Université
Caen Normandie

Année 2018-2019

Portail Mathématiques, Informatique
Licence deuxième année (L2 Info, L2 maths, L2 MIASHS)

Algorithmique et structures de données
TD 4

Exercice 1. Texte bien parenthésé

Une **pile** est une structure de données mettant en œuvre le principe du *dernier entré, premier sorti* (LIFO pour Last-in, First-Out). On supposera ici que les éléments contenus dans la pile sont des caractères. En général, elle se manipule avec les procédures suivantes :

- **initPile(p: pile): pile** retourne une pile vide.
- **pileVide(p: pile): booléen** teste si la pile p est vide ou non
- **sommet(p: pile): caractère** retourne le sommet de la pile (retourne un message d'erreur si la pile est vide).
- **empiler(p: pile, x: caractère)** ajoute le caractère x au sommet de la pile p.
- **depiler(p: pile)** retire de la pile p son sommet.

On souhaite utiliser une pile pour déterminer si un texte est bien parenthésé. On considérera deux types de parenthèses, les parenthèses normales '(', ')', appelées respectivement parenthèses ouvrantes et fermantes et les accolades '{','}', appelées respectivement accolades ouvrantes et fermantes.

On vérifie que le texte est bien parenthésé en parcourant le texte caractère par caractère et en utilisant une pile.

Au départ la pile est vide et ensuite elle ne contiendra ques des parenthèses ou accolades ouvrantes. On procède de la façon suivante

— si on lit une parenthèse ou une accolade fermante, on empile ce caractère
— si on lit une parenthèse ou une accolade fermante, nous avons trois possibilités

1. la pile est vide le texte est alors mal parenthèsé
2. le sommet de la pile contient le mauvais type de parenthèse, le texte est également mal parenthésé
3. le sommet de la pile contient le même type de parenthèse et on dépile

— lorsque le texte est complètement lu, le texte est bien parenthésé lorsque la pile est vide.

Écrivez une procédure **bienParenthese(texte :chaîne de caracteres):booléen** qui retourne Vrai lorsque le texte est bien parenthésé.

Exercice 2. **Dérécursiver une procédure récursive à l'aide d'une pile**
Rappelez la structure de liste simplement chaînée.

On considère la procédure récursive suivante :

Procédure 1 *affichageInverseRec*(L: liste)

```
si L <> None alors
    affichageInverseRec(L→suivant)
    afficher L →valeur
```

Question 1. Exécutez pas à pas **affichageInverseRec** sur une liste à 3 nœuds. Que fait cette procédure ?

On souhaite écrire une version itérative de **affichageInverseRec**. La difficulté provient du fait que dans cette procédure la récursivité est non terminale. En effet, lors de l'exécution de **affichageInverseRec**, il faut revenir après l'appel récursif à **affichageInverseRec(L→suivant)** dans la procédure pour effectuer l'instruction suivante. Concrétement, une pile d'appels implicite est constituée lors de l'exécution. Elle mémorise, lors de l'appel récursif, le contexte nécessaire à la reprise de la procédure au retour de cet appel.

Question 2. Dessinez le contenu de la pile sur l'exemple précédent.

Question 3. Donnez une version itérative de **affichageInverseRec** en utilisant une pile d'entiers.

Exercice 3. **Utilisation d'une liste chaînée circulaire pour représenter une file**

Une **file** est une structure de données mettant en œuvre le principe du *premier entré, premier sorti* (FIFO pour First-in, First-Out). On supposera ici que les éléments contenu dans la file sont tous des entiers.

Question 1. Rappelez les cinq procédures permettant de manipuler une file.

Question 2. Donnez une représentation d'une file à partir d'une liste chaînée circulaire. Pourquoi faut-il pointer sur le dernier nœud ?

Question 3. En utilisant cette représentation, définissez les cinq procédures.