Les tableaux 1D

Partie 1

Les tableaux 1D et premiers sous-programmes associés

Beaucoup de déjà vu et un peu de nouveau

Les tableaux

- Première structure de données pour stocker des valeurs de même type

Les tableaux 1D : les vecteurs

- les définir, stocker : écrire et lire une valeur
- parcourir les éléments d'un tableau

Tableau : oui-non ? dimension ?

Les notes d'algo du S1 des étudiants du L1 math-info en 2014

Le relevé des vitesses de la tramontane mesurées à Perpignan, à midi, tous les jours de janvier 2015 ?

Le relevé des vitesses et de l'orientation du vent mesurées à Perpignan, à midi, tous les jours de janvier 2015 ?

La musique de votre fichier mp3 préféré ?

La photo de fond d'écran de votre smartphone ?

La version noir et blanc de la photo de fond d'écran de votre smartphone ?

L'ensemble des date de naissance des étudiants présents ?

Cette ligne a priori sans intérêt !!! ?

Tableaux 1D : vecteurs

Première structure de données pour stocker des valeurs de même type

Ce qui a été présenté au S1

La définition

mot =

B	o	n	j	o	u	r		!	
0	1	2	3	4	5	6	7	8	9

Les notions importantes

- déclaration = identifiant, dimension, tailles, type des valeurs
- dimension : 1D, 3D, 3, ...
- tailles : nb **maximum** de valeurs stockables pour chaque dimension
- dimension et tailles sont connues et **fixées** avant le premier stockage
- indice des valeurs, indice du premier élément : ici 0
- accès direct à la valeur de l'élément d'indice donné : `mot[5]`

Sur les tableaux, on sait

```
mot: tableau[10] de caractères = "Bonjour ! "
temperatures: tableau[12] d'entiers = [9, 5, 8, 11, 13, 19, 24, 25, 22, 20, 18, 12]
```

Un tableau est une variable à part entière

- un tableau de 10 caractères n'est pas un caractère
- la 4ème valeur du tableau `mot` est un caractère
- son indice vaut 3, sa valeur vaut `j` , on le note `mot[3] = j`
- `mot[11]` n'existe pas et essayer de le trouver provoque une énorme erreur
- Que dire du tableau `temperatures` ?

Le tableau à trois amis

- son préféré : la boucle `pour`
- les autres : la boucle `tant que` ou `jusqu'à`

Parcourir des tableaux 1D

Les tableaux c'est fait pour ça !

Lire, stocker 12 températures, calculer et afficher leur moyenne

```
declare
        moy : entier // la moyenne sera arrondie en entier

debut


        ... on complète ensemble ...

fin
```

Je dois parcourir un tableau. La question à se poser :
Est-ce que je connais a priori le nombre de fois ?
- oui : je choisis une boucle `pour`
- non : je choisis une boucle `tant que` ou `jusqu'à`

1. Lire, stocker 12 températures, calculer et afficher les températures et leur moyenne "arrondie en entier"

```
declare
    moy : entier                          // la moyenne sera en entier
    nbVal : constante entier = 12         // 12 valeurs une fois pour toute !
    temperature: tableau[nbVal] d'entiers // le vecteur de valeurs
    i : entier                            // parcourt le vecteur
    val : entier                          // aide à la lisibilité
debut
    moy = 0                               // aurait pu être initialisé avant ! ou ?
    pour i de 0 a nbVal-1 faire           // pour chaque valeur du vecteur
        lire(val)                         // on lit
        temperature[i] = val              // on stocke
        moy = moy + val                   // un calcul partiel: accumulation
    fin pour
    moy = moy ÷ nbVal                     // division entière (÷) et fin du calcul

    pour i de 0 à nbVal-1 faire           // on affiche ...
        afficher("température du mois", i, " : ", temperature[i], sdl) //
sdl:saut_de_ligne
    fin pour
    afficher("température moyenne ", moy, sdl)
fin
```


En plus, trouver la température minimale de ce relevé

```
declare
    ...
debut


fin
```

Je dois parcourir un tableau. La question à se poser :
*Est-ce que je connais **a priori** le nombre de fois ?*
- oui : je choisis une boucle pour
- non : je choisis une boucle tant que ou jusqu'à

2. En plus, trouver la température minimale de ce relevé

```
declare
    min : entier                          // le min cherché
    nbVal : constante entier = 12         //
    temperature: tableau[nbVal] d'entiers // le vecteur de valeurs
    i : entier                            //

debut
    pour i de 0 à nbVal-1 faire           // phase 1 :on lit et maj le tableau
        lire(temperature[i] )             //
    fin pour
    min = temperature[0]                  // existe si le tableau est non vide
    pour i de 1 à nbVal-1 faire           // phase 2 : traitement
        si temperature[i] < min           // on compare
            min = temperature[i]          // la nouvelle valeur de min
    fin pour
    afficher("La température minimale :", min, sdl) // phase 3 : affichage
fin
```


En plus, trouver la température minimale de ce relevé

Que retenir :

- toujours se poser la question du choix de la structure de répétition !
- la boucle **pour** est adaptée car on doit parcourir tout le tableau une fois
- on connait toutes les valeurs prises par l'indice i = 0, 1, ..., n-1
- il est utile de déclarer la longueur du tableau comme une constante supplémentaire, ici **nbVal**, qui facilite les modifications ultérieures de cette valeur, ou l'écriture d'une fonction.
- on peut se dispenser de la variable intermédiaire de stockage **val** et affecter directement la valeur lue à sa bonne place dans le tableau
- un peu pénible de mélanger lecture-écriture et le calcul
  3 parties = 3 fonctions : lecture des valeurs / traitement / écriture résultats

En plus, trouver le numéro du mois le plus froid

LE numéro -> je suppose qu'il est unique
par exemple, toutes les températures mensuelles sont différentes

1. sans connaître la température minimale

declare
...
debut

Je dois parcourir un tableau. La question à se poser :
*Est-ce que je connais **a priori** le nombre de fois ?*
- oui : je choisis une boucle pour
- non : je choisis une boucle tant que ou jusqu'à

fin

Trouver le numéro du mois le plus froid

1. sans connaître la température minimale

```
declare
    min : entier                                    // la valeur min
    indice_min : entier = 0                 // l'indice du min cherché
    nbVal : constante entier = 12
    temperature : tableau[nbVal] d'entiers      // le vecteur de valeurs
    i : entier
    val : entier

debut
    pour i de 0 à nbVal-1 faire             // phase 1 :on lit et maj le tableau
        lire(temperature[i] )
    fin pour
    min = temperature[0]
    pour i de 1 à nbVal-1 faire             // pour chaque valeur du vecteur
        si temperature[i] < min             // on compare
            min = temperature[i]            // min = min(t) pour t = temperature[0..i]
            indice_min  = i                 // on maj la nouvelle valeur de l'indice
    fin pour

    afficher("mois le plus froid ", indice_min, sdl)
fin
```


Trouver le numéro du mois le plus froid

4. en connaissant la température minimale

On suppose que le tableau des valeurs est connu.

declare
...
debut

Je dois parcourir un tableau. La question à se poser :
*Est-ce que je connais **a priori** le nombre de fois ?*
- oui : je choisis une boucle pour
- non : je choisis une boucle tant que ou jusqu'à

fin

Trouver le numéro du mois le plus froid

2. en connaissant la température minimale

```
declare
    min : entier = 5                        // la valeur min supposée connue
    indice_min : entier = 0                 // l'indice du min cherché
    nbVal : constante entier = 12
    temperature : tableau[nbVal] d'entiers          // le vecteur
        = [9, 5, 8, 11, 13, 19, 24, 25, 22, 20, 18, 12]   // et ses valeurs ...
    i : entier = 0                                  // l'indice de parcours
debut
    tantque temperature[i] > min repéter    // inégalité aussi possible
        i = i + 1                           // on gère l'indice de parcours
    fin tantque                             // termine car le min est connu

    indice_min  = i                         // i est l'indice du min cherché

    afficher("mois le plus froid ", indice_min, sdl)
fin
```

Trouver le numéro du mois le plus froid

2. en connaissant la température minimale

Que retenir :

- choix d'une boucle `tantque` plutôt que `pour`
  - . on fait **au plus** `nbVal` parcours du tableau
  - . on aurait pu utiliser une boucle `pour` —> *exercice*
  - . on aurait pu utiliser une boucle `jusqu'à` —> *exercice*
- dans ces cas (`tantque`, `jusqu'à`), il faut gérer l'indice de parcours

Approfondissons :

Si la valeur minimale est présente plus d'une fois,
quelle est l'occurrence de cette valeur trouvée
dans les deux versions de cette recherche ?

Les sous-programmes qui manipulent des tableaux

Un peu plus dur, surtout en pratique selon les langages

Les sous-prog. qui manipulent des tableaux

Manipuler
un ou des tableaux est un paramètre IN ou INOUT du sous-programme

La difficulté : si on ne lui indique pas ...
le sous-prog. ne connait pas la **taille** de l'argument effectif-tableau

Que doit connaître le sous-prog. dans l'en-tête ?
la dimension et le type des éléments de chaque argument
la taille doit être un paramètre formel supplémentaire

Que doit connaître le sous-prog. dans l'appel ?
l'identifiant de chaque argument effectif-tableau
la taille de chaque argument effectif-tableau

Donc dans le corps
pas la peine de déclarer un tableau local (sauf besoin particulier)
utiliser identifiant et la taille formels

Deux exemples ... et des questions !

1. On veut la valeur max d'un tableau de taille n donnée en entrée
2. On veut trier par ordre croissant les valeurs d'un tableau de taille n donnée en entrée

Questions qui aident à la conception de l'algo :

- quels paramètres en entrée ? quels modes ?
- quels paramètres en sortie ? quel mode ?
- besoin d'un tableau local au sous-programme ?
- nombre de parcours du tableau connu a priori ?

```
fonction max(t : tableau[n] d'entiers, n: entier) retourne entier
```

L'appel —>

```
notes [10] = [12,9,14,9,17,4,10,12,13,8]
bonne_note = max(notes, 10)
```

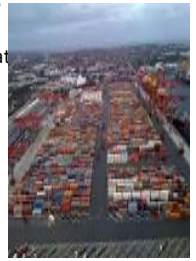
```
fonction max(t : tableau[n] d'entiers, n: entier) retourne entier
    res : entier = t[0]          //résultat, on l'initialise dès la déclaration
    i : entier                   //itérateur
debut
    pour i de 1 à n faire
        si (t[i] > res) alors
            res = t[i]
        fin si
    fin pour
    retourner res
fin fonction
```

<— Le corps

Trier : le début d'une longue histoire

Trier = classer, ordonner des valeurs, des objets

- trier des copies par notes
- trier des copies par ordre alphabétique des candidat
- trier des livres dans une bibliothèque
- trier des containers sur un port



Un premier algorithme : le tri insertion

Trier des cartes

Exemple en pratique : trier une main de cartes.

Vocabulaire :

- une main de cartes est l'ensemble des cartes que je tiens dans ma main. Le nombre de carte d'une main est variable selon les jeux, les moments du jeu, ...
- trier => ordre. il existe plusieurs ordres pour trier une main de cartes : couleurs, valeurs, peut dépendre du jeux, ...

Lorsqu'on parle de tri sur des cartes et que rien n'est mentionné, on suppose un jeu de cartes d'une seule couleur et de n valeurs différentes et uniques.

Trier ... par ordre croissant (par exemple) :

- au début, ma main contient aucune carte
- je reçois une carte, l'une après l'autre et je veux la ranger en ordre dans ma "main"
- une fois la main complète (n cartes), les cartes sont rangées par ordre croissant : chaque carte est plus grande que sa voisine de gauche (si elle existe), et plus petite que sa voisin de droite (si elle existe)

Tri par insertion

Principe :

- je parcours la main de carte triées et j'insère à sa bonne place la carte à classer

Remarque importante : il est difficile d'être exhaustif et précis en français

- *je parcours* :
  il faut décider d'un début et d'un sens de parcours
  . de la carte la plus à gauche jusqu'à la dernière la plus à droite → gauche -> droite
  . ce sens est celui des cartes à classer
  . le contraire est possible
- *j'insère* :
  en pratique pour insérer la nouvelle carte, je fais de l'espace à l'endroit où je vais ranger la carte
  . je créé un espace avec la carte à ranger,
  . puis je déplace les cartes une à une vers la droite → droite -> gauche ... pourquoi ce sens ?
  . jusqu'à que l'espace (qui se déplace vers la gauche) permette l'insertion à la bonne place
  . autre solution possible
- *la bonne place :*
  la *position* du placement est-elle définitive ?
  . par rapport au début ? par rapport à sa voisine de gauche ? de droite ?
  . pourtant la main est triée ... mais incomplète

Tri par insertion

De l'analyse du cas abstrait à l'algorithme :

- tas de cartes, main de cartes : définir une structure de données adaptée
- cartes, valeurs : choisir un type adapté
  un tableau d'entiers ... sa taille ? son initialisation ?
  - taille : le nombre total de cartes
  - initialisation : toutes les cartes sont présentes dans le tableau avant le début du tri

  Déf: le tri par insertion est un tri en place :
  on utilise qu'un seul tableau, celui des valeurs à trier
- ordre ... naturel sur les entiers : ≤
- parcours :
  - parcours principal : itérer sur tous les indices du tableau
  - parcours interne sur le sous-tableau déjà trié :
    . itérer tantque/jusqu'à la bonne place


```
fonction tri_insertion(t : in out tableau[n] d'entiers, n: entier) // procedure
                                                notes [10] = [12,9,14,9,17,4,10,12,13,8]
                                                tri_insertion(notes, 10)
fonction tri_insertion(t : in out tableau[n] d'entiers, n: entier)
    val_a_placer : entier          //valeur à placer correctement
    i : entier                     //indice de parcours des éléments à placer
    j : entier                     //indice de parcours des éléments à gauche de t[i]
debut
    pour i de 1 à n-1 faire        // ne s'exécute pas si n==1
        val_a_placer = t[i]
        j = i - 1
        tant que (j >= 0) et (t[j] > val_a_placer) faire
            t[j + 1] = t[j] //déplace t[j] d'une position à droite
            j = j - 1              //indice du prochain élément à gauche
        fin tant que
        t[j + 1] = val_a_placer
fin fonction
```


Exemples en interaction

On veut déterminer si un tableau contient une certaine valeur

1. quels paramètres IN ? identifiant ? type ?
2. quels paramètres en OUT ? type ?
3. quelles pré-conditions pour que le problème ait un sens ?
4. quel principe pour l'algorithme : comment résoudre le pb ?
5. parcourir le tableau
   1. avec boucle pour ?
   2. avec boucle tantque ou jusqu'à?
6. quelles variables locales ?
7. quelles conditions exceptionnelles "plantent" l'algo ?

Déterminer si un tableau contient une certaine valeur

Appel : le traitement souhaité

```
//role : je teste ma fonction appartient
declare
    mot[10] : tableau de caracteres = "Bonjour ! "
    //chaine à explorer
    c : caracteres
    //caractère à chercher dans mot
    reponse : booleen
    //il y est ou non ?

debut
    c1 = 'j'
    reponse = appartient(mot, 10, c1)
    afficher(reponse)
    // j'attends Vrai
    c = 'o'

    afficher(reponse)
    // j'attends Vrai
    reponse = appartient(mot, 10, c1)
    c = 'a'
    reponse = appartient(mot, 10, c1)
    afficher(reponse)
    //j'attends Faux
```

Déterminer si un tableau contient une certaine valeur
1. avec un **tantque**

```
fonction appartient(c : caractere, s : tableau[n] de caracteres, n : entier)
                                    retourne booleen
        b : booleen = False
debut

        ...


        retourne b
fin
```


Déterminer si un tableau contient une certaine valeur
2. avec un **pour**

```
fonction appartient(c : caractere, s : tableau[n] de caracteres, n : entier)
                                    retourne booleen
        b : booleen = False
debut


        ...

        retourne b
fin
```


Première occurrence :
déterminer le premier indice d'une valeur donnée dans un tableau

```
fonction premiere_occ(c : caractere, s : tableau[n] de caracteres, n: entier)
  retourne entier
        pos : entier = 0
debut

        ...


        retourne pos
fin
```

Question :
- y-a-t-il des conditions exceptionnelles à prendre en compte ?

Dernière occurrence :
déterminer le dernier indice d'une valeur donnée dans un tableau

```
fonction derniere_occ(c : caractere, s : tableau[n] de caracteres, n: entier)
  retourne entier
        pos : entier = 0
debut

        ...


        retourne pos
fin
```

Question :
- y-a-t-il des conditions exceptionnelles à prendre en compte ?

Exercices

Dans l'examen de décembre 2015

- ex 4 : *Écrire l'algorithme qui retourne l'indice de la dernière occurrence d'un tableau T*
- ex 5 : *Écrire l'algorithme (puis la fonction) qui retourne la plus grande différence (absolue) entre deux valeurs consécutives d'un tableau T*

Un petit à préparer

- compter le nombre de valeurs plus petites qu'une valeur donnée

Synthèse de la séance

Ce qu'il faut retenir !

Tableaux 1D

- Structure de stockage linéaire indicée par des entiers de 0 à taille_max - 1
- Déclaration : identifiant, dimension, taille max, le type des valeurs
  notes[32] : tableau d'entiers
- Dimensions et taille sont connues avant le premier stockage
- Parcours élément par élément avec
  1. boucle pour sur les indices
  2. boucles tantque ou jusqua avec gestion de l'indice et de la condition d'arrêt
- Les sous-programmes qui manipulent des tableaux ne connaissent pas la taille des arguments effectifs : il faut passer la taille comme argument
  fonction max(t : tableau[n] d'entiers, n: entier) retourne entier
- Premiers traitements à maîtriser :
  - parcourir le tableau
  - rechercher une valeur ou un indice

A suivre

Des tableaux 2D
- tableaux 2D : images, matrices
- les boucles pour imbriquées
- algorithmes (de traitement) d'images
- algorithmes avec/sur des matrices

Généralisation aux tableaux 3D, ... , multidimensionnels