Exercices sur les listes

Exercice 1

Soit L une liste d'entiers triés par ordre croissant. Écrire l'algorithme logique de la fonction qui retourne la liste des éléments de L triés par ordre décroissant.

Exercice 2

Soit L une liste d'entiers. Ecrire l'algorithme logique de la fonction qui supprime tous les entiers pairs de L (interdiction de créer une autre liste).

Exercice 3

On suppose que la liste de l'exercice 2 est représentée dans un tableau de manière chaînée. Ecrire l'algorithme de programmation correspondant à l'algorithme logique de l'exercice 2.

Exercice 4

On stocke, par ordre alphabétique dans une liste symétrique L, les habitants d'une commune représentés par leur nom et leur adresse. Ecrire l'algorithme logique de la fonction qui, à partir du nom d'un habitant et s'il existe, affiche l'adresse de l'habitant le précédant dans l'ordre alphabétique s'il y en a un.

Exercice 1

```
Fonction Reverse (L :Liste(entier)) :Liste(entier)
    Début
    L2 ← lisVide()
    P ← tête(L)
    Tantque non finliste(L,P) alors
        V ← val(L,P)
        Adjtlis(L2,V)
    Fintantque
    Retourne L2
    Fin


Lexique
    L : Liste(entier), liste initiale
    L2 : Liste(entier), liste inversée
    P : Place, place courante
    V : entier, valeur dans la liste
```

Exercice 2

```
Fonction SupprimePairs (L InOut :Liste(entier))
    Début
    P ← tête(L)
    Tantque non finliste(L,P) alors
        V ← val(L,P)
        P2 ← suc(L,P)
        Si V MOD 2 = 0 alors
            suplis(L,P)
        P ← P2
    ftantque
    Fin

Lexique
    L : Liste(entier), liste d'entiers
    P : Place, place courante
    V : entier, valeur dans la liste
    P2 : Place, place suivant P
```

Exercice 3

```
Fonction SupprimePairsChaînée (L  InOut :Liste(entier))
        Début
        P ← L.tête
        P2 ← L.tête
        Tantque L.tab[P].suc ≠ 0 alors
                V ← L.tab[P].val
                P3 ← L.tab[P].suc
                Si V MOD 2 = 0 alors
                        Si P = P2 alors
                                L.tête = P3
                        Sinon
                                L.tab[P2].suc = P3
                        Fsi
                Sinon
                        P2 ← L.tab[P2].suc
                Fsi
                P ← P3
        Ftantque
        Fin

Lexique
        L : Liste(entier), liste d'entiers
        P : Place, place courante
        V : entier, valeur dans la liste
        P2 : Place, place précédant P
        P3 : Place, place suivant P
```

Exercice 4

```
Fonction prédécesseurHabitant (L :ListeSym(Habitant), nom :chaîne)
    Début
    P ← tête(L)
    Tantque non finls(L,P) faire
        V ← val(L,P)
        Si nom = V.nom faire
            P2 ← precls(L ,P)
            Si non finls(L,P) faire
                V2 ← val(L,P2)
                Ecrire(V2.adresse)
            Fsi
        Fsi
        P ← sucls(L,P)
    ftantque
    Fin

Lexique
    Habitant =<nom : chaîne, adresse, chaîne>
    L : Liste(Habitant), liste d'habitants
    P : Place, place courante
    V : Habitant, valeur dans la liste
    P2 : Place, place précédant P
    V2 : Habitant, valeur de la place P2
```

Exercice 5 (Polycopié page 39)

```
Fonction lOuvrageRechmotCle(lOuvrage : Liste(Ouvrage),motCléCherché : chaîne)
Début
pLOuv ← tête(lOuvrage)
tantque non finliste(lOuvrage,pLOuv) faire
        ouvrage ← val(lOuvrage,pLOuv)
        lMotClé ← ouvrage.motsClés
        pLMot ← tête(lMotClé)
        arrêt ← faux
        tantque non finliste(lMotClé,pLMot) et non arrêt faire
                motCl&Liste ← val(lMotClé,pLMot)
                si (motCléListe = motCl&Cherché)
                        alors arrêt ← vrai
                        sinon pLMot ← suc(lMotClé,pLMot)
                fsi
        fintantque
        si (arrêt = vrai)
                alors écrire(ouvrage.ref,ouvrage.titre)
        fsi
        pLOuv ← suc(lOuvrage,pLOuv)
fintantque
fin
```