Séance 5 de travaux pratiques

1 Tableaux de chaînes de caractères

Tableau de chaînes de caractères (⋆)

tableaudeString [q1]

A la fin de l'exécution du code suivant, que contient `tab[0]` ?

```
String[] tab = {"chien", "petit", "un", "gambade"};
tab[0] = tab[2];
tab[2] = "mouton";
tab[3] = "␣" + tab[3];
tab[2] = "␣" + tab[2] + tab[3];
tab[1] = "␣" + tab[1] + tab[2];
tab[0] = "Un" + tab[1];
```

- ☐ un entier
- ☐ une chaîne de caractères
- ☐ l'entier 42
- ☐ l'entier 0
- ☐ la chaîne "mouton"
- ☐ la chaîne "un petit chien gambade"
- ☐ la chaîne "chien petit un gambade"
- ☐ la chaîne "Un petit chien gambade"
- ☐ la chaîne "mouton petit mouton gambade"
- ☐ la chaîne "Un petit mouton gambade"
- ☐ la chaîne "Un mouton petit gambade"
- ☐ la case n'est pas définie

D'un tableau à une chaîne de caractères (⋆)

tableaudeint String boucleavecaccumulateur [q2]

Compléter la fonction `arrayToString` qui attend un tableau d'entiers et qui renvoie une chaîne représentant ce tableau.

Par exemple, si le tableau est `{42,1,3}`, la fonction renverra la chaîne `"{42,1,3}"`.

▷ `ArrayToString.java`

Produit cartésien de chaînes (⋆⋆)

tableaudeString bouclesimbriquées boucleavecdépendance liretableau [q3]

Compléter la procédure `cartesianProd` qui prend en paramètres deux tableaux de String et qui affiche toutes les paires de mots possibles, une par ligne, avec un espace entre les deux mots de chaque paire.

Par exemple, si le premier tableau est `{"chat", "chien"}` et le second tableau est `{"noir", "blanc", "marron"}`, la procédure affichera :

```
chat noir
chat blanc
chat marron
chien noir
chien blanc
chien marron
```

▷ `CartesianProduct.java`

Produit cartésien de chaînes : le retour (⋆⋆⋆)

tableaudeString tableaudetableaux boucleavecdépendance liretableau définirtableau [q4]

Compléter la fonction `getCartesianProd` qui prend en paramètres deux tableaux de String et qui renvoie un tableau de tableaux contenant toutes les paires de mots possibles, comme sur l'exemple ci-dessous.

Par exemple, si le premier tableau est `{"chat", "chien"}` et le second tableau est `{"noir", "blanc", "marron"}`, la fonction retournera le tableau :

```
{{"chat", "noir"},
 {"chat", "blanc"},
 {"chat", "marron"},
 {"chien", "noir"},
 {"chien", "blanc"},
 {"chien", "marron"}}
```

▷ `CartesianProductReturn.java`

Objet le plus cher (⋆⋆)

tableaudeint tableaudeString boucleavecaccumulateur conditionnelledansboucle [q5]

Compléter la fonction `getMostExpensive` qui prend en paramètres un tableau de noms d'objets (avec au moins un objet) et un tableau contenant leurs prix (supposés tous différents), et qui renvoie le nom de l'objet le plus cher.

Par exemple, pour les tableaux `{"timbre", "voiture"}` et `{1,15000}`, la fonction renverra `"voiture"`.

▷ `MostExpensive.java`

2 Tableaux de tableaux

Tableau de tableaux (⋆)

tableaudetableaux [q6]

Après l'exécution du code suivant, quelles propositions sont vraies ?

```
int[] t1 = {1, 4, 2};
int[] t2 = {8, 1, 3, 5};
int[][] tab = new int[2][];
tab[0] = t1;
tab[1] = t2;
tab[0][1] = tab[1][3];
```

- ☐ tab[1][3] est bien défini
- ☐ tab[0][3] est bien défini
- ☐ tab[1] est un tableau à 3 éléments
- ☐ tab[1] est un tableau à 4 éléments
- ☐ tab est un tableau à 2 éléments

☐ tab est un tableau de tableaux
☐ tab[0][1] contient la valeur 3
☐ tab[0][1] contient la valeur 5
☐ tab[0][1] contient la valeur 1
☐ tab[1][0] contient la valeur 8
☐ tab[1][0] contient la valeur 1

A□cher un tableau à 2D (⋆) tableaudetableaux tableaudeint bouclesimbriquées boucleavecdépendance tableaudeint liretableau [q7]

Compléter la procédure `disp2Darray` qui attend un tableau de tableaux d'entiers et les affiche ligne par ligne. Chaque ligne est suivie d'un retour charriot, chaque entier est suivi d'un espace (même le dernier).

Par exemple, si le tableau est `{{1,2,3},{4,5}}`, la procédure devra afficher :

```
1 2 3
4 5
```

▷ `Display2DArray.java`

A□che le tableau et tu verras de beaux dessins ! (⋆) tableaudetableaux tableaudeString liretableau paramètretableau boucleavecdépendance bouclesimbriquées [q8]

Compléter la procédure `showArray` qui attend un tableau de tableaux de chaînes de caractères, et qui affiche chacun des sous-tableaux sur une ligne (sans espaces entre les éléments).

Par exemple, si le tableau est `{{".",".","*"}, {".", "!", "."}}`, la procédure affichera :

```
..*
.!.
```

▷ `AsciiArt.java`

Scrabble (⋆⋆⋆) boucleavecdépendance conditionnelledansboucle tableaudetableaux tableaudeString liretableau paramètretableau bouclesimbriquées [q9]

Compléter la fonction `getLetterPoints` qui prend en paramètre une chaine de caractère `letter` contenant une seul lettre et un tableau de tableaux de chaines de caractères `letterValues` tel que `letterValues[i]` est le tableau contenant toutes les lettres de valeur `i`, et qui renvoie la valeur de la lettre `letter`.

Ensuite, compléter la fonction `getWordPoints` qui prend en paramètre un mot `word` et qui renvoie le score du mot, c'est-à-dire la somme des valeurs des lettres du mot.

La fonction `getWordPoints` appelera la fonction `getLetterPoints`.

▷ `Scrabble.java`

Cible carrée de fléchettes (⋆⋆⋆) tableaudetableaux conditionnellesimple définirtableau bouclesimbriquées [q10]

Commencer par compléter la fonction `min` qui prend quatre paramètres entiers `x1`, `x2`, `x3` et `x4`, et qui renvoie le plus petit des quatre.

Puis, compléter la fonction `getDartboard` qui prend en paramètres un entier strictement positif `n` et qui renvoie un tableau de tableaux d'entiers, carré de côté `n`, et représentant une cible de fléchettes carrées.

Par exemple, pour `n=6`, le tableau retourné sera

```
{{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}}
```

On utilisera évidemment la fonction `min`.

▷ `SquareDartboard.java`

3 Boucles while

Boucles while (⋆)

bouclewhile [q11]

Quelle est la valeur de la variable `m` à l'issue de l'exécution du code suivant ?

```
int m = 49 * 2 + 23;
int n = 11;
int i = 1;
while (2 * i < 21) {
    i = i + 1;
    m = m - n;
}
```

☐ 11
☐ 0
☐ 121
☐ 49
☐ 23

Jet d'un dé (⋆)

bouclewhile conditionnelledansboucle boucleavecaccumulateur [q12]

On fournit une fonction `rollDice()` sans paramètres, et qui fournit un entier entre 1 et 6 correspondant au jet d'un dé. On veut compter le nombre de jets nécessaires pour obtenir `n` fois la valeur 6.

Programmez une fonction `countRolls(n)` qui prend en paramètre un entier `n`, appelle la fonction `rollDice()` jusqu'à l'obtention de `n` fois la valeur 6, puis renvoie le nombre d'appels à `rollDice()` (autrement dit, le nombre de jets de dés nécessaires à l'obtention de ces `n` 6).

▷ `RollDice.java`

Syracuse (⋆)

bouclewhile conditionnelledansboucle boucleavecaccumulateur [q13]

La suite de Syracuse est une suite mathématique extrêmement simple à décrire, mais qui a donné lieu à une conjecture réputée impossible à prouver.

On commence par définir la fonction `f` suivante : si `n` est pair, alors `f(n) = n / 2`, sinon `f(n) = 3 * n + 1`. La suite de Syracuse partant d'un entier `n` est la suite `n, f(n), f(f(n)), f(f(f(n)))`.....

La conjecture dit que quel que soit l'entier `n`, la suite partant de `n` finit par atteindre `1` – et boucler ensuite sur la séquence `1, 4, 2`.

Malgré la simplicité apparente de cette conjecture, le mathématicien Paul Erdős avait dit que « les mathématiques ne sont pas encore prêtes pour un tel problème. »

Avant de vous lancer dans la résolution de cette conjecture, entraînez-vous un peu avec de la science expérimentale. À cette fin, vous compléterez la fonction `syracuseLength` qui prend un paramètre `n` et renvoie la longueur de la suite de point de départ `n`, en s'arrêtant au premier `1` rencontré.

Par exemple, `syracuseLength(5) = 6`, car les valeurs de la suite sont `5, 16, 8, 4, 2, 1`.

▷ `Syracuse.java`

Écriture décimale d'un entier (⋆⋆)

tableaudeint bouclewhile boucleavecaccumulateur modulo définirtableau [q14]

1. Compléter la fonction `numDigits` qui prend en paramètre un entier positif ou nul `n`, et qui renvoie le nombre de ses chiffres en base 10.

On pourra utiliser le fait que le nombre de chiffres d'un entier strictement positif est le nombre de fois qu'on peut diviser cet entier par 10 avant qu'il ne devienne < 1. On fera bien attention au fait que `0` a 1 chiffre.

2. Compléter la fonction `getDigits` qui prend en paramètre un entier positif ou nul `n`, et qui renvoie un tableau contenant ses chiffres (écrits de gauche à droite).

Par exemple, pour `n=1454`, `numDigits` renvoie 4 et `getDigits` renvoie le tableau `1,4,5,4`.

▷ `Digits.java`

4 Variables booléennes

Variables booléennes (⋆)

boolean [q15]

Considérons la fonction `f` suivante :

```
static boolean f(boolean p, boolean q, boolean r, boolean s) {
    boolean a = p && !q;
    boolean b = r && s;
    boolean c = q && r && !s;
    return (a || b || c);
}
```

Lesquelles de ces expressions valent `true` ?

☐ f(true, true, true, false)
☐ f(false, true, false, true)
☐ f(true, false, true, false)
☐ f(true, false, false, true)

"OU Exclusif" (⋆)

boolean [q16]

Le "ou exclusif", ou "xor" pour son abréviation anglaise, est une fonction prenant en entrée deux booléens, et renvoyant `false` s'ils ont la même valeur et `true`, s'ils sont différents. Par exemple, `xor(true, false) = true` alors que `xor(true, true) = false`.

Ecrivez une fonction `xor` prenant en paramètre deux booléens et qui renvoie le "ou exclusif" de ces deux bouléens.

▷ `Xor.java`

Liste de nombres premiers (⋆⋆)

boolean bouclesimbriquées conditionnelledansboucle [q17]

Un nombre entier `n` est premier s'il est au moins égal à `2` et s'il n'est multiple que de `1` et de `n`.

On rappelle que l'on peut tester si un nombre `n` est multiple d'un nombre `m` à l'aide de la condition booléenne `n % m == 0`.

Compléter la procédure `showPrimes` qui prend un entier `m` et affiche sur des lignes distinctes tous les entiers premiers compris entre `2` et `m`, par ordre croissant. Par exemple, `showPrimes(10)` affiche :

```
2
3
5
7
```

▷ `ShowPrimes.java`

Liste de nombres premiers (⋆⋆)

boolean bouclesimbriquées conditionnelledansboucle bouclewhile tableaudeint [q18]

Un nombre entier `n` est premier s'il est au moins égal à `2` et s'il n'est multiple que de `1` et de `n`.

On rappelle que l'on peut tester si un nombre `n` est multiple d'un nombre `m` à l'aide de la condition booléenne `n%m==0`.

Compléter la procédure `listPrimes` qui prend un entier `n` et renvoie le tableau des `n` plus petits entiers premiers. Par exemple, `listPrimes(5)` renvoie le tableau `{2,3,5,7,11}`.

▷ `CollectPrimes.java`

5 Bonus

Plouf, plouf, ... (⋆⋆⋆)

int String boucleavecaccumulateur modulo conditionnellesimple [q19]

On rappelle le jeu du "Plouf, Plouf". On choisit une comptine comme "une carte en or, c'est toi qui sors". Les joueurs sont ensuite numérotés de `1` à `n` et positionnés en cercle. Le meneur (qui n'est pas un joueur) égrène la comptine et pointe du doigt chacun des joueurs à tour de rôle à chaque mot de la comptine choisie.

Par exemple, pour trois joueurs, le meneur, pointe le joueur 1 sur le mot "une", le joueur 2 sur "carte", 3 sur "en", 1 sur "or", ... Le dernier joueur pointé (ici le numéro 2) est éliminé.

Complétez le fonction `findOut` qui prend en paramètre la comptine et le nombre `n` de joueurs (vous compris) et qui renvoie le numéro du joueur éliminé. La place est un entier entre `1` et `n`.

On supposera que deux mots sont nécessairement séparés par un unique espace.

▷ `PloufPlouf.java`