

Développement initiatique

Sujet 10 : Listes chaînées

Récupérer sur l'ENT les classes `Liste`, `Maillon` et `MainListe` et les compléter avec les algorithmes ci-dessous. La classe `Liste` implante les listes chaînées d'éléments de type entier.

1 Algorithmes de base sur les listes

Question 1 Longueur d'une liste

Ecrire une fonction : `public int longueur()`.

`/* Résultat : le nombre d'éléments de la liste 'this'. */`

Question 2 Somme des éléments

Ecrire une fonction : `public int somme()`.

`/* Résultat : la somme des éléments de la liste 'this'. */`

Question 3 Dernier élément

Ecrire une fonction : `public int dernierElt()`.

`/* Prérequis : 'this' est non vide. */`

`/* Résultat : la valeur du dernier élément de la liste 'this'. */`

Question 4 Longueur supérieure à k ?

Ecrire une fonction : `public boolean estSupK(int k)`.

`/* Prérequis : k est un entier positif */`

`/* Résultat: vrai ssi la longueur de la liste 'this' est supérieure ou égale à k. */`

Ecrire une version en une ligne, mais effectuant dans certains cas des calculs inutiles, et une autre version `estSupKBis` plus longue mais évitant ces calculs inutiles.

Question 5 Longueur paire ?

Ecrire une fonction : `public boolean aLgPaire()`.

`/* Résultat : vrai ssi la liste 'this' possède un nombre pair d'éléments. */`

Question 6 Maximum des éléments

Déterminer la valeur maximum des éléments d'une liste.

Prérequis : la liste est non vide.

Question 7 Nombre d'occurrences

Déterminer le nombre d'occurrences d'un entier n dans une liste.

Question 8 Ajout en fin de liste

Ecrire une méthode : `public void ajoutFin(int n).`

```
/* Action : ajoute un élément de valeur  $n$  comme dernier élément de la liste
this. */
```

Question 9 Ajout en fin de liste *si absent*

Ecrire une méthode : `public void ajoutFinSiAbsent(int n).`

```
/* Action : ajoute un élément de valeur  $n$  comme dernier élément de la liste
'this' au cas où la liste ne possède pas déjà un élément valant  $n$ . */
```

Question 10 Extraction des éléments impairs

Ecrire une fonction : `public Liste extractionImpairs().`

```
/* Résultat : une nouvelle liste contenant les éléments de valeur impaire de 'this'
dans chacun des 2 cas suivants : (a) l'ordre des éléments de la liste retournée
n'a pas d'importance, (b) l'ordre doit être le même que dans this. */
```

Remarque : Une approche permettant de préserver l'ordre des éléments tout en restant efficace est introduite dans le premier bonus.

Question 11 Suppression d'un élément

Supprimer de la liste `this` la première occurrence d'un entier n , si elle existe.

Question 12 Troncation après le $k^{\text{ème}}$ élément

Lorsque la liste `this` est de longueur supérieure à k , la tronquer après son $k^{\text{ème}}$ élément.

Question 13 Clones

Déterminer si deux listes sont clones l'une de l'autre, c'est-à-dire ont les mêmes valeurs rangées dans le même ordre.

Question 14 Liste à l'envers

Ecrire une fonction : `public Liste inverse().`

```
/* Résultat : une nouvelle liste contenant les éléments de 'this' dans l'ordre
inverse. */
```

Question 15 Liste à l'envers, récursive et en place (bonus difficile)

Ecrire une fonction : `public void inverseRec().`

```
/* Action : inverse les éléments de 'this'. Suggestion : fait appel à une fonction
récursive "public Maillon inverseRec (Maillon m)" qui, sans créer de nouveau maillon,
inverse la liste à partir du maillon m et renvoie le nouveau maillon de tête. */
```

2 Exercices difficiles

Pour ces exercices difficiles, il est fortement recommandé de bien *tester* vos algorithmes. Vous trouverez en commentaires de `MainListe.java` des exemples compliqués.

Question 16 Suppression de toutes les occurrences d’une valeur

Ecrire une fonction : `public void suppToutesOcc(int n)` sans faire appel à la fonction de la question 11.

```
// Action : supprime de la liste 'this' toutes les occurrences d'un entier n.  
// Remarque : une version simple à écrire mais coûteuse fait appel aux méthodes  
// précédentes ; la version demandée ne parcourt la liste qu'une seule fois.
```

Question 17 Sous-liste Déterminer si une liste est une sous-liste (éléments consécutifs) d’une autre liste.

3 Bonus : des listes plus riches

Question 18 Créer *le plus vite possible* une classe `Liste2` (testée par `MainListe2`) qui a des spécifications très proches de `Liste`, mais qui contient deux autres attributs pour améliorer les performances en temps de calcul CPU :

- un attribut `longueur` de type entier qui stocke la longueur de la liste `this` ;
- un attribut `dernier` qui est une référence sur le dernier maillon de la liste `this`. Cet attribut permet d’améliorer les performances des méthodes travaillant en fin de liste.

4 Bonus : affichage de polygones dans une fenêtre

Modifier (en copiant) la classe `Liste` en une classe `ListePoints` (sans reprendre toutes les méthodes) et avec un attribut supplémentaire permettant de référencer le dernier élément d’une liste. Copier aussi dans le même répertoire que `Liste.java` les fichiers du répertoire `graphisme` sous l’ENT.

Question 19 Créer une classe `Polygone` décrite par une liste de points (le dernier étant relié au premier par un segment) et une couleur (R,G,B).

Question 20 Ecrire un constructeur de `Polygone`, qui en plus dessine la figure géométrique correspondante sur la fenêtre.

Question 21 Ajouter une méthode `ajoutPoint(Point p, int k)` à la classe `ListePoints` qui ajoute un point p à la liste au rang k .

Ecrire une méthode `ajoutPoint(Point p, int k)` de `Polygone`.

Question 22 Ecrire une méthode `concatener` de `Polygone` prenant un polygône p en paramètre. La procédure remplace `this` par le polygône obtenu à partir des deux polygones `this` et p en reliant, pour chacun d’eux, le dernier point de sa liste de points au premier point de l’autre.