

# Projet – Évaluation d’expressions booléennes

Version du 19 mars 2019

Xavier Badin de Montjoye – [xavier.badin-de-montjoye2@uvsq.fr](mailto:xavier.badin-de-montjoye2@uvsq.fr)  
Pierre Coucheney – [pierre.coucheney@uvsq.fr](mailto:pierre.coucheney@uvsq.fr)  
Franck Quessette – [franck.quessette@uvsq.fr](mailto:franck.quessette@uvsq.fr)  
Yann Strobecki – [yann.strobecki@uvsq.fr](mailto:yann.strobecki@uvsq.fr)  
Sandrine Vial – [sandrine.vial@uvsq.fr](mailto:sandrine.vial@uvsq.fr)

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

## 1 Le langage

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 sont `+` (pour le OU), `.` (pour le ET), `=>` (pour l’implication), `<=>` (pour l’équivalence) ;
4. l’opérateur unaire est `NON` ;
5. les opérateurs binaires sont tous associatifs de gauche à droite, c’est à dire en cas de parenthèses non mises, l’expression sera toujours interprétée comme si elle était parenthésée de gauche à droite. Par exemple `0+1.0` doit être interprété comme `(0+1).0` ;
6. il n’y a pas de priorités entre les opérateurs ;
7. les parenthèses `(` et `)` permettent de forcer la priorité.

## 2 Analyse syntaxique

Au départ une expression booléenne est donnée sous forme d’une chaîne de caractères, par exemple :

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

Pour évaluer cette expression booléenne qui est sous forme de chaîne de caractères, il faut 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 en langage C :

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

permettant de stocker une liste de tokens.

Vous devez donner la fonction en langage C :

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

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

**Question 1 :** Lire une chaîne de caractère contenant une expression arithmétique et la transformer en une liste de tokens.

### 3 Reconnaissance par automate

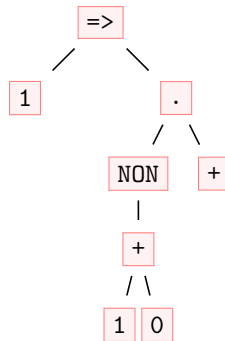
Définir le langage qui est l'ensemble des mots qui sont une liste de tokens correspondant à une expression arithmétique correcte.

**Question 2 :** Donner un automate à pile reconnaissant le langage dont les mots sont les expressions booléennes.

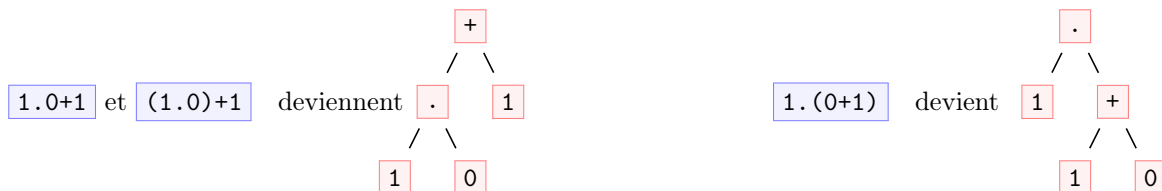
**Question 3 :** Écrire une fonction en langage C qui teste si une liste de token appartient au langage ou non.

### 4 Arbre de l'expression

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



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 stocker un arbre contenant des tokens :

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

**Question 4 :** À partir de la liste de tokens et en utilisant l'automate à pile, créer l'arbre représentant l'expression booléenne. Vous pouvez utiliser la fonction qui teste si la liste des tokens appartient au langage en la modifiant.

## 5 Évaluation

**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.

## 6 Programme final

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 6 :** Modifier l'automate et toute la suite pour tenir compte de la priorité des opérateurs (préciser quelles priorités vous prenez).

## 7 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 moodle au plus tard le **dimanche XX<sup>1</sup> mai 2021 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  $\text{\LaTeX}$  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.

---

1. La date sera précisée ultérieurement.