Table des matières

Introduction

Afin d'évaluer une expression mathématique et obtenir son résultat, nous allons faire appel à ce que nous avons appris à propos de la gestion des arbres binaires.

1. Structures de données

Pour stocker des opérateurs et des nombres dans l'arbre nous devront faire appel à une uninon :

```
typedef union
{
   float nombre;
   char operateur;
} ValeurNoeud;
```

Mais pour savoir le type de données stocké, nous avons défini un type **enum**.

```
typedef enum {FLOAT, CHAR} TypeNeoud;
```

Ansi nous obtenons la structure suivante :

```
typedef struct
{
   ValeurNoeud valeur;
```

```
   TypeNeoud type;
}Noeud;
```

et end la structure de l'arbre contiendera une variable de type **Noeud** pour stocker la valeur, et des pointeurs vers les fils gauche et droit :

```
typedef struct Arbre
{
 Noeud valeur;
 struct Arbre *gauche, *droit;
}Arbre;
```

2. Lecture de l'expression

Pour lire l'expression mathématique entrée comme chaine de caractères, nous avons creé la fonction suivante :

```
Noeud Suivant_expression(char *expr, int* pos)
{
 // Type d'element qui va etre lu
 Noeud noeud;
 static TypeNeoud typeCourant = FLOAT;
 noeud.type = typeCourant;

 switch (typeCourant) {
  // Operateur
  case CHAR :
   if(!estOperateur(expr[*pos]))
    {
     printf("Erreur : expression invalide\n");
     exit(1);
    }
   noeud.valeur.operateur = expr[(*pos)++];
   break;
   // Nombre
```

```
   case FLOAT :
    {
     char *pEnd; // L'indice de la fin du nombre lu
     noeud.valeur.nombre = strtof(expr + *pos, &pEnd);
     *pos = pEnd - expr;
     break;
    }
   }

  // Changement du type
  if(expr[*pos] == '\0') typeCourant = FLOAT;
  else typeCourant = (typeCourant == FLOAT)? CHAR  : FLOAT;

  return noeud;
}
```

Cette fonction prend comme arguments l'expression, et la position ou la lecture est arrivé. Elle utilise en interne une variable **static** nommée typeCourant pour repéré le type d'élément qui doit être lu durant l'appel courant de la fonction.

Si le type courant est un opérateur il suffit de retourne la case à l'indice pos et d'incrémenter ce dernier.

Dans le cas de la lecture d'un nombre en fait appel à la fonction **strtof** . Cette dernière prend deux arguments la chaine de caractère à lire et un pointeur vers une chaine de caractère qui va contenir la position ou s'est arrêté la lecture.

3. Construction de l'arbre

Le code suivant montre la fonction responsable de la construction de l'arbre :

```
Arbre* Construire_arbre(char *expr)
```

```
{
 int pos = 0; // position
 Noeud noeud; // element actuel
 Arbre *courant = NULL, *racine = NULL;

 // Premier nombre
 noeud = Suivant_expression(expr, &pos);
 racine = Malloc_arbre();
 racine->valeur = noeud;

 // Premier operateur
 if(expr[pos] != '\0')
  {
   noeud = Suivant_expression(expr, &pos);
   Arbre *premier_nombre = racine;
   courant = racine = Malloc_arbre();
   racine->valeur = noeud;
   racine->gauche = premier_nombre;
  }

 while(expr[pos] != '\0')
  {
   // Nombre
   noeud = Suivant_expression(expr, &pos);
   Arbre *nombre = Malloc_arbre();
   nombre->valeur = noeud;

   // Operateur
   if(expr[pos] != '\0')
    {
     // L'operateur lu devient courant
     Arbre *ancien_courant = courant;
     noeud = Suivant_expression(expr, &pos);
     courant = Malloc_arbre();
     courant->valeur = noeud;

     if(noeud.valeur.operateur == '-' ||
       noeud.valeur.operateur == '+' )
      {
       // courant devient la nouvelle racine
       Arbre *ancien_racine = racine;
```

```
          racine = courant;
          racine->gauche = ancien_racine;

          // Mettre le nombre comme fils de ancien_courant
          if(!ancien_courant->gauche) ancien_courant->gauche = nombre;
          else ancien_courant->droit = nombre;
         }
        else // L'opérateur est '*' ou '/'
         {
          // Courant devient fils de ancien_courant
          if(!ancien_courant->gauche) ancien_courant->gauche = courant;
          else ancien_courant->droit = courant;

          // Le nombre devient fils de courant
          courant->gauche = nombre;
         }
       }
     else // Pas d'opérateur après le nombre lu
       if(!courant->gauche) courant->gauche = nombre;
       else courant->droit = nombre;
   }

 return racine;
}
```

Cette fonction prend le premier nombre et ensuite l'opérateur qui le suit et met l'opérateur comme racine et le nombre comme son fils gauche.

Après on récupère à chaque fois le nombre et l'opérateur suivants. Si l'opérateur et un + ou un - , le nombre devient fils de l'opérateur courant – qui est le dernier opérateur lu – et l'opérateur devient la nouvelle racine de l'arbre. Sinon si l'opérateur est un * ou un / il devient fils de l'opérateur

courant et le nombre devient son fils.

4. Evaluation de l'expression

La dernière étape de ce processus et d'évaluer l'expression à partir de l'arbre. Cela peut ce faire en utilisant la récursivité avec "l'arbre n'ayant ni de fils gauche ni de fils droit" comme condition d'arrêt. Dans ce cas on retourne la valeur stocké dans l'arbre.

```
float Calcule(float a, char operateur, float b)
{
 switch(operateur)
   {
   case '-' : return a - b;
   case '+' : return a + b;
   case '/' : return a / b;
   default : return a * b;
   }
}

float Calculer_expression(Arbre *arbre)
{
 if(arbre->valeur.type == FLOAT )
   return arbre->valeur.valeur.nombre;
 else
   {
    return Calcule(Calculer_expression(arbre->gauche),
                arbre->valeur.valeur.operateur,
                Calculer_expression(arbre->droit));
   }
}
```