Introduction à la Programmation 1

Séance 4 de cours/TD

Université Paris-Diderot

Objectifs:

- Contrôle continu (45 min, 3 exercices tirés des travaux dirigés)
- Déclarer et initialiser un tableau de type int.
- Écrire des fonctions qui attendent des tableaux en entrée et produisent des tableaux en sortie.

1 Les tableaux d'entiers

Tableaux d'entiers [Cours]

- Un tableau est une suite de cases contenant chacune une valeur.
- Un tableau d'entiers contenant 2 dans sa première case, 4 dans sa deuxième case et 12 dans sa troisième case sera noté $\{2, 4, 12\}$.
- Les tableaux d'entiers ont le type « int [] ».
- Si T est un type alors « T[] » est le type des tableaux dans les valeurs sont de type T.
- Comme toute valeur, un tableau peut être stocké dans une variable. De même, une fonction peut prendre un tableau en argument et renvoyer un tableau. Ainsi, si une fonction renvoie un tableau d'entiers, son type de retour est int [] .
- Le nombre de cases d'un tableau est appelé sa longueur.
- Si l est la longueur d'un tableau alors les cases du tableau sont numérotées de 0 à $l - 1$. Ces numéros sont appelés *indices*.
- Pour lire la valeur de la première case d'un tableau t, on écrit « t[0] ». Plus généralement, pour lire la case d'indice i, on utilise l'expression « t[i] ».
- Pour modifier la valeur de la case d'indice i d'un tableau t, on utilise l'instruction « t[i] = e; » où e est une expression qui s'évalue en une valeur du type des cases de t.
- **Attention** à ne jamais lire ou écrire hors des limites du tableau car cela produit une erreur. Ainsi, les indices négatifs ou plus grand que la longueur du tableau sont à éviter.
- Dans ces travaux dirigés, nous utiliserons les fonctions
  - « int intArrayLength (int [] t) » pour obtenir la taille d'un tableau d'entiers t donné en argument et la procédure
  - « void printIntArray (int [] t) » qui affichera sur une ligne les valeurs des cases d'un tableau d'entiers en les séparant par des espaces.

Exercice 1 (Comprendre la taille et les indices, ⋆)

1. *Si un tableau* a *vaut* $\{1, 3, 5, 7, 9, 11, 13, 15, 17\}$. *Quelle est sa taille ? À quel indice trouve-t-on la valeur* 1 *? À quel indice trouve-t-on la valeur* 17 *? À quel indice trouve-t-on la valeur* 9 *?*
2. *Si le tableau* b *vaut* $\{2, 2, 3, 3, 4, 5, 7\}$. *Quelle est sa taille ? Que vaut l'expression* b[0] *? Que vaut l'expression* b[4] *? Pourquoi ne peut-on pas évaluer* b[7] *?*

□

Exercice 2 (Lecture, ⋆)
La déclaration suivante introduit un tableau t *dont les cases sont de type* int *et qui vaut* $\{2, 3, 4, 5, 6, 7\}$.

```
int [] t = { 2, 3, 4, 5, 6, 7 };
```

1. *Donner une instruction modifiant la deuxième case du tableau* t *pour y mettre la valeur* 10.
2. *En utilisant la procédure* void printInt (int x) *pour afficher la valeur d'un entier, donner les instructions permettant d'afficher la première case du tableau* t *et la dernière case du tableau* t.
3. *Quel est le contenu du tableau* t *après exécution des instructions suivantes :*

```
int[] t = { 2, 3, 4, 5, 6, 7 };
for (int i = 0; i < intArrayLength (t); i++) {
  t[i] = i;
}
```

□

Exercice 3 (Swap, ⋆)
On considère la déclaration suivante :

```
int[] a = { 12, -3, 22, 55, 9 };
```

1. *Donner une suite d'instructions permettant d'inverser les contenus de la première et la deuxième case du tableau* a.
2. *Donner une suite d'instructions permettant d'inverser les contenus de la première et de la dernière case du tableau* a.

□

2 Déclarer un tableau

Déclaration d'un tableau [Cours]

— Pour déclarer qu'une variable t est un tableau, on la déclare avec son type de la façon suivante « int [] t = e; » où e est l'une des formes suivantes :
  — Une valeur d'initialisation pour le tableau.
    Par exemple, la déclaration « int [] t = { 2, 6, 7 }; » introduit une variable de type tableau d'entiers valant $\{2, 6, 7\}$.
  — Une expression de création d'un tableau non initialisé, c'est-à-dire dont le contenu des cases n'est pas précisé. Dans ce cas, cette expression ne spécifie que la longueur du tableau et le type de ses cases.
    Par exemple, la déclaration « int [] t = new int[5]; » crée un tableau t de taille 5, mais les valeurs que l'on trouve dans chaque case ne sont pas spécifiées. (En théorie, cela peut être n'importe quelle valeur entière.)
— Après avoir créé un tableau avec l'opérateur new, il faut en initialiser les cases.
  Par exemple, pour déclarer et initialiser correctement un tableau a pour qu'il vaille $\{-2, -1, 0, 1\}$, on peut procéder ainsi :

```
int[] a = new int[4];
a[0] = -2;
a[1] = -1;
a[2] = 0;
a[3] = 1;
```

— **Attention :** les valeurs d'initialisation (comme par exemple { 2, 6, 7 }) ne sont pas des expressions. On ne peut les utiliser que dans les déclarations de variables et non dans les expressions. Par exemple, on ne peut pas écrire l'affectation « t = { 2, 6, 7 }; ».

Exercice 4 (Création de tableau, ⋆)

1. *Donner les instructions pour créer un tableau* a *valant* $1, 2, 3, 4, 5, \ldots, 1000$.
   (Il est bien sûr recommandé d'utiliser une boucle for *pour remplir un tel tableau.)*

□

Exercice 5 (Manipulation de tableaux, ⋆)
Donner les instructions réalisant les opérations suivantes :

1. *Créer un tableau* t *valant* $\{2, 4, 6, 8, 10, 12, 14, 16\}$.
2. *Créer un tableau* s *de la même taille que* t *qui contient à chaque indice le double de la valeur contenue dans* t *à l'indice correspondant.*

□

Exercice 6 (Indices, ⋆)
Créer un tableau a *de longueur* 10 *et dont chaque élément se trouvant à l'indice* i *a pour valeur* $2 * i$. □

Exercice 7 (Tableau d'éléments non nuls, ⋆⋆)

1. *Que fait la fonction «* int notZero(int [] t) *» dont le code vous est donné ci-dessous :*

```
public static int notZero (int [] t) {
  int l = intArrayLength (t);
  int s = 0;
  for (int i = 0; i < l; i++) {
    if (t[i] != 0) {
      s = s + 1;
    }
  }
  return s;
}
```

2. *Donner une séquence d'instructions réalisant les actions suivantes :*
   (a) *Création d'un tableau* t *valant :* $\{4, 3, 6, 2, 0, 0, 2, 0, 4\}$.
   (b) *Création d'un tableau* s *contenant les éléments du tableau* t *qui sont différents de zéro (dans le même ordre). Pour cela vous utiliserez la fonction* notZero.

□

3 Fonctions utilisées

```
/*
 * Permet d'afficher un entier a l'écran
 * x est l'entier a afficher
 */
public static void printInt (int x) {
   System.out.print(x);
}

/*
```

```java
 * Affiche une chaîne de caractères.
 */
public static void printString (String s) {
    System.out.print (s);
}

/*
 * Affiche un retour de ligne à l'écran.
 */
public static void newLine () {
  System.out.println ("\n");
}

/*
 * Calcule le nombre de cases d'un tableau d'entiers
 * t est le tableau dont on veut connaitre la taille
 * Retourne la taille du tableau
 */
public static int intArrayLength(int [] t) {
   return t.length;
}

/*
 * Affiche le contenu d'un tableau d'entiers
 * t est le tableau que l'on souhaite afficher
 */
public static void printIntArray (int[] t) {
   for (int i = 0; i < t.length; i++) {
     System.out.print(t[i] + " ");
   }
   System.out.print ("\n");
}
```

4 Do it yourself

Exercice 8 (Somme, ⋆)
Écrire une fonction « int sumIntArray (int [] t) *» qui calcule la somme des éléments du tableau* t *donné en argument.* □

Exercice 9 (Moyenne, ⋆)

Écrire une suite d'instructions qui crée un tableau d'entiers t *valant* { 3, 5, 2, 3, 6, 3, 4}*, qui calcule la moyenne entière des éléments de* t *et qui affiche cette moyenne.*

□

Exercice 10 (Grognements, ⋆)

Écrire une fonction qui prend un argument entier n *et affiche les* n *premiers grognements* brhh, brrhh, brrrhh, *etc.*

□

Exercice 11 (Dernière occurrence, ⋆⋆)

Écrire une fonction « int lastOcc(int [] tab, int x) » qui prend en argument un tableau et une valeur entière. Si le tableau contient cette valeur, la fonction renvoie l'indice de la dernière occurrence de la valeur. Si le tableau ne contient pas cette valeur, la fonction renvoie -1.

□

Exercice 12 (Première occurence, ⋆⋆)

Écrire une fonction « int firstOcc (int [] tab, int x) » qui prend en argument un tableau et une valeur. Si le tableau contient cette valeur, la fonction renvoie l'indice de la première *occurrence de la valeur. Si le tableau ne contient pas cette valeur, renvoie* -1.

Indice : on pourra tester si le résultat à renvoyer est différent de -1 pour comprendre si l'on a déjà vu la valeur dans le tableau.

□

Exercice 13 (Pyramide, ⋆⋆⋆)

Écrire une fonction qui prend en entrée un entier n *et affiche une pyramide d'étoiles de hauteur* n. *Si* n *est négatif, la pyramide devra être inversée. Par exemple, sur entrée* $n = 4$, *afficher :*

```
   *
  * *
 * * *
* * * *
```

Sur entrée $n = -3$, *afficher :*

```
* * *
 * *
  *
```

□

Exercice 14 (Suite dans un tableau, ⋆⋆)

Écrire une fonction « int [] progression (int a, int b, int n) » qui prend en argument trois entiers a, b *et* n, *et qui renvoie un tableau de taille* n *contenant les entiers* a + b, a + 2b, a + 3b, ..., a + nb.

□

Exercice 15 (Entrelace, ⋆⋆)

Écrire une fonction « int [] interlace (int [] t1, int [] t2) » qui prend en entrée deux tableaux t1 *et* t2 *de même taille et renvoie un tableau de taille double qui contient les valeurs des tableaux de façon entrelacée, c'est-à-dire* {t1[0], t2[0], t1[1], t2[1], ..., t1[n], t2[n]}. *Par exemple, appliquée à* { 0, 1, 6 } *et* { 2, 4, 7 }, *elle va renvoyer le tableau* { 0, 2, 1, 4, 6, 7}.

□

Exercice 16 (Plagiat, ⋆⋆)

- *Écrire une fonction* plagiarism *prenant en argument deux tableaux* t *et* s *et qui renvoie l'indice d'une valeur de* t *qui apparaît dans* s. *Si une telle valeur n'existe pas, la fonction renvoie* -1.
- *Que se passe-t-il si on appelle la fonction* plagiarism *sur le même tableau* plagiarism (s, s) *?*
- *Écrire une fonction* autoPlagiarism *prenant en argument un tableau* s *et qui renvoie l'indice d'une valeur de* s *qui y apparaît deux fois. Si une telle valeur n'existe pas, la fonction renvoie* -1.

□

Exercice 17 (Récurrence, ⋆⋆⋆)

Soient U et V deux tableaux d'entiers à une dimension, de même taille arbitraire k. On considère l'équation de récurrence suivante :

$$u_i = U[i] \text{ pour } 0 \leq i < k \quad \text{et} \quad u_{n+1} = \sum_{i=0}^{k-1} V[i]u_{n-i} \text{ pour } n \geq k-1.$$

Écrire une fonction qui prend en argument deux tableaux d'entiers U et V (qu'on supposera de même taille) et un entier n, et qui renvoie la valeur de u_n définie ci-dessus. Attention, on ne connaît pas à l'avance la taille des tableaux. □

Exercice 18 (Mastermind, ⋆⋆⋆)

Dans cet exercice, on se propose d'écrire un programme permettant de jouer au Mastermind©.

On rappelle le principe de ce jeu. Un joueur (ici, la machine) choisit une **combinaison secrète** *de 5 couleurs parmi 8 couleurs disponibles (***on représentera durant tout l'exercice les couleurs par les entiers 0, 1, 2, 3, 4, 5, 6, 7***). Une couleur peut apparaître plusieurs fois dans une combinaison. L'adversaire de ce joueur a 10 tentatives pour deviner cette combinaison sachant qu'après chaque proposition, il bénéficie d'une indication : le premier joueur examine sa proposition et lui dit combien de pions colorés sont bien placés et combien sont présents dans la combinaison secrète, mais mal placés.*

Ainsi dans l'exemple suivant :

combinaison secrète	$\{4, 2, 6, 0, 5\}$
combinaison proposée	$\{1, 6, 4, 0, 5\}$

le joueur qui devine se verra répondre "2 pion(s) bien placé(s) et 2 pions mal placé(s)", correspondant aux couleurs 0 et 5 pour les bien placées, et aux couleurs 6 et 4 pour les mal placées.

Voici un deuxième exemple pour bien fixer les idées :

combinaison secrète	$[1, 1, 6, 0, 3]$
combinaison proposée	$[1, 6, 4, 1, 1]$

le joueur qui devine se verra répondre "1 pion(s) bien placé(s) et 2 pion(s) mal placé(s)", correspondant à la couleur 1 en position 0 bien placée et les couleurs 1 (position 3 ou 4) et 6 (position 1).

- *Écrire une fonction « public static int wp (int [] prop, int [] sol) » qui renvoie le nombre de pions bien placées, si* sol *représente la solution et* prop *la proposition.*
- *Écrire une fonction « public static int [] count (int [] t) » qui renvoie un tableau de taille 8 dont la i-ème case contient le nombre d'occurrences de i dans le tableau* t.
- *On veut désormais écrire une fonction permettant de compter le nombre de pions mal placés en se basant sur l'observation suivante. Pour une couleur donnée, si elle apparaît x fois dans la proposition et y fois dans la solution secrète, alors le nombre de pions mal placés de cette couleur plus le nombre de bien placés de cette couleur est égal au minimum de x et de y.*
  En déduire une fonction « public static int bp(int [] prop, int [] sol) » qui renvoie le nombre de couleurs mal placées. On pourra écrire une fonction « int minInt (int x, int y) » qui renvoie le minimum des deux entiers x *et* y *pris en argument.*
- *On suppose que l'on a à notre disposition une fonction « public static int [] randSol () » qui renvoie un tableau aléatoire de taille 5 constitué de valeurs de 0 à 7.*
  Écrire maintenant un programme qui simule le jeu, c'est-à-dire qu'il laisse à l'utilisateur dix tentatives pour trouver la solution en lui donnant après chaque tentative le nombre de bien placées et de mal placées. Si le joueur a trouvé la bonne solution, la boucle doit s'arrêter en affichant un message de

victoire, et si le joueur échoue après dix tentatives, le programme doit afficher un message d'échec en lui donnant la combinaison secrète. A chaque tour, l'utilisateur entrera sa combinaison en tapant les 5 chiffres successivement. Pour cela, on supposera donnée une fonction « public static int inputInt () » qui permet à l'utilisateur de saisir un entier au clavier.

□

Exercice 19 (Réordonnancement, ⋆⋆)

Écrire une fonction qui demande un entier n à l'utilisateur, puis lui demande n fois d'entrer un entier x ainsi qu'une position $j \in \{1, \ldots, n\}$ (pour l'affichage), et qui affiche ensuite les n entiers selon l'ordre spécifié. Ainsi, si l'utilisateur entre

```
4
10
3
12
2
15
4
16
1
```

(la première ligne correspondant au nombre d'entiers à entrer, puis les lignes paires sont les entiers eux-mêmes et les lignes impaires leur position), il souhaite afficher l'entier 10 en position 3, l'entier 12 en position 2, 15 en position 4 et 16 en première position. Le programme devra alors afficher :

16 12 10 15

Pour faire cela on pourra utiliser une fonction « public static int inputInt () » qui bloque tant que l'utilisateur n'a pas rentré de données au clavier et tapé sur la touche Entrée et qui renvoie alors l'entier tapé au clavier par l'utilisateur. □