TP 3 – Calculatrice améliorée

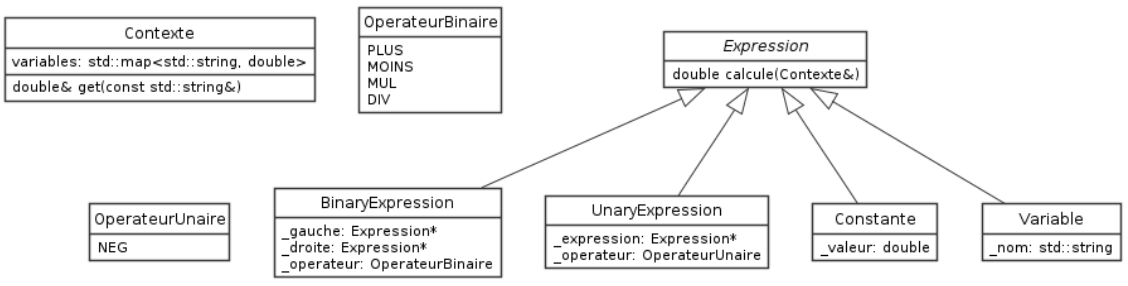
Le but de ce TP est d'améliorer la calculatrice élaborée lors du précédent TP en ajoutant des fonctionnalités permettant d'effectuer des opérations à l'aide de nombres réels, de variables et de conditionnelles.

Structure des arbres de la syntaxe abstraite (AST : Abstract Syntax Tree)

Afin d'implémenter un compilateur pour un langage complexe (et non plus une simple analyse syntaxique) il est courant d'utiliser une structure de données composée d'un AST. Cet arbre va permettre d'ordonner les instructions d'un programme afin de les exécuter par blocs d'instructions. Un exemple avec un programme simple issu de wikipedia :

```
while (b != 0)
  if (a > b)
    a = a - b;
  else
    b = b - a;
return a
```

Nous vous proposons sur moodle des sources implémentant une base pour la structure de l'AST. Le diagramme UML de la structure de données est le suivant :



Télécharger l'archive `bison_expression` sur moodle. Cette archive contient un début de structure pour gérer les expressions. Dans cette structure chaque instruction est une expression, et chaque expression est soit un élément terminal (comme une constante ou une variable par exemple) ou possède une ou plusieurs branches comprenant une expression. Chaque expression hérite d'une méthode calcule qui permettra d'exécuter du code comme par exemple l'application de l'addition pour l'opérateur +. Un exemple d'utilisation de cette structure est présentée dans le fichier `exemple_expression/main.cc`.

Calculatrice V2

Modifier la calculatrice afin que l'utilisateur puisse :

- Utiliser les opérations +, -, / , et × à l'aide d'expressions.
  - Attention au fait que la division par zéro doit renvoyer une erreur
- Utiliser des nombres réels.
- Attention à l'opérateur `moins unaire` qui doit être prioritaire par rapport à tous les autres opérateurs ($-2+3=1$, et non pas $-2+3=-(2+3)=-5$, le même ordre d'opération s'applique aussi pour -2×3). (attention le dernier opérateur déclaré est le plus prioritaire). Ensuite l'opérateur `moins unaire` peut utiliser la priorité de cet opérateur virtuel (pour cela, on peut ajouter "`%prec MOINSU`" juste avant l'action du `moins unaire`).

Pour ajouter ces fonctionnalités vous devez modifier les fichiers `parser.yy` et `scanner.ll`. Vous pouvez vous aider de l'exemple présenté dans le fichier `exemple_expression/main.cc` afin de modifier le symbole non-terminal expression et ses règles. Il sera nécessaire d'ajouter un Contexte à votre programme pour l'exécuter. Les opérations doivent être traitées comme des expressions (de la même manière que présenter dans l'introduction).

Calculatrice V3

Modifier votre calculatrice afin que l'utilisateur puisse :

- Introduire des variables et ajouter des affectations comme : $x=7$
- Utiliser les variables introduites dans les expressions, par exemple $x+x$ doit renvoyer 14.

On implémentera ici correctement la gestion des variables à l'aide de contextes. L'ensemble de ces fonctionnalités devront être implémentées à l'aide des fichiers `driver.hh` et `driver.cc`. Il faudra alors modifier les fonctions en commentaires. L'utilisation de l'objet `driver` se fera ensuite dans le fichier `parser.yy`.).

Calculatrice V4

Nous souhaitons enfin ajouter la possibilité de gérer des opérateurs ternaires tels que des conditionnelles. Ajouter :

- la gestion des expressions ternaires (en créant une classe héritant de Expression)
- la gestion de conditionnelles if/then/else (avec la syntaxe que vous voulez)
- les opérateurs opérateurs de comparaison (<, >, <=, >=, =, !=)
- les opérateurs booléens (&&, ||, XOR, !)

Afin de pouvoir gérer les calculs logiques avec votre calculatrice. Tout chiffre supérieur à 0 est égal à true et tout chiffre inférieur ou égal à 0 est égal à false.