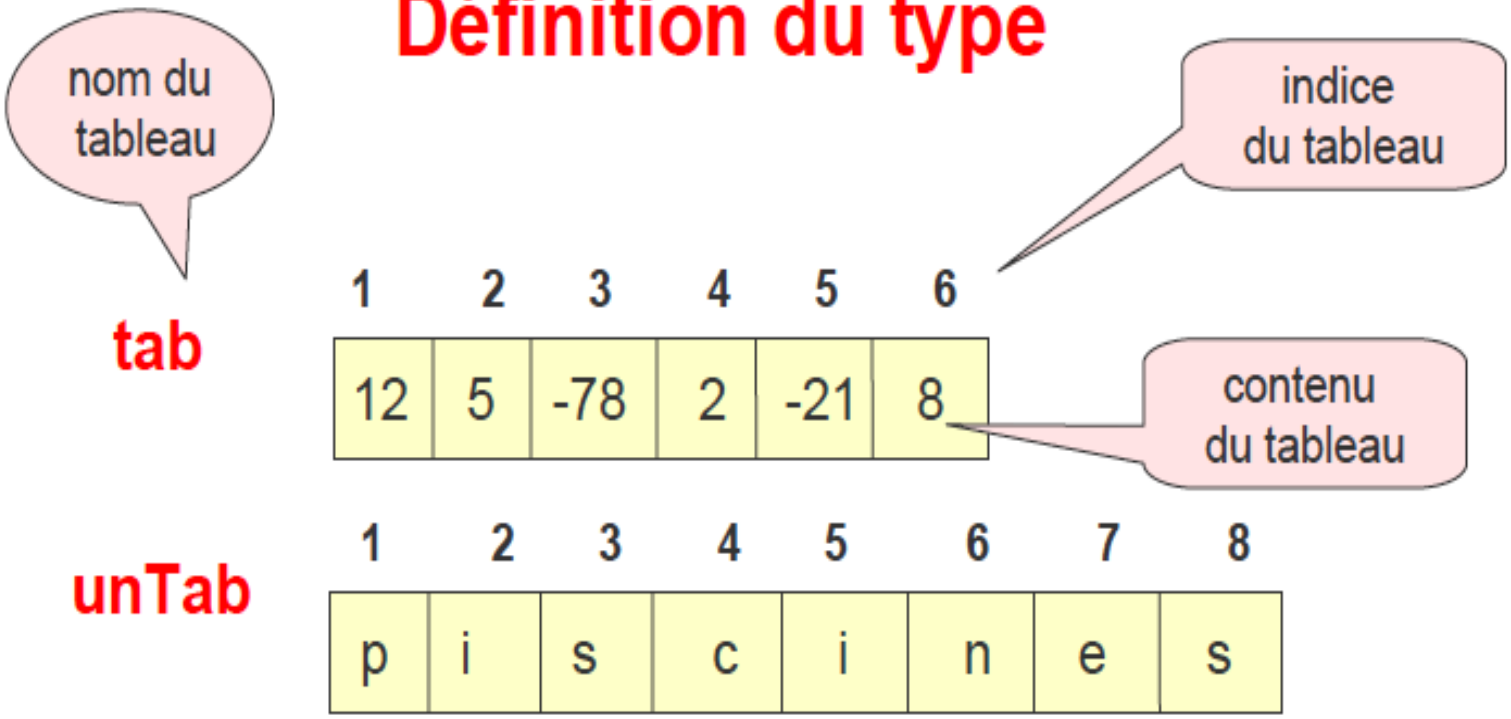
Exemples d’applications

- Ensemble de valeurs entières, réelles, booléennes,....
- Ensemble de noms (type chaîne)
- Ensemble de caractères (type caractère)
- Ensemble d'adresses (type Adresse : nom, adresse, num téléphone)

Traitements sur les tableaux

- On veut pouvoir :
  - **Créer** des tableaux
  - **Ranger** des valeurs dans un tableau
  - **Récupérer, consulter** des valeurs rangées dans un tableau
  - **Rechercher** si une valeur est dans un tableau
  - **Mettre à jour** des valeurs dans un tableau
  - **Modifier** la façon dont les valeurs sont rangées dans un tableau (par exemple : les trier de différentes manières)
  - Effectuer des **opérations entre tableaux :** comparaison de tableaux, multiplication,...

Définition du type



- Indices : en général, démarrage à 1, **mais en C,**

  démarrage à 0

- Nombre d'octets occupés : dépend du type des valeurs enregistrées

Déclaration d'un tableau

- La **déclaration** d'un tableau s'effectue en précisant le **type** de ses éléments et sa **dimension** (le nombre de ses éléments)
  - En pseudo code :

    variable **tableau** identificateur**[dimension] : type**
  - Exemple : déclaration d'un tableau pouvant contenir jusqu'à 30 réels

    variable **tableau** notes**[30] : réel**

Utilisation d'un tableau par les indices

- Accès en lecture :

Ecrire(notes[4]); *// le contenu du tableau à l'indice 4 et aficher à l'ecran*

- Accès en écriture :

notes[3] ←18; *// la valeur 18 est placée dans le tableau à l'indice 3*

- **Lire**(notes[5]); *// la valeur entrée par l'utilisateur est enregistrée dans le tableau à l'indice 5*

Tableaux : exemples (1)

- Pour le calcul du nombre d'étudiants ayant une note supérieure à 10 avec les tableaux, on peut écrire :

```
Algorithme Mon_Premier_Tab
  Variables i ,nbre : entier
            tableau notes[30] : réel
Début
        nbre ← 0;
          Pour i allant de 0 à 29
               Lire(notes[i] );
                 Si (notes[i] >10) alors
                    nbre ←nbre+1;
                    FinSi
           FinPour
        Ecrire ("le nombre de notes supérieures à 10 est : ", nbre);
Fin
```

Tableaux à deux dimensions

- Les langages de programmation permettent de déclarer des tableaux dans lesquels les valeurs sont repérées par **deux indices**. Ceci est utile par exemple pour représenter des matrices

- En pseudo code, un tableau à deux dimensions se **déclare** ainsi :

  variable **tableau** identificateur**[dimension1] [dimension2] : type**

  - Exemple : une matrice A de 3 lignes et 4 colonnes dont les éléments sont réels

    variable **tableau** A**[3][4] : réel**

- **A[i][j]** permet d'accéder à l'élément de la matrice qui se trouve à l'intersection de la ligne i et de la colonne j

Tableaux à deux dimensions (exemple)

	1	2	3	4	5	6	7
1	10	3	25	14	2	1	8
2	9	20	7	12	2	4	7

tableau à 2 lignes et 7 colonnes

- Accès en lecture :
  - **Ecrire(**A[1] [7]); *// la valeur contenue en ligne 1 colonne 7 est affichée à l'écran*
- Accès en écriture :
  - A[2] [4] ← 36;
  - **Lire**(points[2][4]); *// la valeur fournie est enregistrée en ligne 2,col 4*

Exemples : somme de deux matrices

- Procédure qui calcule la somme de deux matrices :

```
Algorithme  SommeMatrices
   variables n, m,i,j : enties
                tableau A[n][m], B[n][m],C[n][m] : réels
Début

        Pour i allant de 0 à n-1
        Pour j allant de 0 à m-1
         C[i][j] ← A[i][j]+B[i][j];
        FinPour
        FinPour
  Fin
```

Tableaux : Exemple d'exercice

Écrire l'algorithme du traitement qui permet

- de saisir 10 nombres entiers dans un tableau à une dimension, puis qui –
- recherche et affiche la valeur minimale entrée dans un tableau.

L'affichage mentionnera également l'indice auquel se trouve ce minimum.

Les Fonctions et les procédures

Exercice 7 :

- Écrire un algorithme qui permette de connaître ses chances de gagner au tiercé, quarté, quinté et autres impôts volontaires.
- On demande à l'utilisateur le nombre de chevaux partants, et le nombre de chevaux joués. Les deux messages affichés devront être :
- -Dans l'ordre : une chance sur X de gagner
- -Dans le désordre : une chance sur Y de gagner
- X et Y nous sont donnés par la formule suivante, si n est le nombre de chevaux partants et p le nombre de chevaux joués (on rappelle que le signe ! signifie "factorielle", comme dans l'exercice 5.6 ci-dessus) :
- $X = n ! / (n - p) !$
- $Y = n ! / (p ! * (n – p) !)$

Problématique

- Dès qu'on commence à écrire des programmes sophistiqués, il devient difficile d'avoir une vision globale sur son fonctionnement

- Difficulté de trouver des erreurs

- Solution : décomposer le problème en sous problèmes
  - Trouver une solution à chacun
  - La solution partielle donne lieu à un sous-programme

Programmation procédurale

- Principe:
  - Il s'agit d'écrire des programmes en utilisant des sous-programmes
- Forme générale d'un programme

Programme P

Sous-programme SP_1

…

Sous-programme SP_n

FinP

Exemples:

- Un Algorithme qui résoudre le problème 7 TD 3:
- $X = n ! / (n - p) !$
- $Y = n ! / (p ! * (n – p) !)$

Algorithme A

A1: calcule de $n !$

A2: calcule de $(n - p) !$

A3: calcule de $p !$

A4: Affichage du résultat

Fin A

Sous-algorithme

- **Définition**
  - Un sous-algorithme est un élément d'algorithme nommé et éventuellement paramétré que l'on définit afin de pouvoir ensuite l'appeler par son nom en affectant, s'il y a lieu, des valeurs aux paramètres.

  - **Intérêt :**
    - Réaliser un découpage d'une tâche en sous-tâche.
    - Effectuer une seule description d'une tâche commune
    - Concevoir une application de manière descendante en entrant de plus en plus dans les détails

  - **Structure :** un sous-algorithme est composé
    - **D'une tête** nom sous-algorithme, paramètres(arguments) avec leur type
    - **D'un corps** des déclarations d'objets locaux aux sous-algorithme, instructions à exécuter

Syntaxe

Sous-algorithme Nom(liste des paramètres)
déclarations des variables locales
Début
Corps du sous-algorihtme
Fin
Algorithme Nom
Déclaration des variables
Début
Instructions
Appel du sous-algorithme
Instructions
Fin

Procédures & Fonctions

- En algorithmique, on distingue deux types de sous-programmes

Les fonctions

- Paramètres
- Type retourné

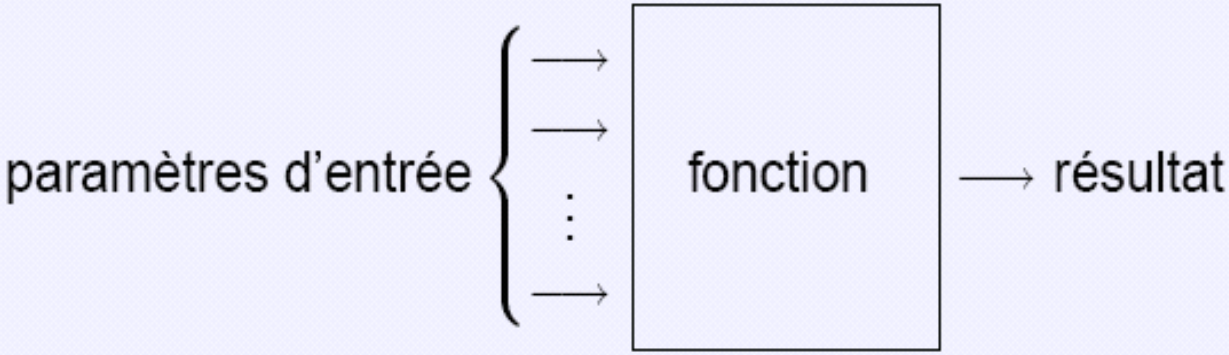
Les procédures

Paramètres

- Appel par variable
- Appel par valeur

Les Fonctions

- Une fonction est un sous-programme qui retourne une valeur calculée en fonction des valeurs passées en entrée



- Une fonction prend zéro ou plusieurs paramètres et renvoie éventuellement un résultat
- Une fonction qui ne retourne pas de résultat est une procédure

Forme d'une Fonction

- Une fonction s'écrit en dehors du programme principal sous la forme

Fonction identificateur (paramètres et leurs types) : type_fonction
Déclaration des variables
Début
    Instructions constituant le corps de la fonction
Retourne(résultat**);**
FinFonction

- **type de fonction :** le type du résultat renvoyé par la fonction
- **Identificateur :** le nom que l'on a donné à la fonction
- **liste de paramètres :** la liste des paramètres formels donnés en entrée avec leurs types
- **corps de la fonction :** un bloc d'instructions, pouvant comprendre la déclaration des « variables locales » à la fonction

Remarque : le corps de la fonction doit comporter une instruction de la forme : Retourne(expression); où expression est du type du résultat de la fonction

Exemple 1

Fonction qui retourne le carré d'un entier :

```
Fonction carre(n : entier): entier
Variables res: entier
 Début
        res ← n * n;
    Retourne(res);
FinFonction
```

Utilisation dans un algorithme

```
Algorithme ex1
  Variables  i, j: entier
  Fonction carre(n : entier): entier
  Début
        Retourne( n * n);
  FinFonction
Debut
        Lire (i);
        j ← carre(i);
        Ecrire(j);
Fin
```

Fonctions : exemple 2

La fonction SommeCarre suivante calcule la somme des carrées de deux réels x et y :

Fonction SommeCarre (x : réel, y: réel) : réel

variable z : réel

Debut

z ←x*x+y*y;

Paramètres formels

retourne (z);

FinFonction

La fonction Pair suivante détermine si un nombre est pair :

Fonction Pair (n : entier) : booléen

Debut

retourne (n%2=0);

FinFonction

Utilisation des fonctions

- L'utilisation d'une fonction se fera par simple écriture de son nom dans le programme principal. Le résultat étant une valeur, devra être affecté ou être utilisé dans une expression, une écriture, ...

Exemple :

```
Algorithme exepmleAppelFonction
                variables    z : réel
                             b : booléen
 Début
                b ←Pair(3);
                z ←5*SommeCarre(7,2)+1;
                Ecrire("SommeCarre(3,5)= ", SommeCarre(3,5));
 Fin
```

- Lors de l'appel Pair(3) le **paramètre formel** n est remplacé par le **paramètre effectif** 3

Fonctions: langage C

- C propose plusieurs fonctions qui sont déjà définies et qu'on peut utiliser dans nos programmes
- Exemples :
  - sqrt : retourne la racine carrée
    - sqrt(4) retourne 2
  - exp : retourne l'exponentielle d'un nombre
    - exp(0) retourne 1
  - …

Exercice

- Problème: Trouver le plus petit nombre premier strictement supérieur à un entier positif donné **n**
  - Utiliser l'algorithme qu'on a déjà fait estPremier (le plus petit nombre premier *p* supérieur à un entier *n)* en tant que fonction.
  - Fait appel a cette fonction a l'intérieur de l'algorithme premier-plus-petit

Procédure: définition

- Une procédure est un sous-programme qui ne retourne pas de valeur
- C'est donc un type particulier de fonction
- En général, une procédure modifie la valeur de ses paramètres

  Je dis bien « en général », ce n'est pas toujours le cas

Procédure: structure

- Tout comme les fonctions, une procédure est un sous-programme qui :
  - A un nom
  - Peut avoir des paramètres
  - Qui retourne une valeur d'un certain type
  - Qui peut avoir besoin de variables
  - Qui est composé d'instructions

Procèdure

- Dans certains cas, on peut avoir besoin de répéter une tache dans plusieurs endroits du programme, mais que dans cette tache on ne calcule pas de résultats ou qu'on calcule plusieurs résultats à la fois
- Dans ces cas on ne peut pas utiliser une fonction, on utilise une **procédure**
- Une **procédure** est un sous-programme semblable à une fonction mais qui **ne retourne rien**
- Une procédure s'écrit en dehors du programme principal sous la forme :

Procédure nom_procédure (paramètres et leurs types)

Déclaration des variables

Début

Instructions constituant le corps de la procédure**i**

FinProcédure

- Remarque : une procédure peut ne pas avoir de paramètres

Appel d'une procédure

- L'appel d'une procédure, se fait dans le programme principal ou dans une autre procédure par une instruction indiquant le nom de la procédure :

```
Procédure exemple_proc (…)
   Début
     …
FinProcédure

Algorithme exepmleAppelProcédure
  Début
      exemple_proc (…)                    …
  Fin
```

- Remarque : contrairement à l'appel d'une fonction, on ne peut pas affecter la procédure appelée ou l'utiliser dans une expression. L'appel d'une procédure est une instruction autonome

Paramètres d'une procédure

- Les paramètres servent à échanger des données entre le programme principale (ou la procédure appelante) et la procédure appelée

- Comme avec les fonctions :
  - Les paramètres placés dans la déclaration d'une procédure sont appelés **paramètres formels**. Ces paramètres peuvent prendre toutes les valeurs possibles mais ils sont abstraits (n'existent pas réellement)

  - Les paramètres placés dans l'appel d'une procédure sont appelés **paramètres effectifs**. ils contiennent les valeurs pour effectuer le traitement

- Le nombre de paramètres effectifs doit être égal au nombre de paramètres formels. L'ordre et le type des paramètres doivent correspondre

Exemple:

- Une procédure qui ajoute 2 à un entier

```
Procédure aug2(n : entier)
Début
        n ← n+2;
FinProcédure
```

Exercice: Ecrire un algorithme qui Lit un entier positif n puis affiche tous les nombres impairs inférieurs à n

Solution

```
Algorithme ex1
  Variables i,n: entier
  Procédure Aug2(..)
        …
  Fin Procédure
Début
        Lire(n);
        i ← 1;
        Tant que (i ≤ n) Faire
              Ecrire(i);
              aug2(i);
        Fin TantQue
Fin
```

Procédure et fonction: appels imbriqués

- Dans la définition d'une procédure, on peut faire appel à une autre procédure ou fonction déjà définie . Même remarque pour les fonctions

Exemple:

```
Fonction Puiss4(n : entier) : entier
Début
        Retourne( carre(carre(n))) ;
FinFonction
```

Variables locales et globales (1)

- On peut manipuler 2 types de variables dans un module (procédure ou fonction) : des **variables locales** et des **variables globales**. Elles se distinguent par ce qu'on appelle leur **portée** (leur "champ de définition", leur "durée de vie")

- Une **variable locale** n'est connue qu'à l'intérieur du module ou elle a été définie. Elle est créée à l'appel du module et détruite à la fin de son exécution

- Une **variable globale** est connue par l'ensemble des modules et le programme principal. Elle est définie durant toute l'application et peut être utilisée et modifiée par les différents modules du programme

Variables locales et globales (2)

- La manière de distinguer la déclaration des variables locales et globales diffère selon le langage
  - En général, les variables déclarées à l'intérieur d'une fonction ou procédure sont considérées comme variables locales
- En pseudo-code, on va adopter cette règle pour les variables locales et on déclarera les variables globales dans le programme principale
- **Conseil :** Il faut utiliser autant que possible des variables locales plutôt que des variables globales. Ceci permet d'économiser la mémoire et d'assurer l'indépendance de la procédure ou de la fonction