Exercices supplémentaires - Tris et Chaînes

Projet de programmation (IK2)

Les tris

Dans chacun des exercices, il est demandé d'écrire une méthode de la forme suivante :
`static double[] NomDuTri(double[] tab){...}`
L'utilisation de méthodes auxiliaires est parfois utile.

Exercice 1 — *Tri par insertion* L'algorithme consiste à trier les i premiers éléments du tableau, pour chaque i entre 1 et la taille du tableau. Ainsi, pour trier les $(i+1)$ premiers éléments, on utilise la liste triée des i premiers, calculée préalablement, puis on insère le $(i+1)^{\text{e}}$élément au bon endroit.

Exercice 2 — *Tri par recherche de minimum* L'algorithme consiste à chercher le minimum des éléments du tableau, puis le placer en première case du tableau de résultat. Plus généralement, on recherche l'élément minimum du tableau mis-à-part ceux que l'on a déjà prélevés (disons qu'il y en a i), on le place dans la $(i+1)^{\text{e}}$case du tableau-résultat, et on recommence.

Exercice 3 — *Tri-bulle* L'algorithme consiste à parcourir le tableau jusqu'à trouver deux éléments consécutifs qui ne soient pas dans le bon ordre, les permuter, et recommencer du début, jusqu'à ce que tout le tableau soit trié.

Exercice 4 — *Tri rapide (quick-sort)* L'algorithme consiste à répartir les éléments du tableau dans deux tableaux `tab1` et `tab2` (non-vides), de manière à ce que chaque élément de `tab1` soit plus petit (au sens large) que tous les éléments de `tab2`. On appelle alors la méthode de tri (appel récursif) sur les tableaux `tab1` et `tab2` puis on les concatène.

Exercice 5 — *Tri-fusion* L'algorithme consiste à scinder le tableau en deux tableaux `tab1` et `tab2` (sans réarranger les éléments). On appelle alors la méthode de tri (appel récursif) sur les tableaux `tab1` et `tab2` puis on les fusionne en faisant attention à l'ordre des éléments.

Les chaînes

Exercice 6 — *Arguments de la ligne de commande* Écrire un programme qui affiche tous les arguments qui sont passé sur la ligne de commande. Par exemple :

```
bash ~\$ java MonProg titi toto tutu
titi
toto
tutu
```

Exercice 7 — *Minuscules* Écrire une méthode qui lit une phrase puis mets toutes ses lettres en minuscules. Par exemple, *Hello World !* devient *hello world !*.

Exercice 8 — *Palindrome* Écrire une méthode qui lit un mot et teste si celui-ci est un palindrome. Par exemple, pour *radar* la méthode doit répondre `true` et pour *avion* elle doit répondre `false`.

Exercice 9 — *Comptage des points au Scrabble.* Écrire une méthode qui compte la valeur en points d'un mot entré par l'utilisateur, selon le barème du Scrabble ('a', 'e', 'i', 'l', 'n', 'o', 'r', 's', 't', 'u' valent 1 point, 'd', 'm', 'g' valent 2 points, 'b', 'c', 'p' valent 3 points, 'f', 'h', 'v' valent 4 points, 'j', 'q' valent 8 points, et enfin 'k', 'w', 'x', 'y', 'z' valent 10 points).

Exercice 10 — *Extraction de voyelles et consonnes.* Écrire deux méthodes `ListeVoyelles` et `ListeConsonnes` qui prennent en entrée une chaîne et qui retournent respectivement la liste des voyelles et des consonnes contenues dans la chaîne. (Pour vérifier ses méthodes, on écrira une méthode d'affichage d'une chaîne.)

Exercice 11 — *Conjugaison des verbes du premier groupe.* Écrire une méthode qui lit un verbe à l'infinitif, qui s'assure que ce verbe est du premier groupe, puis qui le conjugue au présent de l'indicatif. (On supposera le verbe régulier, pour éviter le problème posé par exemple par *manger*.)

Exercice 12 — *Divers ordres sur les mots.* Écrire, pour chacun des ordres suivant, une méthode qui prend en arguments deux mots $m = a_1 \cdots a_n$, $m' = a'_1 \cdots a'_{n'}$ et qui les affiche trié dans l'ordre croissant :

1. Ordre lexicographique : $m < m'$ si $a_1 < a'_1$ ou $(a_1 = a'_1$ et $(a_2 < a'_2$ ou $(a_2 = a'_2$ et...$)))$.
2. Longueur des mots.
3. Nombre de voyelles.
4. Inclusion stricte : $m_1 < m_2$ si les lettres de m_1 apparaissent dans m_2 et dans le même ordre (exemple : *ab* est $<$ à *xaybz* mais pas à *xbyaz*).
5. Inclusion large : $m_1 < m_2$ si les lettres de m_1 apparaissent dans m_2, éventuellement en désordre.