TP C++

Valentin Roussellet

18 octobre 2016

Introduction

Le but de ce TP est de développer un évaluateur d'expression mathématiques simples. Votre programme devra inclure une classe dont le but est de stocke, d'analyser et d'évaluer une expression arithmétique comprenant des entiers et les opérateurs `+`, `-`, `*` et `/`, avec les priorités usuelles.

L'interface publique de votre classe devrait ressembler à ceci :

```
1 class Expr {
2  public:
3   // Construit une expression a partir du texte
4   Expr ( const char * str );
5   // Retourne la valeur de l'expression
6   int eval();
7   // Affiche la représentation interne
8   void print();
9 };
```

Le programme principal appelant la clase `Expr` peut par exemple être

```
1 Expr x("123 / 4 + 123 * 4 - 3");
2 cout << "x = " << x.eval() << "\n";
3 x.print();
```

1 Analyse de l'expression

La première partie consiste a séparer les différents éléments (*tokens*) et à les organiser dans l'ordre de leur évaluation. Dans un premier temps on suppose que chaque élément est séparé par une espace.

Découpage

On peut créer une fonction `split()` avec le prototype suivant :

```
1 std::vector<std::string> split ( const std::string& s, char delim );
```

Cette fonction devra séparer la chaîne de caractères passée en entrée selon le délimiteur spécifié.

Indices : `std::stringstream` et `std::getline`

Priorités

Ensuite il va falloir organiser les expressions dans l'ordre de l'évaluation (*parsing*) avec les règles suivantes :

— La multiplication et la division ont priorité sur l'addition et la soustraction
— Si deux opérations ont une priorité équivalente, on l'évalue de droite à gauche

Exemples :
- `3 + 2 * 4` s'évalue en $3 + (2 \times 4) = 11$
- `17 - 24 / 4 * 3 + 2` s'évalue en $(17 - (24/(4 \times 3)) + 2 = 17$

Une représentation commode de ce genre d'expression est la notation **polonaise inversée** ou notation postfixe(en anglais *reverse polish notation* ou RPN).

Une opération binaire simple comme `a + b` se note `a b +`. Les opérations s'enchainent : par exemple `c / (a + b)` se noterait `c a b + /` tandis que `(a+b)/c` se note `a b + c /` .

Exemple : `17 - 24 / 4 * 3 + 2` devient `17 24 4 3 * / 2 + -` .

On veut stocker dans l'ordre de la notation polonaise inversée une suite de structures représentant soit un nombre, soit un opérateur.
- Définissez la structure `ExprToken` pouvant contenir au choix un nombre ou un opérateur.
- Choisissez un conteneur approprié dans `std::` pour votre liste d'éléments.
- Créez une fonction qui prend en entrée votre liste de tokens et renvoie le conteneur rempli dans le bon ordre.
- N'oubliez pas de détecter les expressions invalides et de renvoyer une erreur le cas échéant.

À ce niveau vous devriez pouvoir implémenter la fonction `print()` pour débuger votre représentation.

Indices : La conversion de notation infixe en RPN se fait en parcourant dans l'ordre la liste des tokens avec une stack pour empiler les opérateurs rencontrés.
- Si l'élément courant est un littéral, le mettre dans la sortie
- Si l'élement courant est un opérateur alors
  - On POP les éléments de la pile dans la sortie jusqu'à rencontrer un opérateur de priorité inférieure
  - On PUSH l'opérateur courant sur la pile.

À la fin de l'algorithme on vide la pile dans la sortie.

2 Évaluation

L'évaluation de votre liste en RPN se fait facilement également avec une stack en parcourant l'expression de gauche à droite :
- Si l'élément courant est un littéral, on le PUSH sur la stack ;
- Si l'élement courant est un opérateur
  - On POP deux éléments de la stack
  - On calcule le résultat de l'opération
  - On PUSH le résultat

À la fin il doit rester un élément dans la stack : c'est le résultat de notre calcul.

3 Pour aller plus loin

- Ajouter la gestion des parenthèses
- Gérer le cas où les éléments de l'expression ne sont pas séparés par des espaces (par exemple `4+3*1234/56`).
- Stocker l'expression sous forme d'un arbre
- Définir une hiérarchie de classe pour les `Token`s en utilisant le polymorphisme pour la fonction d'évaluation.