INSTITUT AFRICAIN DE DJIBOUTI

Licence 1 : MRT & GI

Cours Algorithme avancée et structure des données.

❖ Programme du cours

Semestre étudié	Titre des Chapitres
S2	Chapitre 1 : Tableau à une dimension
	Chapitre 2 : Tableau à deux dimensions
	Chapitre 3 : Tableau à trois dimensions
	Chapitre 4 : Tableau à N dimensions
	Chapitre 5 : Trie sur les tableaux
	Chapitre 6 : Fonctions & Procédures

Cours Algorithme avancée et structure des données.

Chapitre 1 : Tableau à une dimension

Exemple introductif

Supposons qu'on veut conserver les notes d'une classe de 12 étudiants pour extraire quelques informations. Par exemple : calcul du nombre d'étudiants ayant une note supérieure à 10

Le seul moyen dont nous disposons actuellement consiste à déclarer 12 variables, par exemple **N1, …, N12**. Après 30 instructions lire, on doit écrire 30 instructions Si pour faire le calcul

Exemple calcul de la moyenne

- **Moy ← (N1+N2+N3+N4+N5+N6+N7+N8+N9+N10+N11+N12) / 12**

les langages de programmation offrent la possibilité de rassembler toutes les variables dans **une seule structure de donnée** appelée **tableau.**

- **Rassembler toutes ces variables en une seule**, au sein de laquelle chaque valeur sera désignée par un numéro

1. Définition

Un tableau est une structure de données permettant d'effectuer un même traitement sur des données de même nature.

tableau à **une** dimension

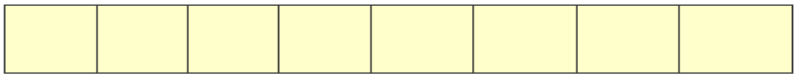
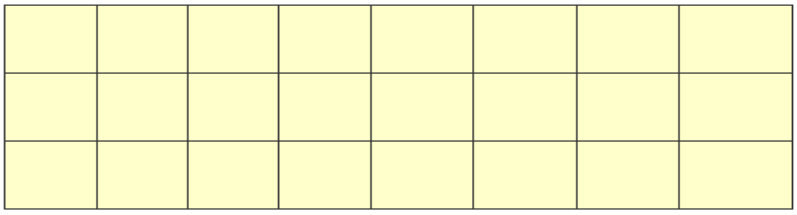
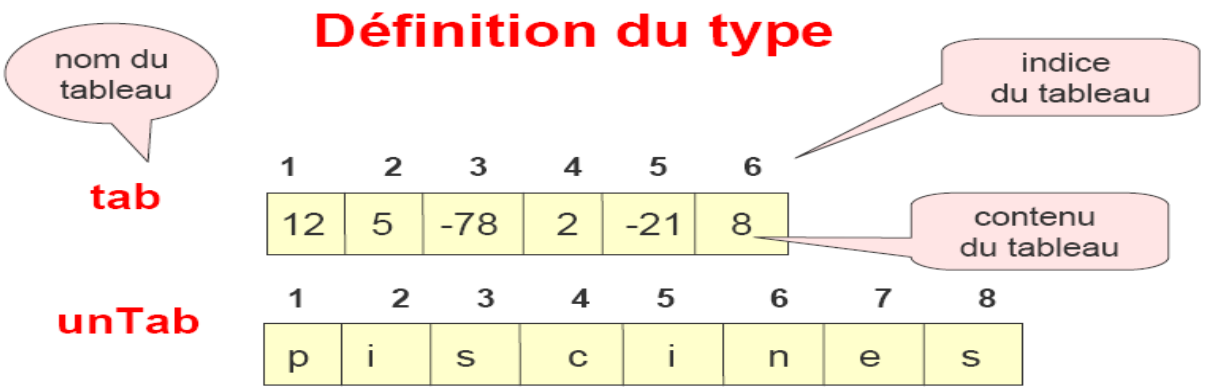


tableau à **deux** dimensions



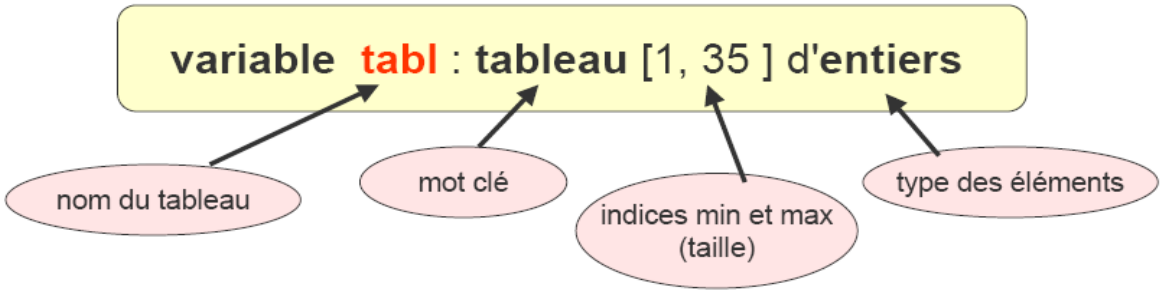
Cours Algorithme avancée et structure des données.



Remarques :
- Indices : en général, démarrage à 1, **mais en C++, démarrage à 0**
- Nombre d'octets occupés : dépend du type des valeurs enregistrées

2. *Déclaration d'un tableau*

Exemple : déclaration d'un tableau pouvant contenir jusqu'à 35 entiers



Autre exemple : déclaration d'un tableau qui contiendra les fréquences des températures comprises entre –40°C et 50°C

variable températures : tableau [-40, 50] de réels

- ✓ Un ***tableau*** possède un nom (ici note) et un nombre d'éléments (de cases) qui représente sa taille (ici 12).
- ✓ Tous les éléments d'un tableau ont le même type (c'est normal car ils représentent des valeurs logiquement équivalentes).
- ✓ Pour designer un élément, on indique le nom du tableau suivi son indice (son numéro) entre crochets:

3. *Les variables d'un tableau*

La notion de ≪case contenant une valeur ≫doit faire penser a celle de variable. Et, en effet, les cases du tableau, encore appelées éléments du tableau, sont des variables, qualifiées d'indices.

- ✓ *Différence entre variables classiques et variables indices:*

_ Les ***variables classiques*** sont déclarées individuellement et ont un nom distinct ;
_ Les ***variables indices*** (constituant le tableau) sont implicitement déclarées lors de la déclaration du tableau. Pour bien montrer que ces l'endroit qu'il faudra remplir.

❖ ***Utilisation d'un tableau : par les indices***

- Accès en lecture :
  - **afficher**(tabl[4]) *{le contenu du tableau à l'indice 4 est affiché à l' écran}*
- Accès en écriture :
  - tabl[3] ←18 *{la valeur 18 est placée dans le tableau à l'indice 3}*
  - **saisir**(tabl[5]) *{la valeur entrée par l'utilisateur est enregistrée dans le tableau à l'indice 5}*
  - attention!

~~tabl ← 18~~ ~~nom[2] ← 3~~

Remarque :
Dans un programme, chaque élément d'un tableau est repéré par un indice. Dans la vie courante, nous utilisons souvent d'autres façons de repérer une valeur. Par exemple, au lieu de parler de note [1], note [2], note [3], nous préférons parler des notes des étudiants. Le tableau ne permet pas de repérer ses valeurs autrement que par un numéro d'indice. Donc si cet indice n'a pas de signification, un tableau ne permet pas de savoir a quoi correspondent les différentes valeurs.

L'indice d'un élément peut être :

- *Directement une valeur ex : Note [10]*
- *Une variable ex : note [i]*
- *Une expression entière ex : note [k+1] avec k de type entier*

Quelque soit sa forme, la valeur de l'indice doit être :

- *entière*
- *comprise entre les valeurs minimales et maximales déterminées à la déclaration du tableau.*

4. ***Méthode d'écriture et de lecture d'un tableau***

✓ Méthode d'écriture :

Pour saisir (remplir) un tableau, il il existe deux façon de le faire :

Cours Algorithme avancée et structure des données.

SOIT JE CONNAIS LE NOMBRE DE CASE **(DIMENSION DU TABLEAU) DANS CE CAS:**

```
Ecrire (veuillez entrer les 12 notes dans le tableau)
Pour i allant de 1 à 12 faires
Lire (Note[i])
Fin pour
```

II. ***SOIT JE NE CONNAIS PAS LE NOMBRE DE CASE*** **(DIMENSION DU TABLEAU) ET DANS CE CAS :**

```
//Demande de saisir la dimension du tableau
Ecrire (veuillez entrer la dimension du tableau)
Lire (N)
//Ensuite on demande de saisir les valeurs du tableau
Ecrire (« veuillez entrer les 12 notes dans le tableau »)
Pour i allant de 1 à N faire
Lire (Note[i])
Fin pour
```

Fin du chapitre1

Chapitre 2 : Tableau à deux dimension

Introduction :

Une seule ne suffisait-elle pas déjà amplement à notre bonheur, me demanderez-vous ? Certes, répondrai-je, mais vous allez voir qu'avec deux (et davantage encore) c'est carrément insuffisant.

Alors Prenons le cas de la modélisation du saisi de 4 notes par étudiants, comment cela va se passer ?

Avec les outils que nous avons abordés jusque là, le plus simple serait évidemment de modéliser une note et de les remplir dans un tableau :

Exemple

unTab

1	2	3	4	5	6	7	8
p	i	s	c	i	n	e	s

Cours Algorithme avancée et structure des données.

❖ Définition

La création d'un tableau à deux dimensions (**aussi appelé matrice**) permet l'accès à plus d'une valeur à partir d'un indice. Autrement dit, un numéro d'index pointe sur un autre tableau.

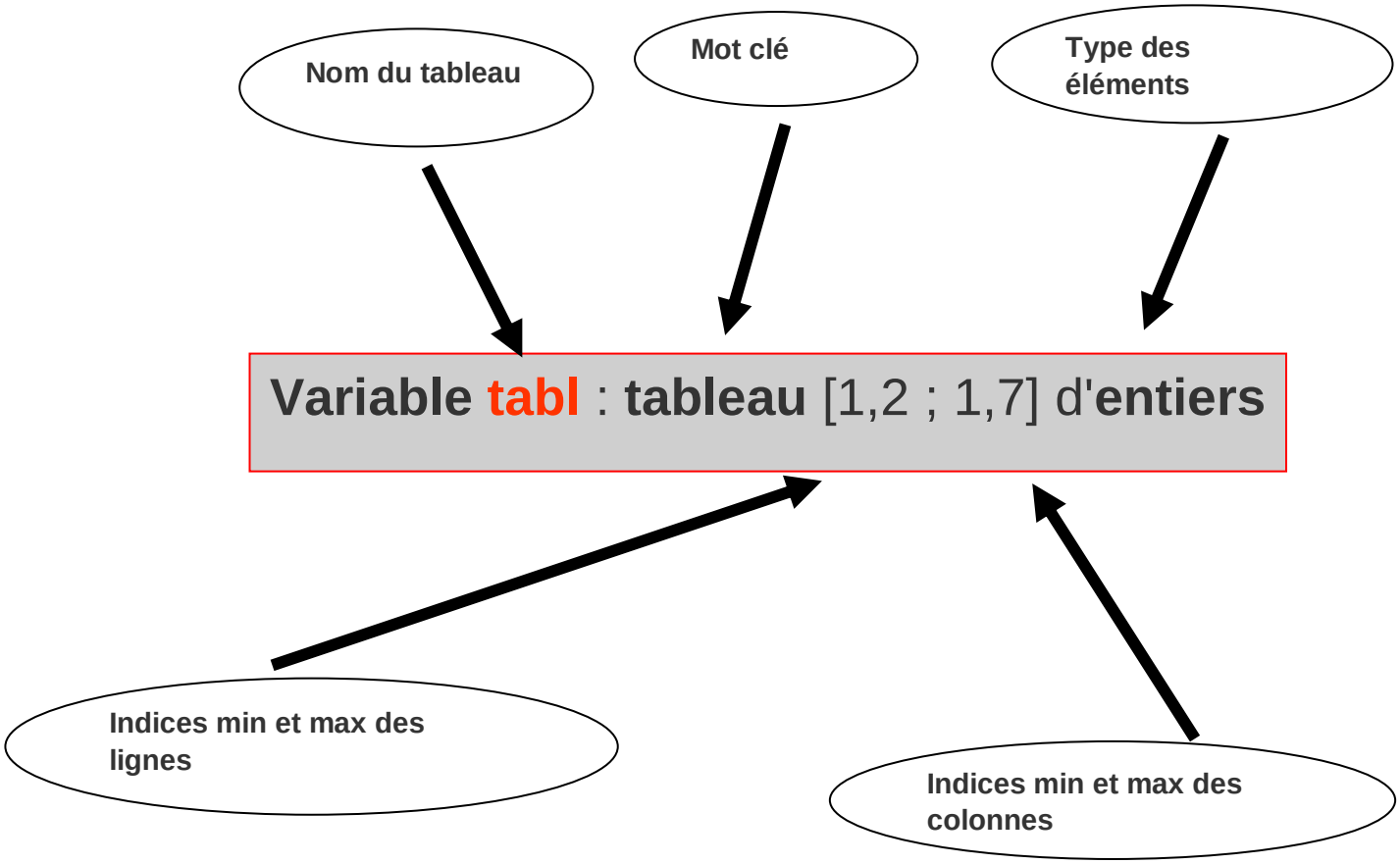
On représente un tableau à deux dimensions de cette manière :

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

❖ Déclaration :

L'informatique nous offre la possibilité de déclarer des tableaux dans lesquels les valeurs ne sont pas repérées par une seule, mais par **deux coordonnées**. Pour cela nous avons trois façons de les déclarer :

1er cas :



Exemple : points : **tableau [1,2 ; 1,7] d'entiers**

2eme cas :

Variable tabl : tableau [2 ; 7] d'**entiers**

C'est le même principe mais cette fois-ci ns déclarons que le maximum

Cela veut dire : réserve moi un espace de mémoire pour 8 x 8 entiers, et quand j'aurai besoin de l'une de ces valeurs, je les repèrerai par deux indices.

3eme cas :

tab[3][2];

Remarque importante :

Il n'y a aucune différence qualitative entre un tableau à deux dimensions (i, j) et un tableau à une dimension (i * j). Tout problème qui peut être modélisé d'une manière peut aussi être modélisé de l'autre. Simplement, l'une ou l'autre de ces techniques correspond plus spontanément à tel ou tel problème, et facilite donc (ou complique, si on a choisi la mauvaise option) l'écriture et la lisibilité de l'algorithme.

Cours Algorithme avancée et structure des données.

❖ *Structure d'un tableau à deux dimensions*

Exemple d'un algo de tableau à deux dimensions :

```
Algorithme SaisieTableau2D
{remplit un tableau à 2 dimensions }
constantes (TailleMAX : entier) ← 100
variables nbLignes, nbColonnes, indL, indC : entiers
nombres : tableau [1, TailleMAX ; 1, TailleMAX] d'entiers
début
afficher ("Combien de lignes?") ; saisir (nbLignes)
afficher ("Combien de colonnes?") ; saisir (nbColonnes)
si nbLignes > TailleMAX ou nbColonnes > TailleMAX
alors afficher ("trop de valeurs à saisir")
sinon pour indL ← 1 à nbLignes faire
pour indC ← 1 à nbColonnes faire
afficher ("Ligne" , inL, "colonne", indC, " : ")
saisir (nombres[indL indC])
fpour
fpour
fsi
fin
```

Chapitre 3 : Tableau à trois dimension

1. Introduction

Dans le chapitre 1, nous avons étudié le **tableau à une dimension** (liste de valeurs).

Dans le chapitre 2, nous avons étudié le **tableau à deux dimensions** (matrice : lignes × colonnes).

Mais dans certains problèmes réels, deux dimensions ne suffisent pas.

❓ Exemple concret

Supposons que nous voulons stocker :

- Les notes des étudiants
- Dans plusieurs matières
- Pendant plusieurs semestres

Nous avons donc :

- Dimension 1 → Étudiants
- Dimension 2 → Matières
- Dimension 3 → Semestres

Nous avons besoin d'un **tableau à trois dimensions**.

2. Définition

Un tableau à trois dimensions est une structure de données permettant de stocker des valeurs de même type, repérées par **trois indices**.

On l'appelle aussi :

- Tableau 3D
- Tableau tridimensionnel
- Cube de données

Chaque élément est repéré par :

```
Tab[i][j][k]
```

Où :

- i → première dimension
- j → deuxième dimension
- k → troisième dimension

3. Représentation d'un tableau 3D

On peut imaginer un tableau 3D comme :

Un cube composé de plusieurs matrices empilées.

Exemple :

Si nous avons :

```
Notes[3][4][2]
```

Cela signifie :

- 3 étudiants

- 4 matières
- 2 semestres

Donc :

```
Notes[étudiant][matière][semestre]
```

4. Déclaration d'un tableau à trois dimensions

✓ En pseudo-code

```
Notes : tableau [1,3 ; 1,4 ; 1,2] d'entiers
```

✓ En langage C

```
int Notes[3][4][2];
```

Cela signifie :

Réserve un espace mémoire pour :

3 × 4 × 2 = 24 entiers

5. Accès à un élément

Pour accéder à une valeur précise :

```
Notes[2][3][1]
```

Signifie :

- Étudiant 2
- Matière 3
- Semestre 1

Les indices doivent être :

- Entiers
- Compris dans les bornes déclarées

6. Méthode de saisie d'un tableau 3D

Pour remplir un tableau 3D, nous devons utiliser **trois boucles imbriquées**.

7. Algorithme complet : Saisie d'un tableau 3D

Objectif :

Remplir un tableau Notes contenant les notes des étudiants dans plusieurs matières et semestres.

Algorithme SaisieTableau3D

```
Algorithme SaisieTableau3D

Constantes
    MAXE ← 100
    MAXM ← 20
    MAXS ← 10

Variables
    nbEtudiants, nbMatieres, nbSemestres : entiers
    i, j, k : entiers
    Notes : tableau [1, MAXE ; 1, MAXM ; 1, MAXS] d'entiers



Début
```

```
    Ecrire("Donner le nombre d'étudiants : ")
    Lire(nbEtudiants)

    Ecrire("Donner le nombre de matières : ")
    Lire(nbMatieres)

    Ecrire("Donner le nombre de semestres : ")
    Lire(nbSemestres)

    Pour i allant de 1 à nbEtudiants faire
        Pour j allant de 1 à nbMatieres faire
            Pour k allant de 1 à nbSemestres faire
                Ecrire("Etudiant ", i,
                       " Matière ", j,
                       " Semestre ", k, " : ")
                Lire(Notes[i][j][k])
            FinPour
        FinPour
    FinPour

Fin
```

8. Exemple en langage C

```
#include <stdio.h>

int main() {

    int Notes[3][4][2];
    int i, j, k;




    for(i = 0; i < 3; i++) {
        for(j = 0; j < 4; j++) {
```

```
            for(k = 0; k < 2; k++) {
                printf("Etudiant %d, Matiere %d, Semestre %d : ",
                       i+1, j+1, k+1);
                scanf("%d", &Notes[i][j][k]);
            }
        }
    }

    return 0;
}
```

9. Différence entre 1D, 2D et 3D

Dimension	Représentation	Exemple
1D	Liste	Notes[10]
2D	Matrice	Notes[10][5]
3D	Cube	Notes[10][5][2]

10. Cas pratique

Exercice 1

Écrire un algorithme permettant de :

- Saisir les ventes d'un produit
- Pour 3 magasins
- Sur 4 mois
- Pour 2 années

Calculer : La vente totale d'un magasin donné.

11. Remarque

Il n'y a **aucune différence qualitative** entre :

- Un tableau 3D
- Et un tableau 1D contenant (i × j × k) éléments

Mais :

✓ Le tableau 3D facilite la compréhension

✓ Il correspond mieux aux problèmes réels

✓ Il rend l'algorithme plus lisible

12. Complexité

Si :

- n = taille première dimension
- m = taille deuxième dimension
- p = taille troisième dimension

Alors le nombre total d'éléments est :

`n × m × p`

Conclusion

Le **tableau à trois dimensions** constitue une extension naturelle des tableaux à une et deux dimensions. Il permet de modéliser des situations où les données dépendent simultanément de **trois paramètres différents**.

1. Vision conceptuelle

Un tableau 3D peut être vu comme :

- Une liste de matrices
- Un empilement de tableaux 2D
- Un cube de données

Si :

- n = taille de la première dimension
- m = taille de la deuxième dimension
- p = taille de la troisième dimension

Alors le nombre total d'éléments est :

n×m×p
n × m × p
n×m×p

Cela signifie que la mémoire nécessaire augmente proportionnellement au produit des trois dimensions.

2. Complexité algorithmique

Pour parcourir entièrement un tableau à trois dimensions, on utilise trois boucles imbriquées.

La complexité temporelle est :

O(n×m×p) O(n × m × p) O(n×m×p)

Cela montre que :

- Si une dimension double, le temps d'exécution augmente proportionnellement.
- Si les trois dimensions augmentent, le coût devient rapidement important.

Plus la dimension augmente, plus il faut faire attention à :

- L'utilisation mémoire
- La performance
- L'optimisation des boucles

3. Applications concrètes

Le tableau 3D est très utilisé dans :

Domaine académique

- Notes (Étudiants × Matières × Semestres)
- Présence (Étudiants × Jours × Séances)

Réseaux et informatique (intéressant pour vos étudiants en Data & Réseaux)

- Trafic (Serveur × Protocole × Temps)
- Logs (Utilisateur × Action × Date)

Gestion commerciale

- Ventes (Magasin × Produit × Mois)
- Stock (Entrepôt × Catégorie × Année)

Graphisme et Jeux vidéo

- Pixels (x × y × profondeur)
- Cartes 3D

4. Avantages pédagogiques

Le tableau 3D permet aux étudiants de :

✓ Comprendre la généralisation des structures de données
✓ Maîtriser les boucles imbriquées
✓ Développer la logique spatiale
✓ Comprendre la notion de dimension en informatique

Il prépare également à des concepts avancés comme :

- Les bases de données
- Les structures matricielles avancées
- Le Big Data
- Les tenseurs (utilisés en Intelligence Artificielle)

5. Comparaison globale

Dimension	Indices	Structure	Complexité
1D	1 indice	Liste	O(n)
2D	2 indices	Matrice	O(n × m)
3D	3 indices	Cube	O(n × m × p)

On remarque une progression logique :

Chaque nouvelle dimension ajoute :

- Un indice supplémentaire
- Une boucle supplémentaire
- Une complexité supplémentaire

6. Remarque importante (niveau avancé)

Mathématiquement, un tableau 3D n'est qu'un tableau 1D linéarisé en mémoire.

Par exemple :

```
int T[3][4][2];
```

Est stocké en mémoire comme une suite continue de :

3 × 4 × 2 éléments.

Mais la notation 3D facilite :

- La compréhension
- La modélisation
- La lisibilité de l'algorithme

Conclusion Générale du Chapitre

Le tableau à trois dimensions est une structure puissante permettant de représenter des données complexes dépendant de trois paramètres.

Il constitue :

- Une généralisation logique des tableaux précédents
- Un outil essentiel en algorithmique avancée
- Une base fondamentale pour les structures multidimensionnelles modernes

La maîtrise des tableaux 3D marque une étape importante vers :

- La programmation scientifique
- L'analyse de données
- L'intelligence artificielle
- Les systèmes complexes

Chapitre 4 : Tableau à N dimension

1. Introduction

Après avoir étudié :

- Tableau à 1 dimension (liste)
- Tableau à 2 dimensions (matrice)
- Tableau à 3 dimensions (cube)

Nous pouvons généraliser ce concept vers un **tableau à N dimensions**. Dans certains problèmes complexes (scientifiques, statistiques, intelligence artificielle), les données dépendent de **plus de trois paramètres**.

? Exemple concret

Supposons que nous voulons stocker :

- Étudiants
- Matières
- Semestres
- Années
- Centres d'examen

Nous avons alors :

Dimension 1 → Étudiants
Dimension 2 → Matières
Dimension 3 → Semestres
Dimension 4 → Années
Dimension 5 → Centres

➞ Nous avons besoin d'un tableau à 5 dimensions.

2. Définition

Un tableau à N dimensions est une structure de données permettant de stocker des éléments de même type, repérés par **N indices**.

Chaque élément est repéré par :

```
Tab[i1][i2][i3]...[iN]
```

Où :

- i1 → 1ère dimension
- i2 → 2ème dimension
- …
- iN → Nème dimension

3. Nombre total d'éléments

Si un tableau possède :

- n_1 éléments dans la 1ère dimension
- n_2 éléments dans la 2ème dimension
- …
- n_n éléments dans la Nème dimension

Alors le nombre total d'éléments est :

$n1 \times n2 \times n3 \times ... \times nn$

Cela montre que la mémoire augmente très rapidement lorsque le nombre de dimensions augmente.

4. Complexité algorithmique

Pour parcourir entièrement un tableau à N dimensions, il faut utiliser **N boucles imbriquées**.

La complexité devient :

$O(n1 \times n2 \times ... \times nn)$

Plus N augmente :

- Plus le nombre d'opérations augmente
- Plus le programme devient coûteux en temps d'exécution

5. Représentation conceptuelle

Dimension **Représentation**

Dimension	Représentation
1D	Ligne
2D	Matrice
3D	Cube
4D	Hypercube
ND	Hyperstructure multidimensionnelle

Au-delà de 3 dimensions, la représentation devient abstraite (on parle d'**hypercube**).

6. Déclaration d'un tableau à N dimensions

En pseudo-code

```
T : tableau [1,n1 ; 1,n2 ; ... ; 1,nN] d'entiers
```

En langage C (exemple 4D)

```
int T[5][4][3][2];
```

Cela signifie :

5 × 4 × 3 × 2 éléments.

7. Parcours général d'un tableau à N dimensions

Principe général :

Pour N dimensions → N boucles imbriquées

Exemple pour 4 dimensions :

```
Pour i1 allant de 1 à n1
   Pour i2 allant de 1 à n2
```

```
      Pour i3 allant de 1 à n3
         Pour i4 allant de 1 à n4
              traiter T[i1][i2][i3][i4]
         FinPour
      FinPour
   FinPour
FinPour
```

8. Linéarisation en mémoire (Notion avancée)

En réalité, en mémoire, un tableau à N dimensions est stocké comme **un tableau à une seule dimension**.

L'adresse d'un élément est calculée par une formule mathématique basée sur les dimensions.

Cela montre que :

✓ Toutes les dimensions sont une abstraction logique
✓ Physiquement, la mémoire reste linéaire

9. Applications modernes

Les tableaux à N dimensions sont très utilisés dans :

Intelligence Artificielle

- Tenseurs (Deep Learning)
- Réseaux neuronaux

Big Data

- Données multidimensionnelles
- Data warehouse

Simulation scientifique

- Physique
- Météorologie
- Modélisation 3D/4D

Analyse réseau (intéressant pour vos étudiants Data & Réseaux)

Exemple :
Trafic[Serveur][Protocole][Heure][Jour]

10. Limites des tableaux à N dimensions

Lorsque N devient grand :

- Le code devient difficile à lire
- La mémoire explose rapidement
- Les performances diminuent

C'est pourquoi en pratique :

✓ On utilise des structures dynamiques
✓ On utilise des bases de données
✓ On utilise des structures spécialisées (tenseurs, structures creuses)

Conclusion Générale : Tableau à N Dimensions

Le tableau à N dimensions représente la généralisation ultime des tableaux classiques.

Il permet de modéliser des systèmes complexes dépendant de plusieurs paramètres simultanément.

Cependant :

- L'augmentation du nombre de dimensions entraîne une augmentation exponentielle du nombre d'éléments.
- La complexité algorithmique devient rapidement importante.
- La gestion mémoire doit être maîtrisée.

La compréhension des tableaux à N dimensions prépare les étudiants à :

- L'algorithmique avancée
- L'optimisation
- La programmation scientifique
- L'intelligence artificielle
- Les systèmes de données complexes

Chapitre 5 : Trie sur les tableaux

Définition

Un **algorithme de tri** est, en informatique ou en mathématiques, un algorithme qui permet d'organiser une collection d'objets selon un ordre déterminé. Les objets à trier font donc partie d'un ensemble muni d'une relation d'ordre (de manière générale un ordre total). Les ordres les plus utilisés sont l'ordre numérique et l'ordre lexicographique (dictionnaire).

Par ailleurs, parmi les algorithmes listés plus bas, les tris étant stables sont : le tri à bulles, le tri par insertion et le tri à sélection. Les autres algorithmes nécessitent O(n) mémoire supplémentaire pour stocker l'ordre initial des éléments. Pour cela nous verrons trois types d'algorithmes :

Le tri par insertion (facultative)

Le principe du tri par insertion est d'insérer à la n-ième itération le n-ième élément à la bonne place.
La démo ci-après détaille le fonctionnement du tri par insertion :

Voici, en pseudo-code, l'algorithme du tri par insertion :

```
tri_insertion(tableau T)
  debut
    entier longueur, i, memoire, compteur;
    booleen marqueur;
    longueur<-taille(T)
        pour i=1 à (longueur-1) faire




memoire<-T(i) //valeur à insérer au tour i
        compteur<-(i-1)
```

```
        faire
    marqueur=faux //on n'a pas fait de décalage
    si T(compteur)>memoire alors
      T(compteur+1)<-T(compteur) //décalage des plus grandes valeurs du tableau
      compteur<-compteur-1
      marqueur=vrai //on vient de faire un décalage
    fin si
    si (compteur<0) alors //on a atteint la premiére valeur du tableau
      marqueur=faux //il n'y a plus de décalages possibles
    fin si
  tantque marqueur

  T(compteur+1)<-memoire //affectation de la valeur à insérer dans la bonne case
 fin pour
fin
```

Le tri par sélection

Le principe du tri par sélection / échange (ou *tri par extraction*) est d'aller chercher le plus petit élément du vecteur pour le mettre en premier, puis de repartir du second élément et d'aller chercher le plus petit élément du vecteur pour le mettre en second, etc...

Principe : soit un tableau de n valeurs

- Recherche du minimum dans le tableau et échange du contenu d'indice 1 et d'indice correspondant à la valeur du minimum.
- Applications du même principe sur (n - 1) valeur (n - premier) pour (n - 1), ... jusqu'à traitement de 2 cases.

Cours Algorithme avancée et structure des données.

Code source tri par "selection"

```
Var       valeur TABLEAU [1..n] D'Entier
          inter, j, indmin, i : réel

Début
   |  i ← 1
   |  Pour i de 1 à n Faire
   |     |  Afficher (« Donner un nombre »)
   |     |  Saisir (valeur [i])
   |  Fin pour
   |  Pour i de 1 à (n – 1) Faire
   |     |  indmin ← i
   |     |  Pour i de 1 à (n + 1) Faire
   |     |     |    Si  valeur [j] < valeur [indmin] Alors
   |     |     |     |  indmin ← j
   |     |     |   FSI
   |     |  Fin pour
   |     |   Si  indmin <> i Alors
   |     |     |  inter ← valeur [i]
   |     |     |  valeur [i] ← valeur [indmin]
   |     |     |  valeur [indmin] ← inter
   |     |  FSI
   |  Fin pour
 Fin
```

Exemple :

8	1	7	5	4
1	8	7	5	4
1	4	7	5	8
1	4	5	7	8
1	4	5	7	8

Le tri "bulle"

Le principe du tri bulle (*bubble sort*) est de comparer deux à deux les éléments e1 et e2 consécutifs d'un tableau et d'effecteur une permutation si e1 > e2. On continue de trier jusqu'à ce qu'il n'y ait plus de permutation. La démo ci-après détaille le fonctionnement du tri bulle :

Code source du tri "bulle"

```
var     valeur TABLEAU [1..n] D Entier
        echange : booléen /*détermine si un échange de valeurs a été effectué*/
        inter : réel

Début
  | /* Saisie du tableau */
  | i ← 1
  | Pour i de 1 à n Faire
  |   | Afficher (« Donner un nombre »)
  |   | Saisir (valeur [i])
  | Fin pour
  | /* Tri du tableau */
  | Répéter
  |   | échange ← Faux
  |   | Pour i de 1 à (n – 1) Faire
  |   |   | si valeur [i] > valeur [i + 1] alors
  |   |   |   | échange ← Vrai
  |   |   |   | inter ← valeur [i]
  |   |   |   | valeur [i] ← valeur [i + 1]
  |   |   |   | valeur [i + 1] ← inter
  |   |   | FSI
  |   | Fin pour
  | Jusqu'à non échange
Fin
```

Cours Algorithme avancée et structure des données.

Exemple : Soit le tableau suivant à trier par ordre croissant

5	**18**	14	4	26
5	**18**	**14**	4	26
5	14	**18**	**4**	26
5	14	4	**18**	**26**
5	**14**	4	18	26
5	**14**	**4**	18	26
5	4	**14**	**18**	26
5	4	14	**18**	**26**
5	**4**	14	18	26
4	**5**	**14**	18	26
4	5	**14**	**18**	26
4	5	14	**18**	**26**
4	**5**	14	18	26
4	5	14	18	26

4	5	**14**	**18**	26
4	5	14	**18**	**26**

Il n'y a plus d'échange.

Recherche d'un élément par dichotomie dans un tableau à tri

Principe : Sur un tableau délimité par les indices borneinf et bornesup, ordonné de manière croissante, le principe de recherche est le suivant :

- Recherche de l'élément situé à l'indice médian du tableau (milieu du tableau)

- Effectuer une comparaison entre l'élément recherché et l'élément médian. Trois cas peuvent se présenter :
  - élément à chercher = élément médian : arrêt du traitement.
  - élément à chercher < élément médian : on modifie la borne supérieure de recherche dans le tableau (bornesup ← indice médian - 1) car l'élément à chercher est obligatoirement (s'il existe) dans la partie gauche du tableau.
  - élément à chercher > élément médian : on modifie la borne inférieure de recherche dans le tableau (borneinf ← indice médian + 1) car l'élément à chercher est obligatoirement (s'il existe) dans la partie droite du tableau.

Les conditions d'arrêt du traitement sont :

- égalité entre la valeur cherchée et élément médian.

- la borne supérieure est devenue inférieure à la borne inférieure car si la valeur n'existe pas, le tableau a été réduit à 0 case.

Cours Algorithme avancée et structure des données.

Exemple 1 : Soit le tableau

1	4	5	7	8

et l'élément à chercher = 7

1	4	5	7	8
borneinf		median		bornesup

élément à chercher > élément médian (7 > 5)
borneinf ← indice médian + 1

1	4	5	7	8
			borneinf médian	bornesup

élément à chercher = élément médian (7 = 7)

Exemple 2 : Soit le tableau

1	4	5	7	8

et l'élément à chercher = 3

élément à chercher < élément médian (3 < 5)
bornesup ← indice médian - 1

1	4	5	7	8
Borneinf		médian		bornesup

1	4	5	7	8
borneinf médian	bornesup			

élément à chercher > élément médian (3 > 1)
borneinf ← indice médian + 1

1	4	5	7	8
	borneinf bornesup médian			

élément à chercher < élément médian (3 < 4)
bornesup ← indice médian -1

1	4	5	7	8
bornesup	borneinf			

3 n'est pas dans le tableau.

Algorithme

Programme Dichotomie

```
Const   n = vous ne vous 5
Var     valeur TABLEAU [1..n] D'Entier
        borneinf, bornesup, median, element : entier

Début
   | /* Saisie d'un élément a rechercher */
   | Afficher (« aucune valeur »)
   | Saisir (élément)
   | /* Recherche de l'élément */
   | Tantque (element <> valeur [median]) et (borneinf < = bornesup) Faire
   |    |       /* Comparaison */
   |    |   Si (element < valeur[median]) alors
   |    |    | bornesup ← (median – 1)
   |    | sinon
   |    |    | borneinf ← (median + 1)
   |    |  FSI
   |    | median ← (borneinf + bornesup) DIV 2
   | FTQ
   |       /* Résultat de la recherche */
   |   Si (element = valeur[median]) alors
   |    | Afficher ( element, « trouvé à l'indice médian », median, « . »)
   | sinon
   |    | Afficher ( « L'élément »,element, « n'est pas dans le tableau. »)
   | FSI
FIN
```

Chapitre 6 : Fonctions & Procédures

Introduction

Lorsque l'on progresse dans la conception d'un algorithme, ce dernier peut prendre une taille et une complexité croissante. De même des séquences d'instructions peuvent se répéter à plusieurs endroits.

Un algorithme écrit d'un seul tenant devient difficile à comprendre et à gérer dès qu'il dépasse deux pages. La solution consiste alors à découper l'algorithme en plusieurs parties plus petites. Ces parties sont appelées des sous-algorithmes.

Le sous-algorithme est écrit séparément du corps de l'algorithme principal et sera appelé par celui-ci quand ceci sera nécessaire.

Il existe deux sortes de sous-algorithmes : les procédures et les fonctions.

Les procédures

Une procédure est une série d'instructions regroupées sous un nom, qui permet d'effectuer des actions par un simple appel de la procédure dans un algorithme ou dans un autre sous-algorithme.
Une procédure renvoie plusieurs valeurs (par une) ou aucune valeur.

Déclaration d'une procédure

Syntaxe :

```
Procédure nom_proc(liste de paramètres)
Variables identificateurs : type
Début
        Instruction(s)
FinRoc
```

Après le nom de la procédure, il faut donner la liste des paramètres (s'il y en a) avec leur type respectif. Ces paramètres sont appelés paramètres formels. Leur valeur n'est pas connue lors de la création de la procédure.

Exemple :

Ecrire une procédure qui affiche à l'écran une ligne de 15 étoiles puis passe à la ligne suivante.

Solution :

```
Procédure Etoile()
Variables i : entier
Début
         Pour i Allant de 1 à 15 faire
             Afficher("*")
         FinPour
         //\n : retour à la ligne
         Afficher("\n")
FinProc
```

L'appel d'une procédure

Pour déclencher l'exécution d'une procédure dans un programme, il suffit de l'appeler.
L'appel de procédure s'écrit en mettant le nom de la procédure, puis la liste des paramètres, séparés par des virgules.
A l'appel d'une procédure, le programme interrompt son déroulement normal, exécute les instructions de la procédure, puis retourne au programme appelant et exécute l'instruction suivante.

Syntaxe :

```
Nom_proc(liste de paramètres)
```

Les paramètres utilisées lors de l'appel d'une procédure sont appelés paramètres effectifs. Ces paramètres donneront leurs valeurs aux paramètres formels.

Exemple :
En utilisant la procédure Etoiles déclarée dans l'exemple précédent, écrire un algorithme permettant de dessiner un carré d'étoiles de 15 lignes et de 15 colonnes.

Solution :

```
Algorithme carré_étoiles
Variables j : entier

//Déclaration de la procédure Etoiles()
Procédure Etoile()
Variables i : entier
Début
   Pour i Allant de 1 à 15 Faire
        Afficher("*")
   FinPour
   Afficher("/n")
FinProc

//Algorithme principal (Partie principale)
Début
     Pour j Allant de 1 à 15 Faire
            //Appel de la procédure Etoiles
            Etoile()
     FinPour
Fin
```

Remarque 1 :

Pour exécuter un algorithme qui contient des procédures et des fonctions, il faut commencer l'exécution à partir de la partie principale (algorithme principal)

Remarque 2 :

Lors de la conception d'un algorithme deux aspects apparaissent :

La définition (déclaration) de la procédure ou fonction.
L'appel de la procédure ou fonction au sein de l'algorithme principal.

Passage de paramètres

Les échanges d'informations entre une procédures et le sous algorithme appelant se font par l'intermédiaire de paramètres.
Il existe deux principaux types de passages de paramètres qui permettent des usages différents :
Passage par valeur :
Dans ce type de passage, le paramètre formel reçoit uniquement une copie de la valeur du paramètre effectif.
La valeur du paramètre effectifs ne sera jamais modifiée.

Exemple :
Soit l'algorithme suivant :

```
Algorithme Passage_par_valeur
Variables N : entier

//Déclaration de la procédure P1
Procédure P1(A : entier)
Début
    A ← A * 2
    Afficher(A)
FinProc

//Algorithme principal
Début
   N ← 5
   P1(N)
   Afficher(N)
Fin
```

Cet algorithme définit une procédure P1 pour laquelle on utilise le passage de paramètres par valeur.
Lors de l'appel de la procédure, la valeur du paramètre effectif N est recopiée dans le paramètres formel A. La procédure effectue alors le traitement et affiche la valeur de la variable A, dans ce cas 10.
Après l'appel de la procédure, l'algorithme affiche la valeur de la variable N dans ce cas 5.
La procédure ne modifie pas le paramètre qui est passé par valeur.

Passage par référence ou par adresse :
Dans ce type de passage, la procédure ***utilise l'adresse*** du paramètre effectif. Lorsqu'on utilise l'adresse du paramètre, ***on accède directement à son contenu***. La valeur de la variable effectif sera donc modifiée.

Les paramètres passés par adresse sont précédés du mot clé ***Var***.

Exemple :
Reprenons l'exemple précédent :

```
Algorithme Passage_par_référence
Variables N : entier

//Déclaration de la procédure P1
Procédure P1 (Var A : entier)
Début
    A ← A * 2
    Afficher(A)
FinProc

//Algorithme Principal
Début
      N ← 5
      P1(N)
      Afficher(N)
Fin
```

A l'exécution de la procédure, l'instruction ***Afficher(A)*** permet d'afficher à l'écran 10. Au retour dans l'algorithme principal, l'instruction ***Afficher(N)*** affiche également 10.
Dans cet algorithme le paramètre passé correspond à la référence (adresse) de la variable N. Elle est donc modifiée par l'instruction :
`A ← A *2`

Remarque :
Lorsqu'il y a plusieurs paramètres dans la définition d'une procédure, il faut absolument qu'il y en ait le même nombre à l'appel et que l'ordre soit respecté.

Les fonctions

Les fonctions sont des sous algorithmes admettant des paramètres et retournant un seul résultat (une seule valeur) de type simple qui peut apparaître dans une expression, dans une comparaison, à la droite d'une affectation, etc.

Déclaration d'une fonction

Syntaxe :

```
Fonction nom_Fonct (liste de paramètres) : type
Variables identificateur : type
Début
        Instruction(s)
        Retourner Expression
Fin
```

La syntaxe de la déclaration d'une fonction est assez proche de celle d'une procédure à laquelle on ajoute un type qui représente le type de la valeur retournée par la fonction et une instruction Retourner Expression. Cette dernière instruction renvoie au programme appelant le résultat de l'expression placée à la suite du mot clé Retourner.

Note :
Les paramètres sont facultatifs, mais s'il n'y pas de paramètres, les parenthèses doivent rester présentes.

Exemple :
Définir une fonction qui renvoie le plus grand de deux nombres différents.

Solution :

```
//Déclaration de la fonction Max
```

```
Fonction Max(X: réel, Y:réel) : réel
Début
      Si  X > Y Alors
            Retouner X
      Sinon
            Retouner Y
      FinSi
FinFonction
```

L'appel d'une fonction

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom suivie des paramètres effectifs. C'est la même syntaxe qu'une procédure.
A la différence d'une procédure, la fonction retourne une valeur. L'appel d'une fonction pourra donc être utilisé dans une instruction (affichage, affectation, ...) qui utilise sa valeur.

Syntaxe

```
Nom_Fonc(list de paramètres)
```

Exemple :

Ecrire un algorithme appelant, utilisant la fonction Max de l'exemple précédent.

Solution :

```
Algorithme Appel_fonction_Max
Variables A, B, M : réel

//Déclaration de la fonction Max
Fonction Max(X: réel, Y: réel) : réel
Début
        Si X > Y Alors
             Retourner X
        Sinon
             Retourner Y
        FinSi
FinFonction

//Algorithme principal
Début
        Afficher("Donnez la valeur de A :")
        Saisir(A)
        Afficher("Donnez la valeur de B :")
        Saisir(B)
        //Appel de la fonction Max
        M ← Max(A,B)
        Afficher("Le plus grand de ces deux nombres est : ", M)
Fin
```

Portée des variables

La portée d'une variable désigne le domaine de visibilité de cette variable. Une variable peut être déclarée dans deux emplacements distincts.

Cours Algorithme avancée et structure des données.

Une variable déclarée dans la partie déclaration de l'algorithme principale est appelée variable globale. Elle est accessible de n'importe où dans l'algorithme, même depuis les procédures et les fonctions. Elle existe pendant toute la durée de vie du programme.
Une variable déclarée à l'intérieur d'une procédure (ou une fonction) est dite locale. Elle n'est accessible qu'à la procédure au sein de laquelle elle définie, les autres procédures n'y ont pas accès. La durée de vie d'une variable locale est limitée à la durée d'exécution de la procédure.

```
Algorithme Portée
Variables X, Y : Entier
Procédure P1()
Variables A : Entier
Début
....
FinProc
//Algorithme principal
Début
....
Fin
```

X et Y sont des variables globales visibles dans tout l'algorithme.

A est une variables locale visibles uniquement à l'intérieur de la procédure

Remarque :
Les variables globales sont à éviter pour la maintenance des programmes.

Bibliographie

- **Introduction à l'Algorithmique.**
- **Algorithme Avancé et Structures de Données**.

1. 🕸 Web :

- *AlgoCours* — Cours et exercices d'algorithmique (structures de données, tris, complexité, etc.) — https://www.algocours.fr
- *France-IOI* — Ressources interactives pour apprendre les algorithmes — https://www.france-ioi.org
- *OpenClassrooms* — Cours d'algorithmique et programmation — https://openclassrooms.com/fr/courses/
- *GeeksforGeeks (Français)* — Explications et exemples sur les structures de données (listes, piles, files, arbres…) — https://fr.geeksforgeeks.org
- *AlgoExpert* — Concepts d'algorithmique expliqués étape par étape — https://www.algoexpert.io (ressources anglophones souvent utiles)

2. 📓 Livres :

a) 📌 Pour Introduction à l'Algorithmique

1. **« Introduction à l'algorithmique »** — *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein* — Dunod
   → Référence classique pour comprendre les bases de l'algorithmique, les structures de données, la complexité.
2. **« Algorithmique — Cours avec 957 exercices et 158 problèmes »** — *Claude Delannoy* — Ellipses
   → Avec nombreux exercices corrigés.

3. **« Initiation à l'algorithmique »** — *Nicolas Bertin* — Dunod
   → Approche pédagogique avec exemples et exercices pour débutants.

b) 📌 Pour Algorithme Avancé & Structures de Données

4. **« Structures de données et algorithmique en Java / C / Python »** — *Jean-Claude Feillet* — Dunod
   → Décompose les structures de données avancées avec implémentations.
5. **« Algorithmique avancée »** — *Michel Goossens* — Eyrolles
   → Concepts avancés (graphes, optimisation, récursivité complexe).
6. **« The Algorithm Design Manual »** — *Steven S. Skiena* — Springer (version française disponible)
   → Approche pratique de la conception algorithmique.
7. **« Algorithms »** — *Robert Sedgewick & Kevin Wayne* — Pearson (traduction française possible)
   → Structures de données et algorithmes appliqués avec exemples semblables à la pratique.