

Projet – Évaluation d’expressions booléennes

Version du 19 mars 2019

Franck Quessette – Franck.Quessette@uvsq.fr
Guillaume Scerri – Guillaume.Scerri@uvsq.fr
Yann Strozecki – Yann.Strozecki@uvsq.fr

Le but de ce projet est d’évaluer les expressions booléennes.

1 Formalisation du problème

Les expressions booléennes sont définies de la façon suivante :

1. il n’y a pas de variables ;
2. les constantes sont uniquement `0` (pour faux) et `1` (pour vrai) ;
3. les opérateurs binaires autorisés sont `+` (pour le OU), `.` (pour le ET), `=>` (pour l’implication), `<=>` (pour l’équivalence) ;
4. l’opérateur unaire autorisé est `NON` ;
5. les opérateurs binaires sont tous associatifs de gauche à droite ;
6. il n’y a pas de priorités entre les opérateurs ;
7. les parenthèses `(` et `)` permettent de forcer la priorité.

Question 1 : Donner une grammaire reconnaissant le langage dont les mots sont les expressions booléennes 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 booléenne `(1.(0+1)=>NON1)<=>(1=>0)` devient la liste de tokens suivante :

`(` → `1` → `.` → `(` → `(` → `0` → `+` → `1` → `)` → `=>` → `NON` → `1` → `)` → `)` → `<=>` → `(` → `1` → `=>` → `0` → `)`

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, constante ;
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);
```

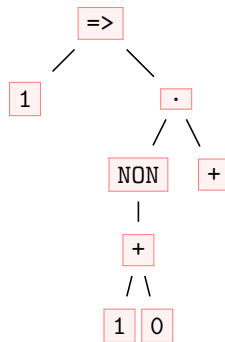
qui transforme une chaîne de caractère en une liste de tokens.

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

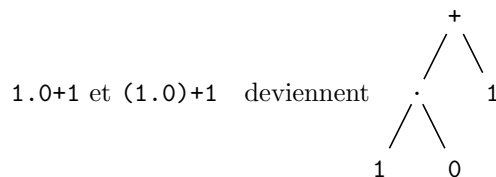
Définir le langage qui est l'ensemble des mots qui sont une liste de tokens correspondant à une expression arithmétique correcte. Vous devez donner l'alphabet et donner l'automate (éventuellement à pile) reconnaissant ce langage.

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

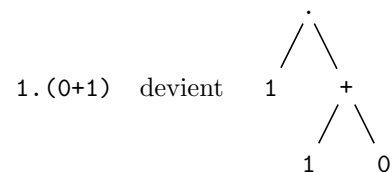
Une expression arithmétique se représente par un arbre. Par exemple l'expression : $(1 \Rightarrow (\text{NON } (1+0) \cdot 1))$ se modélise par :



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 :



1.0+1 et (1.0)+1 deviennent



1.(0+1) devient

Vous devez donner une structure de données permettant de stocker 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 (1=>(NON (1+0).1)
FAUX
$>
```

Attention à 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 des opérateurs (préciser quelles priorités vous prenez).

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 26 mai 2019 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 questions 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 entraîne 2 points de moins sur la note finale. Le retard de la remise du projet entraîne 1 point de moins par heure de retard.