Travaux Dirigés sur le langage *C* n°.8

Cours d'Informatique de Deuxième Année

—Deug SSM M3—

On cherche dans ce TD 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 de gauche et de droite de ces opérateurs respectivement dans les sous-arbres gauches et les sous-arbres droits. Les feuilles représentent quant à elles les valeurs entières.

Arbres et expressions arithmétiques. . .

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;  /* pointeur sur un noeud               */
```

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 puisque si les deux pointeurs `Gauche` et `Droit` sont à `NULL`, il s'agira d'une valeur entière.

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

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

▸ **Exercice 2**. On cherche dans cet exercice à construire l'arbre représentant une expression arithmétique donnée sous sa forme **préfixe**. Faire tourner cet algorithme de construction sur l'expression `+ - 4 * 3 + 1 7 - 4 3`.

▸ **Exercice 3**. Déduire de l'exercice précédent la fonction de nom `AnalyseExpr()` d'analyse d'une expression sous forme préfixe rentrée par l'utilisateur et de construction de l'arbre correspondant. La fonction retournera un pointeur sur l'arbre associée à l'expression analysée. On supposera dans cet exercice que les expressions entrées par l'utilisateur sont bien formées.

▸ **Exercice 4**. Maintenant que l'on sait construire l'arbre correspondant à une expression arithmétique lue en ordre **préfixe**, on désire afficher cet arbre en ordre **infixe**. Donner une fonction de nom `AffInfixe()` prenant en argument une expression `expr` de type `Expression` et affichant cette expression en **infixe** en parenthèsant au maximum.

▸ **Exercice 5**. Donner de même, les fonctions de noms `AffPrefixe()` et `AffSuffixe()` prenant en argument une expression `expr` de type `Expression`, respectivement d'affichage de cette expression en **préfixe** et **suffixe**.

▸ **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`. On pourra supposer que l'arbre n'est pas vide (`NULL`) et on justifiera cette hypothèse.

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