Fortier
Amandine
17-18

CHAPITRE 1 : TABLEAU 2D

Intro

Déclaration :

nomTab :tableau de nbLignes X nbColonnes typeTab
tabTest :tableau de 3X4 entier

	0	1	2	3
0				
1				
2				

Récupérer une cellule :

J'ai envie de récupérer l'élément de la 2ème lignes et de la 3ème colonne.
variable ← tabTest[1,2] **TOUJOURS LIGNES PUIS COLONNE**

Initialiser :

On utilise 2 « pour », un qui parcoure les lignes et dans celui-ci, un autre « pour » qui parcoure les colonnes.

Algorithme

Parcours du tableau

Sans arrêt

```
// Parcours d'un tableau à 2 dimensions, ligne par ligne
algorithme affichageElémentsLigneParLigne(tab : tableau de n × m T)
    pour lg de 0 à n-1 faire
        pour col de 0 à m-1 faire
            afficher tab[lg,col]                    // On peut faire autre chose qu'afficher
        fin pour
    fin pour
fin algorithme
```

Avec arrêt

```
// Parcours d'un tableau à 2 dimensions, ligne par ligne, via un tant que
algorithme affichageElémentsLigneParLigne(tab : tableau de n × m T)
    lg, col : entiers
    lg ← 0
    tant que lg < n faire
        col ← 0
        tant que col < m faire
            afficher tab[lg, col]                   // On peut faire autre chose qu'afficher
            col ← col + 1
        fin tant que
        lg ← lg + 1
    fin tant que
fin algorithme
```

DEV2 - ALG

Diagonale montante

```
// Parcours de la diagonale montante d'un tableau carré - version 1 indice
algorithme affichageElémentsDiagonaleMontante(tab : tableau de n × n T)
    pour i de 0 à n-1 faire
        afficher tab[i, n - 1 - i]                    // On peut faire autre chose qu'afficher
    fin pour
fin algorithme
```

		1
	2	
3		

Diagonale descendante

```
// Parcours de la diagonale descendante d'un tableau carré
algorithme affichageElémentsDiagonaleDescendante(tab : tableau de n × n T)
    pour i de 0 à n-1 faire
        afficher tab[i,i]                             // On peut faire autre chose qu'afficher
    fin pour
fin algorithme
```

1		
	2	
		3

Serpent

```
// Parcours du serpent dans un tableau à deux dimensions
algorithme affichageElémentsSerpent(tab : tableau de n × m T)
    lg, col, depl : entiers
    lg ← 0
    col ← 0
    depl ← 1                                          // 1 pour avancer, -1 pour reculer
    pour i de 1 à n*m faire
        afficher tab[lg, col]                         // On peut faire autre chose qu'afficher
        si 0 ≤ col + depl ET col + depl < m alors
            col ← col + depl                          // On se déplace dans la ligne
        sinon
            lg ← lg + 1                               // On passe à la ligne suivante
            depl ← -depl                              // et on change de sens
        fin si
    fin pour
fin algorithme
```

1	2	3	4	5
10	9	8	7	6
11	12	13	14	15

CHAPITRE 2 : OO

Classe : moule
Objet : le gâteau qui est dans le moule
Constructeur permet de créer l'objet (initialiser)
Signature : nom méthode et description (paramètre)
État : valeur de ces attributs à un instant t.
J'appelle la méthode sur l'objet durée2 = écriture pointée : durée2.difference(duree1)

Syntaxe

```
classe Durée
 privé:
    totalSecondes : entier
 public:
    // Ici viennent les constructeurs et les méthodes
fin classe
```

DEV2 - ALG

Constructeur

Avec test

```
constructeur Durée(secondes : entier)
    si secondes < 0 alors
        erreur "paramètre négatif" a
    fin si
    totalSecondes ← secondes
fin constructeur
```

L'instruction **erreur** indique que l'algorithme ne peut pas poursuivre normalement. Il s'arrête avec un message d'erreur.

Sans test

constructeur Duree(uneSeconde :entier)
seconde ←uneSeconde
fin constructeur

Méthode

```
méthode getSeconde() → entier
    // On doit enlever les minutes éventuelles
    retourner totalSecondes MOD 60
fin méthode
```

Utilisation

```
algorithme diffDurée()
    durée1, durée2 : Durée                    // Les variables sont déclarées/créées
    durée1 ← nouvelle Durée(3, 4, 49)         // Les objets sont créés
    durée2 ← nouvelle Durée(3, 24, 37)        // Les objets sont créés
    afficher durée2.différence(durée1)
fin algorithme
```

Demander Classe

rendezVous : Durée
demander rendezVous

MCV Modèle-Vue-Contrôleur

Pour la vue graphique, on met 1 classe en plus, comme ça on peut même proposer au début, qu'elle vue le joueur à envie.

Controleur = le « main »
Modèle = les classes, lièvres, tortue et dé
Vue = vuelièvretortue ➔ afficherEtat, afficherVainqueur

CHAPITRE 3 : LISTE

Une liste ne contient des éléments que d'un seul type. Liste d'entier, …
Le 1er élément de la liste = 0.

Méthodes données que l'on peut utiliser

```
classe Liste de T                                          // T est un type quelconque
 public:
    constructeur Liste de T()                                  // construit une liste vide
    méthode get(pos : entier) → T                          // donne un élément en position pos
    méthode set(pos : entier, valeur : T)                  // modifie un élément en position pos
    méthode taille() → entier                              // donne le nombre actuel d'éléments
    méthode ajouter(valeur : T)                            // ajoute un élément en fin de liste
    méthode insérer(pos : entier, valeur : T)              // insère un élément en position pos
    méthode supprimer()                                        // supprime le dernier élément
    méthode supprimerPos(pos : entier)                     // supprime l'élément en position pos
    méthode supprimer(valeur : T) → booléen                // supprime l'élément de valeur donnée
    méthode vider()                                                    // vide la liste
    méthode estVide() → booléen                                // la liste est-elle vide ?
    méthode existe(valeur ↓ : T, pos ↑ : entier) → booléen     // recherche un élément
fin classe
```

« supprimer », s'il y a plusieurs valeurs il ne supprime que la 1ère. Pareil pour existe etc
Inserer : insère et déplace, quand on supprime, tout est aussi déplacer.
Set : modifie donc ne déplace pas.

Afficher liste

```
algorithme afficher(liste : Liste d'entiers)
    pour i de 0 à liste.taille()-1 faire
        afficher liste.get(i)
    fin pour
fin algorithme
```

CHAPITRE 4 : TRAITEMENTS DE RUPTURE

Il faut avoir un tableau trié pour pouvoir faire des ruptures. **!! utiliser tant que !!**

Exemple de rupture : savoir combien d'élèves par sections. LA rupture c'est, dès qu'il voit que l'option change de valeur, il affiche le résultat puis continu jusqu'à ce qu'il y ait une autre rupture/changement de valeur.

Clé de tri : majeur, médian, mineur
On classe d'abord par majeur puis par médian et enfin par mineur.
Exemple : une liste avec des personnes, on peut trier en majeur sur les noms et en mineur sur les prénoms. Si on a 3 tri alors on utilise médian.

Classement complexe : classement qui peut se faire sur plusieurs clés de tri.

DEV2 - ALG

Niveau rupture

- Niveau 0 : fin des données, exemple en fin de parcoure de tableau.

```
algorithme RuptureNiveau0(etudiants : liste d'Etudiant)
    etd : Etudiant
    i : entier
    i ← 0
    tant que i < etudiants.taille() faire
        // traitement de etudiants.get(i)
        i ← i + 1
    fin tant que
fin algorithme
```

- Niveau 1 : clé **tri majeur**, avec 2 boucle tant que.

```
algorithme RuptureNiveau1(etudiants : liste d'Etudiant)
    // on suppose les données classées en majeur sur l'option
    etd : Etudiant
    saveOption : chaine
    cpt : entier
    i : entier
    i ← 0
    tant que i < etudiants.taille() faire
        saveOption ← etudiants.get(i).option
        cpt ← 0
        tant que i < etudiants.taille() ET etudiants.get(i).option = saveOption faire
            cpt ← cpt + 1
            i ← i + 1
        fin tant que
        afficher cpt, « étudiant dans l'option », saveOption
    fin tant que
fin algorithme
```

- Niveau 2 : **tri majeur et mineur**, 3 boucle tant que.

```
algorithme RuptureNiveau2(etudiants : liste d'Etudiant)
    // on suppose les données classées en majeur sur l'option
    // et en mineur sur la date de naissance (ordre chronologique)
    etd : Etudiant
    saveOption : chaine
    saveAnnéeNaissance : entier
    cpt : entier
    i : entier
    i ← 0
    tant que i < etudiants.taille() faire
        saveOption ← etd.option
        tant que i < etudiants.taille() ET etudiants.get(i).option = saveOption faire
            saveAnnéeNaissance ← etudiants.get(i).dateNaissance.année
            cpt ← 0
            tant que i < etudiants.taille()
                        ET etudiants.get(i).option = saveOption
                        ET etudiants.get(i).dateNaissance.année = saveAnnéeNaissance faire
                cpt ← cpt + 1
                i ← i + 1
            fin tant que
            afficher cpt, « étudiant dans l'option », saveOption, « sont nés en », saveAnneeNaissance
        fin tant que
    fin tant que
fin algorithme
```

CHAPITRE 5 : REPRESENTATION DES DONNEES

- Les données « simples » (variables isolées : entiers, réels, chaines, caractères, booléens)
- Les variables structurées, qui regroupent en une seule entité une collection de variables simples.
- Le tableau, qui contient un nombre déterminé de variables de même type, accessibles via un indice ou plusieurs pour les tableaux multidimensionnels ;
- Les objets, qui combinent en un tout une série d'attributs et des méthodes agissant sur ces attributs ;
- La Liste, qui peut contenir un nombre indéfini d'éléments de même type.

Lorsque tableau 2 dés compliqué et/ou qu'on n'utilise pas toutes les cases, on peut parfois utiliser listes.