Projet – Évaluation d'expressions arithmétiques

Version du 19 février 2018

Alicia Filipiak – Alicia.Filipiak@gmail.com
Franck Quessette – Franck.Quessette@uvsq.fr
Guillaume Scerri – Guillaume.Scerri@uvsq.fr

Le but de ce projet est d'évaluer les expressions arithmétiques bien formées.

1 Formalisation du problème

Les expressions arithmétiques sont définies comme celles du langage `C` avec néanmoins quelques restrictions :

1. il n'y a pas de variables ;
2. les constantes sont uniquement des entiers non signés ;
3. les opérateurs autorisés sont `+`, `-`, `*` et `/` ;
4. les opérateurs sont tous associatifs de droite à gauche ;
5. il n'y a pas de priorités entre les opérateurs ;
6. les parenthèses `(` et `)` permettent de forcer la priotité.

Question 1 : Donner une grammaire reconnaissant le langage dont les mots sont les expressions arithmétiques définies ci-dessus.

Question 2 : Lire une chaîne de caractère contenant une expression arithmétique et la transformer en une liste de tokens.

Un token est soit un opérateur, soit une parenthèse (ouvrante ou fermante), soit un entier. Ainsi l'expression arithmétique `(2*( (3+ 45)-8))+14 /3` devient la liste de tokens suivante :
`(` → `2` → `*` → `(` → `(` → `3` → `+` → `45` → `)` → `-` → `8` → `)` → `)` → `+` → `14` → `/` → `3`

Dans la liste des tokens, les éventuels espaces ont disparu et chaque token doit contenir les trois informations suivantes :

1. le type de token : parenthèse, opérateur, entier ;
2. la valeur du token qui dépend du type ;
3. un accès au token suivant.

Vous devez donner la structure de données :

```
struct token {
...
};
```

permettant de stocker un token.

Vous devez donner la structure de données :

```
typedef liste_token ...;
```

permettant de stocker une liste de tokens.

Vous devez donner la fonction :

```
liste_token string_to_token (char *string);
```

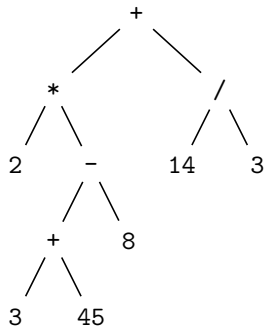
qui transforme une chaine de caractère en une liste de tokens.

Question 3 : À partir de la liste de tokens, vérifier si cette liste est correspond à une expression arithmétique bien formée.

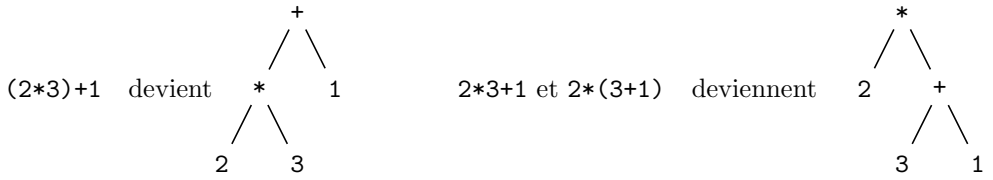
Définir le langage qui est l'ensemble des mots qui sont une liste de tokens correspondant à une expression arithmétique correcte. Vous devez donner l'alphabet et donner l'automate (éventuellement à pile) reconnaissant ce langage.

Question 4 : À partir de la liste de tokens créer l'arbre représentant l'expression arithmétique.

Une expression arithmétique se représente par un arbre. Par exemple l'expression : `(2*((3+45)-8))+14/3` se modélise par :



Noter que les parenthèses ne sont pas présentes dans l'arbre et que l'associativité est contenue dans la structure de l'arbre. Par exemple :



Vous devez donner une structure de données permettant de stoker un arbre contenant des tokens :

```
typedef arbre_token ...;
```

Question 5 : Calculer la valeur de l'expression arithmétique et afficher le résultat

Vous devez donner la fonction :

```
int arbre_to_int (arbre_token at);
```

qui évalue l'arbre de tokens.

Question 6 : Programme

Le programme que vous devez rendre doit prendre en argument la chaîne de caractère et afficher son évaluation ou bien "expression incorrecte". Par exemple :

```
$> ./eval (2*( (3+ 45)-8))+14 /3
84
$>
```

Attention a bien gérer les espaces sur la ligne de commande.

Question facultative 7 : Modifier la grammaire et toute la suite pour tenir compte de la priorité habituelle des opérateurs.

Question facultative 8 : Modifier la grammaire et toute la suite pour tenir compte de l'associativité habituelle des opérateurs.

Question facultative 9 : Modifier la grammaire et toute la suite pour pouvoir utiliser des entiers signés.

Question facultative 10 : Modifier la grammaire et toute la suite pour pouvoir utiliser des flottants signés.

2 Modalités pratiques

Merci de respecter les consignes suivantes :

- le projet est à faire en seul ou en binôme ;
- le projet est à rendre dans l'espace e-campus2 au plus tard le **dimanche 27 mai 2018 à 23h59** ;
- vous devez déposer un fichier `NOM1_Prenom1-NOM2_Prenom2.zip` qui est le zip du dossier `NOM1_Prenom1-NOM2_Prenom2` contenant :
  - un compte-rendu (5 pages max) contenant les réponses aux question théoriques et d'éventuels commentaires sur votre programme. Ce compte-rendu doit être en LaTeX et vous devez fournir le .tex `cr.tex` et le .pdf `cr.pdf` ;
  - votre programme écrit en `C` qui doit s'appeler `eval.c` ;
  - un `Makefile` permettant de compiler et d'exécuter le programme et une cible `test` qui efface l'exécutable, compile et lance le programme sur 5 exemples montrant les fonctionnalités de votre programme. La cible `test` doit être la première du `Makefile`

Le non-respect d'une consigne entraine 2 points de moins sur la note finale. Le retard de la remise du projet entraine 1 point de moins par heure de retard.