Introduction à la Programmation 1 – Travaux Pratiques

Séance 8b: TABLEAUX IMBRIQUÉS

Université de Paris

Objectifs:

- Apprendre à lire, modifier et créer des tableaux imbriqués (tableaux de tableaux) d'entiers ou de chaînes de caractères.
- Chercher si une condition est satisfaite par (au moins) un des éléments d'un tableau imbriqué.
- Compter le nombre d'éléments vérifiant une condition donnée dans un tableau imbriqué.
- Trouver les positions dans un tableau imbriqué des éléments vérifiant une condition donnée.
- Créer un tableau vérifiant certains motifs.

On se familiarise avec les tableaux de tableaux (ou tableau à deux dimensions). Ils demandent naturellement l'utilisation de boucles imbriquées. Ce sujet est **long**. L'objectif est de traiter tous les thèmes en faisant le début de chaque exercice (les questions finales sont réservées à l'entraînement et au travail personnel).

Exercice 1 (Préliminaires, ⋆)

Écrire la procédure `printLines` *qui attend un tableau de tableaux d'entiers et affiche son contenu ligne par ligne (une ligne représente un* tableau imbriqué*, ou* sous-tableau*).*

On se servira de cette fonction pour tester certaines fonctions demandées dans les exercices ultérieurs.

Contrat:
Par exemple, `printLines({{1, 2, 3}, {4}, {5, 6}})` *va afficher :*

```
1 2 3
4
5 6
```

□

Exercice 2 (Création de tableaux imbriqués, ⋆⋆)

1. *(⋆) Programmer* `squareOfZeros`*, la fonction qui attend un entier* `n` *et renvoie le tableau de tableaux de dimensions* `n` × `n` *et ne contenant que des 0.*

   Contrat:
   Par exemple, `squareOfZeros(3)` *va retourner le tableau* `{{0, 0, 0}, {0, 0, 0}, {0, 0, 0}}`

2. *(⋆) Programmer* `rectOfInt`*, la fonction qui attend deux entiers* `n` *et* `p`*, et renvoie le tableau de tableaux de dimensions* `n` × `p` *contenant, dans l'ordre de lecture,* 1, 2, 3*...*

   Contrat:
   Par exemple, `rectOfInt(2, 3)` *va retourner le tableau* `{{1, 2, 3}, {4, 5, 6}}`

3. *(⋆⋆) Programmer* `triangleOfInt`*, la fonction qui attend un entier* `n`*, et renvoie le tableau triangulaire de* `n` *lignes, qui contient 1 sur sa 1re ligne, 2 et 3 sur sa 2e ligne, 4, 5, 6 sur sa 3e ligne, 7, 8, 9, 10 sur sa 4e ligne, etc.*

   Contrat:
   Par exemple, `triangleOfInt(3)` *va retourner le tableau* `{{1}, {2, 3}, {4, 5, 6}}`

4. *(⋆ ⋆ ⋆) Programmer* `target`*, la fonction qui attend un entier* `n` *et renvoie le tableau d'entiers carré de taille* `n`*, bordé par des 1 et tel que, à chaque fois qu'on se rapproche d'une case vers le centre (horizontalement ou verticalement), on augmente de 1.*

   Contrat:
   Par exemple, `target(6)` *va retourner le tableau de tableaux :*

```
1 1 1 1 1 1
1 2 2 2 2 1
1 2 3 3 2 1
1 2 3 3 2 1
1 2 2 2 2 1
1 1 1 1 1 1
```

□

Exercice 3 (Copie de tableaux imbriqués, ⋆)

1. *Écrire la fonction* `copyIntArrayArray` *qui attend un tableau de tableaux d'entiers et en renvoie une copie.*

   Contrat:
   Le code suivant :

```
int[][] arr = { {1, 2}, {3}, {4, 5, 6, 7} };
int[][] cpy = copyIntArrayArray(arr);
arr[2][2] = 999; // la copie ne doit pas \^etre modifi\'ee
printLines(cpy);
```

   doit afficher ceci :

```
1 2
3
4 5 6 7    // et pas : 4 5 999 7
```

2. *Écrire la fonction* `cutIntArrayArray` *qui attend un tableau de tableaux d'entiers* `arr` *et un entier* `n`*, et qui retourne un tableau de tableaux d'entiers de taille* `n` *contenant une copie des* `n` *premiers sous-tableaux de* `arr`*.*
   On se servira de cette fonction dans les exercices ultérieurs pour gérer les tableaux dont on ne connaît pas la taille exacte mais un majorant.

   Contrat:
   Par exemple, `cutIntArrayArray({{1}, {1, 1}, {1, 2, 1}, {1, 3, 3, 1}, {1, 4, 6, 4, 1}}, 3)` *doit retourner* `{{1}, {1, 1}, {1, 2, 1}}`*.*

□

Exercice 4 (Tableaux imbriqués d'entiers, ⋆)

Dans cet exercice, on adapte des tâches déjà posées pour les tableaux à une dimension. Vous n'êtes pas obligés de répondre à toutes les questions ! Il faut travailler les exercices suivants aussi. Faites au moins les trois ou quatre premières. Tous les programmes de cet exercice prennent (au moins) un tableau de tableaux d'entiers en argument.

1. *Programmer* `sumArrayOfArrays`*, la fonction qui attend un tableau de tableaux d'entiers et retourne la somme de tous les entiers qu'il contient.*

   Contrat:
   Par exemple, `sumArrayOfArrays({{1, 2, 3}, {4, 5}})` *doit retourner* `15`*.*

2. *Programmer* `holdsOddInt`*, la fonction qui attend un tableau de tableaux d'entiers et retourne* `true` *si et seulement si au moins un entier est impair. Elle devra être écrite de façon à ce que le programme s'arrête dès qu'il rencontre un entier impair (si le tableau en contient effectivement un).*

   Contrat:
   Par exemple, `holdsOddInt({{2, 4, 6}, {8, 10}, {12}})` *doit retourner* `false` *et* `holdsOddInt({{4, 6, 3}, {2}, {9, 11}})` *doit retourner* `true`*.*

3. *Programmer* `numberOfOneDigitInt`*, la fonction qui attend un tableau de tableaux d'entiers et retourne le nombre d'entiers (positifs) qui s'écrivent avec un seul chiffre.*

   Contrat:
   Par exemple, `numberOfOneDigitInt({{11, 3, 12}, {1, 100}})` *doit retourner* `2`*.*

4. *Programmer* `positionArray`*, la fonction qui attend un tableau de tableaux d'entiers et renvoie le tableau des positions des multiples de 9 qu'il contient.*
   Indice : *On commencera par créer un tableau de tableaux suffisamment grand pour pouvoir contenir toutes les positions. On le redimensionnera ensuite avec la fonction* `cutIntArrayArray` *une fois connu le nombre de positions.*

   Contrat:
   Par exemple, `positionArray({{9, 4, 27}, {81}, {3, 45}})` *doit retourner* `{{0, 0}, {0, 2}, {1, 0} {2, 1}}`*.*

5. *(⋆⋆) Programmer* `sumOfEvenInt`*, la fonction qui attend un tableau de tableaux d'entiers et retourne la somme de tous les entiers* pairs *qu'il contient.*

   Contrat:
   Par exemple, `sumOfEvenInt({{1, 2, 3}, {4, 5}})` *doit retourner* `6`*.*

6. *(⋆⋆) Programmer* `rowSums`*, la fonction qui attend un tableau de tableaux d'entiers et retourne le tableau constitué des sommes de chaque ligne.*

   Contrat:
   Par exemple, `rowSums({{2, 3}, {5, 8, 13, 21}, {34}})` *doit retourner* `{5, 47, 34}`*.*

7. *(⋆⋆) Programmer* `rowWiseCount`*, la fonction qui attend un entier* `n` *et un tableau de tableaux d'entiers et retourne le tableau du nombre d'entiers plus grands (strictement) que* `n` *dans chaque ligne.*

   Contrat:
   Par exemple, `rowWiseCount(10, {{12, 1}, {37, 8, -1, 21}, {0}})` *doit retourner* `{1, 2, 0}`*.*

8. *(⋆⋆⋆) Programmer* `columnSums`*, la fonction qui attend un tableau de tableaux d'entiers et retourne le tableau constitué des sommes de chaque colonne. Le programme devra tenir compte du fait que les lignes n'ont pas forcément la même longueur. Pour chaque colonne, il faut donc éventuellement sauter certaines lignes.*

   Contrat:
   Par exemple, `columnSums({{2, 3}, {5, 8, 13, 21}, {34}})` *doit retourner* `{41, 11, 13, 21}`*.*

□

Exercice 5 (Tableaux imbriqués de chaînes de caractères, ⋆⋆)

Tous les programmes de cet exercice prennent (au moins) un tableau de tableaux de chaînes de caractères en argument.

1. *(⋆) Programmer* `holdsCharlie`, *la fonction qui attend un tableau de tableaux de chaînes de caractères et retourne* `true` *si et seulement si le tableau contient la chaîne* `"Charlie"`. *Elle devra aussi être écrite de façon à ce que le programme s'arrête dès qu'il rencontre* `"Charlie"` *(si le tableau contient effectivement cette chaîne).*

   Contrat:
   Par exemple, `holdsCharlie({{"Riri", "Fifi", "Loulou"}, {"Charlie", "Georgio", "Valéry"}, {"Franz"}})` *doit retourner* `true`.

2. *Programmer* `holdsE`, *la fonction qui attend un tableau de tableaux de chaînes de caractères et retourne* `true` *si et seulement si une des chaînes de ce tableau contient la voyelle* `'e'`. *Elle devra aussi être écrite de façon à ce que le programme s'arrête dès qu'il rencontre un* `'e'` *(si une chaîne en contient effectivement un).*

   Contrat:
   Par exemple, `holdsE({{"Il", "abandonna", "son", "roman", "sur", "son", "lit"}, {"Il", "alla", "à", "son", "lavabo"}})` *doit retourner* `false`.

3. *Programmer* `fWordPositions`, *la fonction qui attend une chaîne de caractères* `s` *et un tableau de tableaux de chaînes de caractères* `arr` *et retourne les positions de la chaîne* `s` *dans le tableau* `arr`.
   Indice : *On commencera par créer un tableau de tableaux suffisamment grand pour pouvoir contenir toutes les positions. On le redimensionnera ensuite avec la fonction* `cutIntArrayArray` *une fois connu le nombre de positions.*

   Contrat:
   Par exemple, `fWordPositions("son",{{"Il", "abandonna", "son", "roman", "sur", "son", "lit"}, {"Il", "alla", "à", "son", "lavabo"}})` *doit retourner* `{{0, 2}, {0, 5}, {1, 3}}`.

□

Exercice 6 (Bonus : applications aux mathématiques, ⋆⋆⋆)

1. *Le triangle de Pascal apparaît en algèbre et en probabilités. Il est donné par une famille d'entiers* $\binom{n}{k}$, *où* n *et* k *sont des entiers naturels tels que* $0 \leq k \leq n$.
   Le triangle de Pascal se calcule par $\binom{n}{n} = \binom{n}{0} = 1$ *et* $\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$.
   Programme la fonction `pascal`, *qui attend un entier* `sz` *et retourne le tableau triangulaire d'entiers tels que le terme à la* `n`*-ème ligne et la* `k`*-ème colonne est* $\binom{\texttt{n}}{\texttt{k}}$, *pour* $\texttt{k} \leq \texttt{n} \leq \texttt{sz}$.

   Contrat:
   Par exemple, `pascal(4)` *va retourner* `{{1}, {1, 1}, {1, 2, 1}, {1, 3, 3, 1}, {1, 4, 6, 4, 1}}`.

2. *On rappelle que la suite de Fibonacci commence par 0, 1, 1, 2, 3, 5, 8, 13, 21, 34... En général, chaque terme est obtenu en effectuant la somme des deux précédents.*
   Programmer `fiboByDigits`, *la fonction qui attend un entier* `n` *et retourne le tableau des* `n` *premiers termes de la suite de Fibonacci avec la condition suivante : les termes de 1 chiffre seront sur la 1re ligne, les termes de 2 chiffres seront sur la 2e ligne, les termes de 3 chiffres sur la 3e ligne et ainsi de suite.*

   Contrat:
   Par exemple, `fiboByDigits(13)` *va retourner* `{{0, 1, 1, 2, 3, 5, 8}, {13, 21, 34, 55, 89}, {144}}`.

   Afficher `fiboByDigits(43)`. *Que remarque-t-on ?*

□