Université Bordeaux 1 LST Mention Informatique Semestre 3 2012-2013

Algorithmique 1 : Devoir Surveillé 1

Types de données abstraits. Listes.

Durée : 45mn
Sans documents

Exercice 1.1 *Récursivité*
Soit les fonctions `f` et `g`.

```
fonction f(val n: entier, ref cmpt : entier):entier;
debut
  cmpt= cmpt + 1;
  si n < 2 alors
    retourner n;
  sinon
    retourner f(n-1, cmpt) + f(n-2, cmpt);
  fsi
fin

fonction g(val n: entier, ref F: tableau[1..10] d'entiers, ref cmpt:entier):vide;
debut
  cmpt= cmpt + 1;
  si n > 2 alors
    si F[n/2 + 1] == 0 alors
      g(n/2 + 1, F, cmpt);
    fsi
    si F[(n-1)/2] == 0 alors
      g((n-1)/2, F, cmpt);
    fsi
    F[n]= F[n/2+1]*F[n/2+1] - sign(n) *  F[(n-1)/2]*F[(n-1)/2];
  fsi
fin
```

La fonction `sign(var n:entier):entier` renvoie `1` si `n` est pair et `-1` sinon.

1. On fait appel à la fonction `f` avec `n=3` et `cmpt=0`. Quelle est la valeur retournée par cet appel ? Quelles sont les valeurs de `n` et de `cmpt` après l'exécution de cet appel ?
2. On fait appel à la fonction `g` avec `n=3`, `F={0,0,0,0,0,0,0,0,0,0}` et `cmpt=0`. Quelles sont les valeurs de `n`, de `F` et de `cmpt` après l'exécution de cet appel ?
3. Quelle est la complexité de `f` et `g` en fonction de `n` ?

Exercice 1.2 *Listes chaînées (Extrait de sujet d'examen, Déc. 2007)*
On considère un tableau de N entiers tous différents et appartenant à l'intervalle `[1..N]`. Par exemple, pour $N = 8$, un tableau vérifant ces contraintes est `T={5,8,2,6,3,4,1,7}`.

1. Ecrire une fonction `tableauToListe` qui transforme un tableau en une liste simplement chaînée en conservant l'ordre des entiers dans le tableau et renvoie `vrai` si les éléments du tableau sont conformes à l'énoncé (des entiers tous différents) et `faux` sinon.
2. Ecrire une fonction `picDeListeSC` qui à partir de la liste ainsi constituée fournit la liste des valeurs $T[j]$ telles que $T[j-1] < T[j]$ et $T[j+1] < T[j]$. Sur l'exemple, `4,6,8`.
3. Si on choisit d'utiliser des listes doublement chaînées, écrire la fonction `picDeListeDC`
4. Donner les avantages et les inconvénients des deux implémentations.

Exercice 1.3 *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;
```