Université Bordeaux 1 LST Mention Informatique Semestre 3 2010-2011

Algorithmique 1 : Devoir Surveillé 1

Types de données abstraits. Listes.

Durée : 60mn
Sans documents

Exercice 1.1 *Récursivité*
Donner un exemple d'algorithme récursif présentant une récursivité terminale.

Exercice 1.2 *Listes simplement chaînées*
Dans cet exercice on considère le type abstrait `listeSC` d'objets défini avec l'ensemble des primitives données en *annexe A*

1. Supposons qu'on dispose d'une liste `L` de type `listeSC`. Écrire une fonction `listeClassement` qui pour un objet donné renvoie sa distance à la fin de la liste. On supposera qu'il n'y a pas d'occurrences multiples.
   Par exemple, si la liste `L` est composée par la séquence d'objets `<3,4,6,1,0,5>`, pour l'objet `6`, la fonction `listeClassement` calcule `3`.
2. Supposons qu'on dispose de deux listes `L1` et `L2` du type `listeSC` triées dans l'ordre décroissant. Ecrire une fonction `listeFusion` qui construit une liste `L` constituée de tous les éléments de `L1` et de `L2` dans l'ordre croissant.
   Par exemple, pour `L1= <4, 3, 2, 2, 1>` et `L2= < 5, 4, 3, 1 >`. Le résultat de `listeFusion` sera `L= < 1, 1, 2, 2, 3, 3, 4, 4, 5>`

Exercice 1.3 *Listes doublement chaînées*
Dans cet exercice on considère le type abstrait `listeDC` défini avec l'ensemble des primitives données en *annexe B*

1. Proposer une implémentation de `listeDC` par un tableau.
2. Écrire la primitive `insererEnTete`

Exercice 1.4 *Jeu de Josephus*
Dans ce problème on considère `N` personnes qui forment un cercle et un entier `M`.
Le jeu consiste à éliminer de manière itérative la `M`-ième personne du cercle jusqu'à ce qu'il ne reste personne.
Par exemple, pour `N=9` et `M=5`, l'ordre d'élimination est `5,1,7,4,3,6,9,2,8`.
Proposer une structure de données pour la sauvegarde de l'ensemble de personnes et écrire une fonction qui affiche cet ensemble dans l'ordre d'élimination décrit.

Annexe A **Liste simplement chaînée listeSC**

```
listeSC= liste d'objets;
fonction valeur(val L:listeSC) : objet;
fonction debutListe(ref L:listeSC) :vide;
fonction suivant(ref L:listeSC) : vide;
fonction listeVide(val L:listeSC) : booleen;
fonction getCleListe(val L: listeSC) : curseur;
fonction creerListe(ref L:listeSC) : vide;
fonction insererApres(ref L:listeSC, val x:objet;) : vide;
fonction insererEnTete(ref L:listeSC, val x:objet) : vide;
fonction supprimerApres(ref L:listeSC) : vide;
fonction supprimerEnTete(ref L:listeSC) : vide;
fonction setCleListe(ref L: listeSC, val c:curseur) : vide;
fonction detruireListe(ref L:listeSC) : vide;
```

Annexe B **Liste doublement chaînée listeDC**

```
listeDC= liste d'objets;
fonction valeur(val L:listeDC) : objet;
fonction debutListe(ref L:listeDC) : vide;
fonction finListe(ref L:listeDC): vide;
fonction suivant(ref L:listeDC) : vide;
fonction precedent(ref L:listeDC) : vide;
fonction listeVide(val L:listeDC) : booleen;
fonction getCleListe(val L:listeDC) : curseur;
fonction creerListe(ref L:listeDC) : vide;
fonction insererApres(ref L:listeDC, val L:objet;) : vide;
fonction insererEnTete(ref L:listeDC, val X:objet) : vide;
fonction supprimerApres(ref L:listeDC) : vide;
fonction supprimerEnTete(ref L:listeDC) : vide;
fonction detruireListe(ref L:listeDC) : vide;
fonction setCleListe(ref L:listeDC, curseur c) : vide;
```