Université Bordeaux 1 LST Mention Informatique Semestre 3 2012-2013

Algorithmique 1 : Devoir Surveillé 2

Piles, Files, Arbres

Durée : 60mn
Sans documents

Exercice 2.1 *Pile et file*

Un palindrome est une chaîne de caractères qui se lit de la même manière de gauche à droite et de droite à gauche. En utilisant un nombre fixe de piles et de files, des primitives du type `pile de objet` et `file de objet` , et un nombre fixe de variables de type `entier` et `car`, écrire un algorithme qui détermine si une chaîne de caractères est un palindrome.
On suppose que la chaîne de caractères est passée en paramètre dans une liste simplement chaînée qui ne sera parcourue qu'une seule fois.
L'algorithme doit donner en sortie vrai ou faux selon le cas.

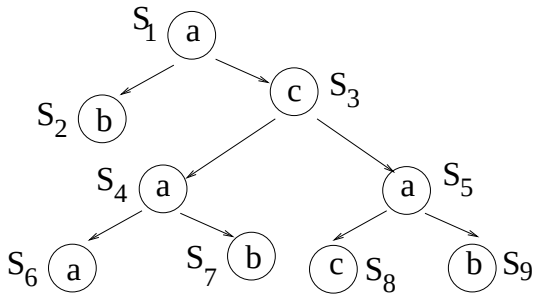

FIG. 1 – A-arbre

Exercice 2.2 *Arbres Binaires*

Soit un arbre binaire dont les sommets contiennent un élément de $A = \{'a', 'b', 'c'\}$. Dans la suite on appellera ce type d'arbre A-arbre. La figure 1 donne un exemple d'un A-arbre.

1. Ecrire une fonction `verifier` qui teste si un arbre binaire est un A-arbre.
2. On dit qu'un sommet `s` est sympathique s'il a deux fils et si la valeur `getValeur(filsGauche(s))` précède la valeur `getValeur(filsDroit(s))` dans l'ordre lexicographique.
   Sur l'exemple donné le sommet S_1 est sympathique et le sommet S_3 ne l'est pas.
   Ecrire une fonction `nbSympa` qui prend pour argument un A-arbre et fournit comme résultat le nombre de noeuds sympathiques de l'arbre.

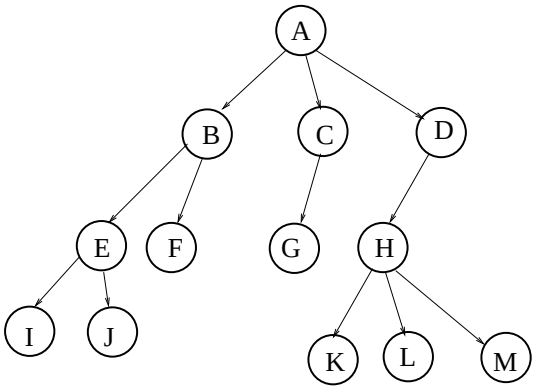

FIG. 2 – Arbre planaire

Exercice 2.3 *Arbres Planaires*

Soit l'arbre planaire A_{pl} illustré sur la figure 2.

1. Donner la suite des sommets lors d'un parcours postfixe de A_{pl}.
2. Dessiner la représentation de A_{pl} dans le type `arbreBinaire`.
3. Écrire une fonction itérative de parcours préfixe d'un arbre de type `sommetArbrePlanaire`.

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 **Type abstrait** *pile de objet*

```
fonction valeur(val P:pile de objet):objet;
fonction pileVide(val P:pile de objet):booleen;
```

```
fonction creerPile(ref P:pile de objet): vide;
fonction empiler(ref P:pile de objet, val x:objet):vide;
fonction depiler (ref P:pile de objet):vide;
fonction detruirePile(ref P:pile de objet):vide;
```

Annexe C **Type abstrait** *file de objet*

```
fonction valeur(val F:file de objet):objet;
fonction fileVide(val F:file de objet):booleen;
fonction creerFile(ref F:file de objet): vide;
fonction enfiler(ref F:file de objet, val v:objet):vide;
fonction defiler(ref F:file de objet):vide;
fonction detruireFile(ref F:file de objet):vide;
```

Annexe D **Type abstrait** *arbreBinaire*

```
arbreBinaire= curseur;
sommet= curseur;
fonction creerArbreBinaire(val Racine:objet):sommet;
fonction detruireArbreBinaire(ref S:sommet):vide;
fonction getValeur(val S:sommet):objet;
fonction filsGauche(val S:sommet):sommet;
fonction filsDroit(val S:sommet):sommet;
fonction pere(val S:sommet):sommet;
fonction setValeur(ref S:sommet, val x:objet):vide;
fonction ajouterFilsGauche(ref S:sommet, val x:objet):vide;
fonction ajouterFilsDroit(ref S:sommet, x:objet):vide;
fonction supprimerFilsGauche(ref S:sommet):vide;
fonction supprimerFilsDroit(ref S:sommet):vide;
fonction detruireSommet(ref S:sommet):vide;
```

Annexe E **Implémentation du type abstrait** *arbreBinaire*

```
cellule=structure
         info:objet;
         gauche: sommet;
         droit: sommet;
         pere: sommet;
       finstructure
sommet= ^cellule;
arbreBinaire= sommet;
```

Annexe F **Type abstrait** *sommetArbrePlanaire*

```
sommetArbrePlanaire= curseur;
fonction creerArbrePlanaire(val Racine:objet):sommetArbrePlanaire;
fonction detruireArbrePlanaire(ref S:sommetArbrePlanaire):vide;
fonction getValeur(val S:sommetArbreBinaire):objet;
```

```
fonction premierFils(val S:sommetArbreBinaire):sommetArbreBinaire;
fonction frere(val S:sommetArbreBinaire):sommetArbreBinaire;
fonction pere(val S:sommetArbreBinaire):sommetArbreBinaire;
fonction setValeur(ref S:sommetArbrePlanaire, val x:objet):vide;
fonction ajouterFils(ref S:sommetArbrePlanaire, val x:objet):vide;
fonction supprimerSommet(ref S:sommetArbrePlanaire):vide;
```

Annexe G **Implémentation du type abstrait par fils gauche-frère droit** *sommetArbrePlanaire*

```
cellule= structure
          info: objet;
          premierFils: sommet
          frere: sommet;
          pere: sommet
         finstructure
sommet= ^cellule;
sommetArbrePlanaire= sommet;
```