TD n°3

Exercice 1 (évaluation d'expressions)

On s'intéresse à l'évaluation des expressions arithmétiques écrites en notation post-fixe. Dans cette notation, les opérateurs sont placés à la suite de leurs opérandes, par exemple :

infixe	post-fixe
$3 + 2$	$3\ 2\ +$
$3 \times 5 + 2$	$3\ 5 \times 2\ +$
$3 \times (5 + 2)$	$3\ 5\ 2 + \times$
$3 + 5 \times 2$	$3\ 5\ 2 \times +$
$(3 + 5) \times 2$	$3\ 5 + 2\ \times$
$(3 + 5) \times 2 + (4 + 1) \times 6$	$3\ 5 + 2 \times 4\ 1 + 6 \times +$

L'avantage de cette notation est d'être non ambigüe. Elle évite l'écriture des parenthèses et la prise en compte des priorités des opérateurs. Dans cet exercice, on veut écrire un algorithme qui évalue une expression écrite en notation post-fixe et représentée par une liste de chaînes de caractères dans laquelle chaque élément est un entier ou une opération. Par exemple, l'expression $3\ 5 + 2\ \times$ introduite ci-dessus est donnée par la liste $\langle$`"3", "5", "+", "2", "*"`$\rangle$. On suppose l'existence des deux fonctions suivantes : « booléen est_entier(Chaine ch) » teste si une chaîne représente un entier ; « entier chaine_vers_entier(Chaine ch) » transforme une chaîne de caractères en entier.

1. Écrire un algorithme en considérant des expressions contenant seulement des entiers, des additions et des multiplications. Réfléchir à l'utilisation d'une pile.
2. Montrer comment étendre cet algorithme aux expressions plus générales contenant d'autres opérations (sinus, cosinus, etc.) et des nombres réels.

Exercice 2 (vérification du parenthésage)

Les langages informatiques utilisent en général des couples de symboles pour délimiter des parties de programmes, par exemple des parenthèses `( )`, des crochets `[ ]`, des accolades `{ }`, des balises `<html> </html>` ou `<div> </div>`, des instructions `pour finpour` ou `si finsi`, etc. Ces couples de symboles, l'un ouvrant et l'autre fermant, apparaissent soit en séquence, soit de manière imbriquée, sachant qu'un symbole fermant doit toujours être apparié avec le symbole ouvrant correspondant.

On considère ici une liste de chaînes de caractères extraites d'un code source écrit dans un langage quelconque et représentant (seulement) les symboles ouvrants et fermants dans l'ordre où ils apparaissent dans le programme. Par exemple, les listes suivantes sont valides :

```
[ ( ) ]   ||   [ ] ( )   ||   { [ ] ( ) }
```

Les listes suivantes sont invalides :

```
[ ( ) )   ||   [ ( ] )   ||   [ ) ( ]
[ ] )     ||   [ ( )     ||   ] [ ( )
```

Écrire un algorithme de test pour savoir si les symboles donnés dans une liste telle que décrite précédemment sont bien appariés. Cet algorithme pourrait utiliser une pile. Les fonctions suivantes sont supposées connues : « booléen est_ouvrant(Chaine ch) » teste si une chaîne est un symbole ouvrant ; « booléen est_fermant(Chaine ch) » teste si une chaîne est un symbole fermant ; « booléen en_couple(Chaine ouv, Chaine fer) » teste si *ouv* est un ouvrant associé au symbole fermant *fer*.

Exercice 3 (files avec capacité)

Dans un aéroport, une piste d'envol fonctionne comme une file d'attente avec une capacité correspondant au nombre maximum d'avions pouvant se trouver en attente. Cette capacité est connue lors de la construction de la piste.

1. Définir la SDA des files avec capacité.
2. Proposer une SDC sous forme de tableau.
3. Proposer une SDC sous forme de chaînage.
4. Maintenant on considère la liste des pistes d'un aéroport et on veut diriger un nouvel avion vers la piste la moins encombrée. Écrire un algorithme pour réaliser cette opération.

Exercice 4 (files avec priorité)

Aux urgences d'un hôpital, la réception des patients fonctionne (de manière simplifiée) comme suit. Chaque patient est associé à un niveau d'urgence (degré de gravité) selon sa pathologie et il est mis en attente. Un patient est pris en charge par un médecin quand il n'y a pas de cas plus urgent en attente et que c'est le premier arrivé parmi les patients de même niveau d'urgence.

1. Définir la SDA des files avec priorité.
2. Proposer une SDC sous forme de liste d'éléments.
3. Proposer une SDC sous forme de liste de files d'éléments.
4. Proposer une SDC spécifique pour les situations où les différents niveaux de priorité sont connus à l'avance au moment de créer la file. Est-il nécessaire de modifier la SDA ?