Algorithmique et programmation procédurale - TD No 5

Liste - CORRIGE

Exercice 1. Quelle est la notation polonaise (ou postfixée) des expressions suivantes ?

a & (b | c | d)
d – ((56 * a) / (37 – b)) * (5 – c)

Corrigé

a b c | d | &

d 56 a 37 b - / 5 c - * -

Exercice 2. Écrire un algorithme ajoutant un élément dans une liste simplement chaînée et triée en ordre croissant.

Corrigé : Voici l'intégralité du code nécessaire

```
/* Principe : si on utilise la boucle suivante pour parcourir la liste
        Tantque (l != NULL) et (valeur(l) < val)
                l <- suivant(l)
        FinTantque
il y a 2 possibilités :
    -   l = NULL : on arrive à la fin de la liste, pour ajouter la valeur val, on n'a
        plus le lien vers le prédécesseur de l, il faut faire comme la fonction
        ajouterFin pour re-parcourir la liste et arrêter cette fois plus tôt.
    -   l != NULL : il faut ajouter val juste avant le nœud pointé par l => astuce pour
        échanger les valeurs sans oublier les cas particuliers !
Donc c'est plus simple d'utiliser cette boucle :
        Tantque (suivant(l) != NULL) et (valeur(suivant(l)) < val)
                l <- suivant(l)
        FinTantque
pour s'arrêter plus tôt, l va pointer sur le nœud dont la valeur est inférieure à val,
il suffit d'ajouter val après l, mais il faut différencier le cas quand la liste n'a
qu'un seul élément, … */


Fonction InsererListeTrie(teteListe : Pointeur, val : T) : Pointeur
Variables l, p : Pointeur
Début
        // créer le nœud à insérer

        p ← créerNoeud()
        valeur(p) ← val
        suivant(p) ← NULL
```

```
Si (teteListe = NULL) Alors
        // si la liste est vide, retourner le nœud
        retourner p
Sinon
        Si (suivant(teteListe) = NULL) Alors
                // si la liste n'a qu'un seul élément
                Si (valeur(teteListe) < val) Alors
                        suivant(teteListe) ← p
                        retourner teteListe
                Sinon
                        suivant(p) ← teteListe
                        retourner p
                Finsi
        Sinon
                /* si la liste a au moins 2 éléments, on parcourt la liste, il y
                a 2 possibilités :
                - l'élément à ajouter est supérieur à tous les éléments de la
                liste, on l'ajoute à la fin
                - on s'arrête juste avant l'élément supérieur à val, on l'ajoute
                à cet endroit */

                l ← teteListe
                Tantque (suivant(l) ≠ NULL) et (valeur(suivant(l)) < val)
                        l ← suivant(l)
                FinTantque
                Si (suivant(l) != NULL) Alors
                        // il y a encore des éléments après val
                        suivant(p) ← suivant(l)
                FinSi
                // dans les 2 cas, on fait le lien entre suivant de l et p
                suivant(l) ← p
                retourner teteListe
        FinSi
Fin
```

Exercice 3. Écrire un algorithme qui permet de supprimer dans une liste :

1. le premier élément dont la valeur égale à une valeur donnée
2. tous les éléments dont la valeur égale à une valeur donnée
3. toutes les occurrences du minimum d'une liste.

Corrigé

```
Fonction supprimerOccurrence(teteListe : Pointeur, x : T) : Pointeur
Variables l, p : Pointeur
Début
        l ← teteListe
        Tantque (l ≠ NULL) et (valeur(l) ≠ x)
                l ← suivant(l)
        FinTantque
        Si (l = NULL) Alors
                // x n'est pas dans la liste
                retourner teteListe
        Sinon
                // x est dans la liste
```

```
                Si (suivant(l) = NULL)
                        // x est le dernier élément
                        retourner supprimerFin(teteListe)
                Sinon
                        // x n'est pas le dernier élément,
                        // échanger avec l'élément suivant
                        q ← suivant(l)
                        valeur(l) ← valeur(q)
                        suivant(l) ← suivant(q)
                        liberer(q)
                        retourner teteListe
                FinSi
        FinSi
Fin

Fonction supprimerOccurrences(teteListe : Pointeur, x : T) : Pointeur
Variables l : Pointeur
Début
        l ← teteListe
        Tantque (l ≠ NULL)
                Si (x = valeur(l)) alors
                        Si (suivant(l) = NULL)
                                // x est le dernier élément
                                retourner supprimerFin(teteListe)
                        Sinon
                                // x n'est pas le dernier élément,
                                // échanger avec l'élément suivant
                                q ← suivant(l)
                                valeur(l) ← valeur(q)
                                suivant(l) ← suivant(q)
                                liberer(q)
                        FinSi
                Sinon
                        l ← suivant(l)
                FinSi
        FinTantque
        Retourner teteListe
Fin

Fonction supprimerMins(teteListe : Pointeur, x : T) : Pointeur
Variables l : Pointeur, min : T
Début
       Si (teteListe != NULL) Alors
              l ← teteListe
              min ← valeur(l)
              l ← suivant(l)
              Tantque (l != NULL)
                     Si (valeur(l) < min) Alors
                            min ← valeur(l)
                     FinSi
                     l ← suivant(l)
              FinTantque
              Retourner supprimerOccurences(teteListe,min)
        Sinon
              Retourner teteListe
        FinSi
Fin
```

Exercice 4.

1. Écrire un algorithme qui permet de concaténer deux listes (copie !).
2. Écrire un algorithme concatène une liste à une autre liste (sans copie !).

Corrigé

```
Fonction concatener1(l1 : Pointeur, l2 : Pointeur) : Pointeur
Variables res, l1tmp, l2tmp : Pointeur
Début
        res ← NULL                  // créer une liste vide
        l1tmp ← l1
        Tantque (l1tmp  ≠ NULL)
                res ← ajouterFin(res, valeur(l1tmp)
                l1tmp ← suivant(l1tmp)
        FinTantque
        l2tmp ← l2
        Tantque (l2tmp  ≠ NULL)
                res ← ajouterFin(res, valeur(l2tmp)
                l2tmp ← suivant(l2tmp)
        FinTantque
        retourner res
Fin

Fonction concatener2(l1 : Pointeur, l2 : Pointeur) : Pointeur
Variables l1tmp: Pointeur
Début
        Si (l1 = NULL) Alors
                Retourner l2
        Sinon
                l1tmp  ← l1
                Tantque (suivant(l1tmp)  ≠ NULL)
                        l1tmp ← suivant(l1tmp)
                FinTantque
                suivant(l1tmp) ← l2
                retourner l1
        FinSi
Fin
```

Exercice 5

1. Écrire un algorithme qui crée l'inverse d'une liste (copie !).
2. Écrire un algorithme qui inverse une liste. (sans copie !).

Corrigé

```
Fonction inverser1(teteListe : Pointeur) : Pointeur
Variables res, l : Pointeur
Début
       res ← NULL
       l ← têteListe
       Tantque l != null
              res ← ajouterTete(res, valeur(l))
       Fintantque
       retourner res
Fin
```

```
Fonction inverser2(teteListe : Pointeur) : Pointeur
Variables l, p, q : Pointeur
Début
        l ← têteListe
        q ← nul
        p ← nul
        Tant que l != null
                p ← l
                l ← suivant(l)
                suivant(p) ← q
                q ← p
        Fintantque
        retourner q
Fin
```

Exercice 6

Écrire un algorithme qui permet de copier une liste de départ dans une nouvelle liste **sans doublons**. Par exemple, {2, 6, 7, 7, 3, 2, 8} devient {2, 6, 7, 3, 8}.

Corrigé

```
Fonction copier(teteListe : Pointeur) : Pointeur
Variables res, l1, l2 : Pointeur
Début
        res ← NULL
        l1 ← teteListe
        Tantque (l1 ≠ NULL)
                Si res = NULL Alors
                        res ← creerNoeud()
                        valeur(res) ← valeur(l1)
                        suivant(res) ← NULL
                Sinon
                        // on cherche si valeur(l1) est déjà dans res
                        // en utilisant l2 pour parcourir res
                        l2 ← res
                        Tantque (l2 ≠ NULL) et (valeur(l2) ≠ valeur(l1))
                                l2 ← suivant(l2)
                        FinTantque
                        Si (l2 = NULL) Alors
                                // valeur(l1) n'est pas dans res, on l'ajoute à la fin
                                res ← ajouterFin(res, valeur(l1))
                        FinSi
                FinSi
                l1 ← suivant(l1)
        Fintantque
        Retourner res
Fin
```