Travaux Pratiques
séance nr. 4

Exercice: arbres binaires

Nous définissons un type "expr" pour un arbre binaire dont les nœuds internes sont des caractères et les feuilles des nombres flottants. La structure correspondante est :

```
typedef struct s_noeud {
        union {
                char    op;
                float   val;
                } etiq;
        struct s_noeud *fg, *fd;
} noeud, *expr;
```

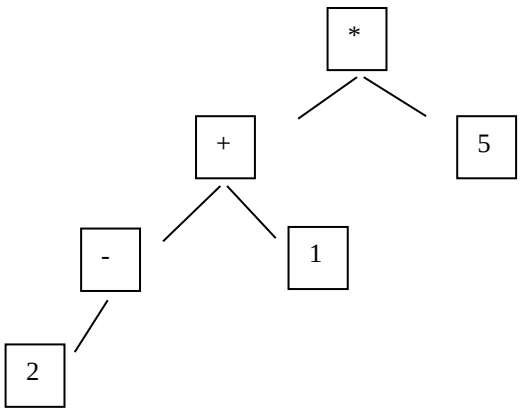
L'union permet de faire varier le type de l'étiquette en fonction du nœud : pour un nœud interne, l'étique est un caractère, pour une feuille c'est un flottant.

Implanter les opérations:

- exprnouv() → expr
- enrac (char x, expr g, expr d) → expr
- creerfeuille(float x) → expr
- agauche, adroit (expr e) → expr
- est_vide, est_feuille, est_interne (expr e) → bool
- racine_interne(expr e) → char
- racine_feuille(expr e) → float

Les deux dernières opérations renvoient la valeur de l'étiquette en fonction de la nature du nœud (interne ou feuille). « enrac » permet de créer un arbre à l'aide de deux sous arbres : l'étiquette est nécessairement un caractère. « créer_feuille » permet de créer un arbre réduit à une seule feuille : son étiquette est nécessairement un flottant.

Ecrire le code permettant de créer l'arbre suivant:



L'arbre précédent correspond à une expression arithmétique en interprétant le caractère comme un opérateur. Dans l'exemple, il s'agit de l'expression : (-2 + 1) * 5

Ecrire une fonction qui permet d'afficher l'expression avec le parenthésage correct :

```
void        affiche(expr e)
```

Note : mettre systématiquement des parenthèses, même si, a priori, les règles de priorité permettraient de les éviter.

Écrire une fonction qui calcule la valeur de l'expression :

```
float       eval(expr e)
```

en se limitant aux caractères qui représentent les quatre opérateurs suivants : + – * /

Attention : l'opérateur – peut être un opérateur unaire (il permet de prendre l'opposé d'un nombre). Pour l'exemple précédent, la fonction doit renvoyer : -5.0

Ecrire une fonction qui vérifie que l'expression est valide.

Par exemple les opérateurs binaires +, * et / ont nécessairement deux fils non nuls. Par ailleurs, il n'est pas possible de diviser par 0.

```
bool        est_valide(expr e)
```