Travaux Pratiques d'algorithmique n°8

Cours d'Informatique de Deuxième Année

—DEUG Sciences S3-MI—

Arbres et expressions arithmétiques

On cherche dans ce TP à utiliser la structure d'arbre binaire pour coder des expressions arithmétiques. L'idée est de stocker les opérateurs dans les noeuds et les opérandes dans les sous-arbres gauches et droits Les feuilles représentent quant à elles les valeurs entières.

On considère les déclarations suivantes :

```
typedef struct noeud
      {
        int Op;                  /* operateur '+', '-', '*', '/', '%', */
                                 /* ou valeur entiere                  */
        struct noeud * Gauche;   /* operande de gauche                 */
        struct noeud * Droit;    /* operande de droite                 */
      } Noeud;

typedef Noeud * Expression;
```

A noter que l'on utilise le même champ pour désigner un opérateur — *i.e.* '+', '-', '*', '/', ou encore '%' — ou une valeur entière. Étant donné un noeud, on n'aura pas de difficulté à décider si `Op` est un opérateur ou une valeur entière puisqu'il s'agit d'une valeur entière si et seulement si les deux pointeurs `Gauche` et `Droit` sont égaux à `NULL`.

▸ **Exercice 1.** *Afin de pouvoir manipuler une telle structure, on demande au préalable d'écrire les fonctions suivantes :*

- *`CreerFeuille(int val)` qui retournera un pointeur sur un noeud contenant la* **valeur** `val` *;*
- *`CreerNoeud(int op, Expression gauche, Expression droite)` qui retournera un pointeur sur un noeud dont les champs `Op`, `Gauche` et `Droit` seront respectivement initialisés à `op`, `gauche` et `droit` ;*
- *`EstFeuille(Noeud * expr)` qui testera si le noeud pointé par `expr` est une feuille.*

▸ **Exercice 2.** *Construisez à la main l'arbre correspondant à l'expression préfixée* `+ - 4 * 3 + 1 7 - 4 3`. *Quel est le résultat de cette expression ? Écrire la fonction* `Expression AnalyseExpr(void)` *qui analyse une expression (valide) sous forme préfixe rentrée par l'utilisateur, et construit l'arbre correspondant.*

▸ **Exercice 3.** *Maintenant que l'on sait construire l'arbre correspondant à une expression arithmétique lue en ordre préfixe, on désire afficher l'arbre (en fait l'expression) en ordre infixe. Donner une fonction* `void AffInfixe(Expression expr)` *affichant l'expression* `expr` *en infixe, en parenthésant au maximum.*

▸ **Exercice 4.** *Donner de même les fonctions de noms* `AffPrefixe` *et* `AffSuffixe` *prenant en argument une expression* `expr` *de type* `Expression`, *affichant cette expression en mode préfixe et suffixe. De quelle autre façon appelle-t-on la notation suffixe ?*

▸ **Exercice 5.** *Il est important de libérer l'espace mémoire qui n'est plus utilisé. Donner les fonctions suivantes :*

- `EffacerNoeud(Noeud * n)` *qui libèrera la mémoire occupée par un noeud ;*
- `EffacerExpression(Expression expr)` *qui libèrera la mémoire occupée par une expression.*

▸ **Exercice 6.** *Donner une fonction de nom* `EvalExpr` *prenant en argument une expression* `expr` *de type* `Expression` *retournant la valeur entière de l'expression* `expr`.

▸ **Exercice 7.** *Faire une fonction appelée* `AffInfixeMini` *qui minimisera le nombre de parenthèses utilisées dans l'affichage infixe d'une expression* `expr` *de type* `Expression`.