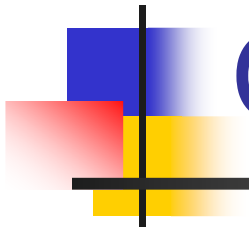
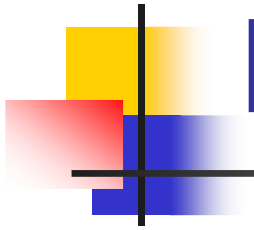
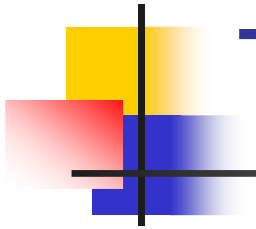


Cours 3: **Tableaux**



Les différentes structures

- Variable scalaire
- **Tableau**
- Structure de données linéaire
  - Liste
  - Pile
  - File
- Structure de données non linéaire
  - Arbre
  - Graphe

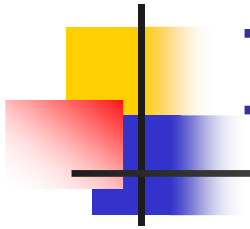


Tableau

- Regrouper dans une structure plusieurs valeurs *du même type*
- Déclaration

  A[début : fin] : **Tableau de T**

  - nom du tableau : A
  - taille du tableau : fin – début + 1
  - type des éléments : T
  - indices i: début ≤ i ≤ fin
- A[n] est une notation courte pour A [0 : n-1]



Initialisation

Variables mois[12] : **Tableau de Chaîne**

mois[0] ← "janvier"
mois[1] ← "février"
…
mois[11] ← "décembre"

mois ← {"janvier", "février", "mars", "avril", "mai",
"juin", "juillet", "août", "septembre",
"octobre", "novembre", "décembre"}

Boucler pour initialiser

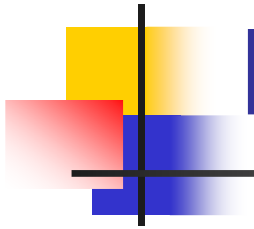
- Déclarer et remplir un tableau de 1000 valeurs en les mettant toutes à zéro.

```
Programme InitTab
Variables
        tab[1000] : Tableau de Entier
        i : Entier
Début
        Pour i ← 0 à 999
            tab[i] ← 0
        FinPour
Fin
```

Passage à une méthode

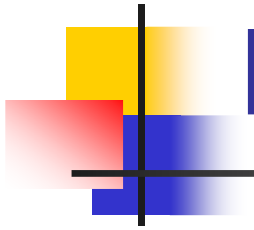
```
Fonction moyenne(tab[n]: Tableau de Réel): Réel
Variables
        i : Entier
        s : Réel
Début
        s ← 0
        Pour i ← 0 à n-1
            s ← s + tab[i]
        FinPour
        Retourner s/n
Fin
```

Appel: m ← moyenne(tab[n])



Exercice

- Ecrire une procédure qui reçoit un tableau d'entiers et affiche ses valeurs dans l'ordre inverse.



Exercice

```
Procédure affichageInverse(tab[n] : Tableau de Entier)
Variables i: Entier
Début
    Ecrire " L'ordre inverse de votre tableau: "
    Pour i ← n-1 à 0 pas -1
        Ecrire tab[i]
    FinPour
Fin
```

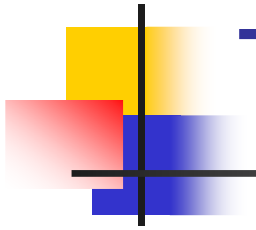
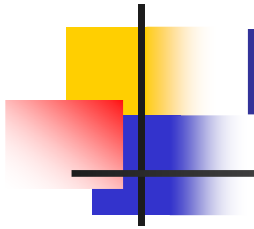


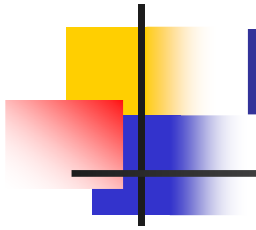
Tableau multidimensionnel

- Tableau à 2 dimensions
  - A[début$_1$: fin$_1$, début$_2$: fin$_2$] : **Tableau de T**
  - A[n_1 , n_2] : **Tableau de T**
  - Exemple : damier
    - cases[10,10] : **Tableau de Entier**
    - Les cases voisines de cases[i,j] sont : cases[i-1,j], cases[i+1,j], cases[i,j-1], cases[i,j+1]
- Tableau à n dimensions
  - A[début$_1$: fin$_1$, …, début$_n$: fin$_n$] : **Tableau de T**



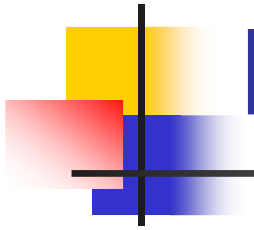
Exemple

```
Programme Dimension2
Variables X[2, 5] : Tableau de Entier
                  i, j, val : Entier
Début
      val ← 1
      Pour i ← 0 à 1
          Pour j ← 0 à 4
                X[i, j] ← val
                val ← val + 1
          FinPour
      FinPour
Fin
```



Exercice

- Ecrire une fonction qui reçoit deux vecteurs à n dimensions (tableaux de réels) et renvoie leur produit scalaire.



Exercice

```
Fonction prodScal(v[n], w[n]: Tableau de Réels): Réel
   Variables
        i : Entier
        s : Réel
   Début
        s ← 0
        Pour i ← 0 à n-1
            s ← s + v[i]*w[i]
        FinPour
        Retourner s
   Fin
```

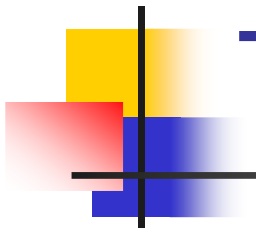
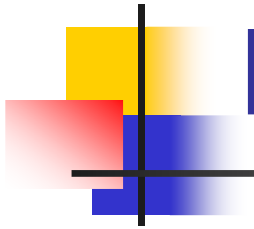


Tableau dynamique

- Quand on ne connaît pas à l'avance le nombre d'éléments d'un tableau :
  - Déclarer le tableau sans préciser au départ son nombre d'éléments

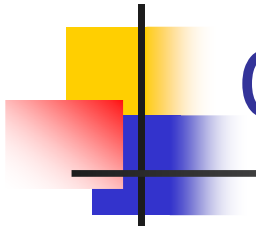
    tab[] : **Tableau de « type »**
  - Utiliser la fonction **CréerTableau** pour créer un tableau de taille quelconque.

    CréerTableau (nombre de cases, type)



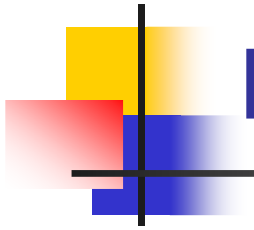
Exemple

```
Programme Saisie
Variables tab[] : Tableau de Entier, nb : Entier
Début
    Ecrire "Combien y a-t-il d'entiers à saisir ?"
    Lire nb
    tab ← CréerTableau(nb, Entier)
    Pour i ← 0 à nb-1
        Ecrire "Entrez l'entier numéro ", i
        Lire tab[i]
    FinPour
Fin
```



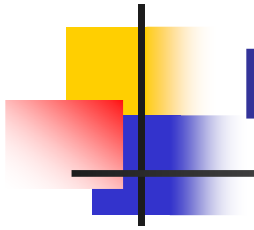
Quelques algorithmes avec tableau

1. Le plus grand élément
2. Recherche séquentielle
3. Recherche dichotomique
4. La plus grande somme



Le plus grand élément

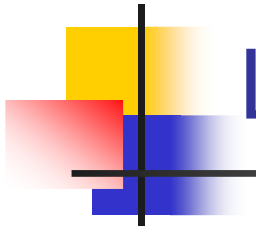
```
Fonction getMax(tab[n] : Tableau de Entier) : Entier
Variables max, i : Entier
Début
    max ← tab[0]                                    (1)
    Pour i ← 1 à n-1                                (2)
        Si (tab[i] > max) Alors                     (3)
            max ← tab[i]                            (4)
        FinSi
    FinPour
    Retourner max                                   (5)
Fin
```



Le plus grand élément

Compléxité en nombre d'affectations

- (1): 1 affectation
- (2) dépend de (3), (3) dépend de (4)
- (4): 1 affectation
- (5): 0 affectations



Le plus grand élément

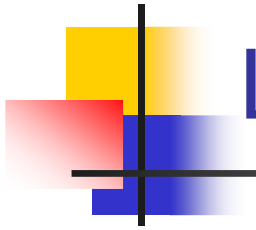
- (3): test
  - Pire cas : (3): 1 affectation
  - Meuilleur cas : (3): 0 affectations
  - Cas moyen : (3): 1/2 affectation

Le plus grand élément

- (2) : boucle
  - Gestion de compteur i : n affectations
  - Corps de boucle :

$$\sum_{2}^{n} \text{affectations dans (3)}$$

    - Pire cas : n-1 affectations
    - Meilleur cas : 0 affectations
    - Cas moyen : (n - 1)/2 affectations

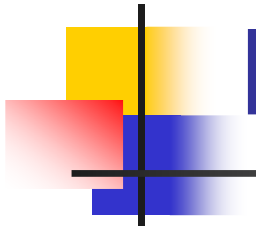


Le plus grand élément

- La complexité de l'algorithme
  - C(n) = C1 + C2(n) + C5

  - le pire de cas : 2n affectations
  - le meilleur de cas : (n + 1) affectations
  - le cas moyen : (3n + 1)/2 affectations

- Complexité linéaire !



Recherche séquentielle

27	63	1	72	64	58	14	9

Recherche séquentielle

```
Fonction RechSeq (A[n] : Tableau de T, val : T) : Booléen
Variables i : Entier
Début
    i ← 0
    Tantque (i < n)
        Si (A[i] = val) Alors
                Retouner vrai
        Sinon
                i ← i+1
        FinSi
    FinTantque
    Retouner faux
Fin
```

Recherche séquentielle : complexité

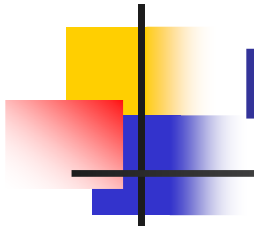
- Meilleur cas :
  - nombre de tests d'égalité : 1
- Pire cas :
  - nombre de tests d'égalité : n
- Moyenne :
  - p : probabilité que val soit dans *A*
  - places équiprobables : p/n
  - nb de tests d'égalité :

$$\sum_{i=1}^{n} \frac{p}{n} i + (1-p)n = p(n+1)/2 + (1-p)n$$

- Complexité linéaire !

Recherche dichotomique

```
Fonction RechDic (A[n] : Tableau de T, début, fin : Entier, val : T) : Booléen
Variables mid : Entier
Début
    Si (debut > fin) Alors
        Retourner faux
    Sinon
        mid ← (debut + fin) /2
        Si (A[mid] = val) Alors
            Retourner vrai
        Sinon
            Si (val > A[mid]) Alors
                Retourner RechDic(A[n], mid+1, fin, val)
            Sinon
                Retourner RechDic(A[n], debut, mid-1, val)
            FinSi
```

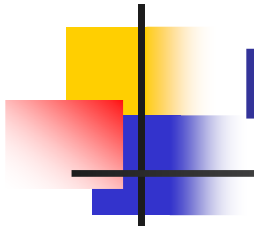


Recherche dichotomique : complexité

- Tableau A doit être trié !
- RechDic(A[n],0,n-1,val)
- C(n) : nombre de tests d'égalité (pire des cas)

  C(1) = 1

  C(n) = 1 + C(n/2)



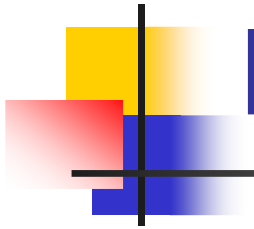
Recherche dichotomique : complexité

- Tableau A doit être trié !
- RechDic(A[n],0,n-1,val)
- C(n) : nombre de tests d'égalité (pire des cas)

  C(1) = 1

  C(n) = 1 + C(n/2)

  => C(n) = O(log n)



La plus grande somme

- Calculer la plus grande somme contiguë d'une séquence d'entiers placés dans un tableau

- Exemple
  - A = {31, -41, 59, 26, -53, 58, 97, -93, -23, 84}
  - A[2]+A[3]+…+A[6] = 187

Algorithme 1

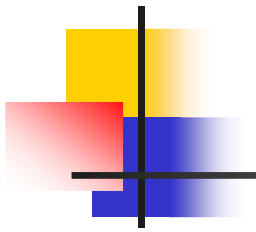
```
Fonction Algo1(A[n] : Tableau d'Entier) : Entier
Variables i, j, k, max, somme : Entier
Début
    max ← 0
    Pour i de 0 à n-1
        Pour j de i à n-1
            somme ← 0
            Pour k de i à j
                somme ← somme + A[k]
                Si (somme > max) alors
                    max ← somme
                FinSi
            FinPour
        FinPour
    FinPour
    retouner max
Fin
```

Algorithme 2

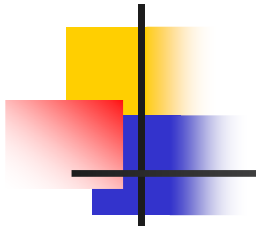
```
Fonction Algo2(A[n] : Tableau d'Entier) : Entier
Variables i, j, max, somme : Entier
Début
    max ← 0
    Pour i de 0 à n-1
        somme ← 0
        Pour j de i à n-1
            somme ← somme + A[j]
            Si (somme > max) alors
                max ← somme
            FinSi
        FinPour
    FinPour
    retouner max
Fin
```

Algorithme 3

```
Fonction Algo3(A[n] : Tableau d'Entier, g, d : Entier) : Entier
Variables m, i, max1, max2, max, somme : Entier
Début
    Si (g > d) alors
        retourner 0
    Sinon
        Si (g == d) alors
            Si (A[g] > 0) alors
                retourner A[g]
            Sinon
                retourner 0
            Finsi
        Sinon
```



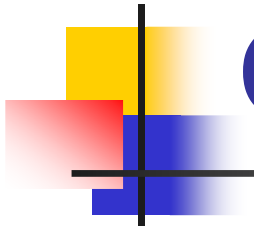
```
m ← (g + d)/2
max1 ← 0
somme ← 0
Pour i de m à g pas -1
        somme ← somme + A[i]
        Si (somme > max1) alors
                max1 ← somme
        FinSi
FinPour
max2 ← 0
somme ← 0
Pour i de m+1 à d
        somme ← somme + A[i]
        Si (somme > max2) alors
                max2 ← somme
        FinSi
FinPour
max ← max1 + max2
```



```
                    somme ← Algo3(A[n], g, m-1)
                    Si (somme > max) alors
                            max ← somme
                    FinSi
                    somme ← Algo3(A[n], m+1, d)
                    Si (somme > max) alors
                            max ← somme
                    FinSi
                    retourner max
            FinSi
        FinSi
Fin
```

Algorithme 4

```
Fonction Algo4(A[n] : Tableau d'Entier) : Entier
Variables i, max, maxalafin : Entier
Début
    max ← 0
    maxalafin ← 0
    Pour i de 0 à n-1
            maxalafin ← maxalafin + A[i]
            Si (maxalafin < 0) alors
                    maxalafin ← 0
            Sinon
                    Si (maxalafin > max) alors
                            max ← maxalafin
                    FinSi
            FinSi
    FinPour
    retouner max
Fin
```



Comparaisons

- PC à 400MHz
- Tableau de taille 10^7

Algorithme	1	2	3	4
Temps	41000 ans	1,7 semaines	11 secondes	0,48 secondes