TP 3 : Utilisation de `bison`

Exercice 1 - Suppression des commentaires avec `flex` et `bison`

Dans cet exercice, on va utiliser `flex` et `bison` pour produire un programme qui supprime les commentaires d'un code `C` fournit sur l'entrée standard. On ne considérera que les commentaires multilignes, compris entre `/*` et `*/` inclus.

Il va falloir faire attention à plusieurs points :

- Le commentaire se termine dès qu'on rencontre la chaîne `*/`. En revanche, le commentaire peut contenir plusieurs fois la chaîne `/*`.
- Il ne peut pas y avoir de commentaires à l'intérieur des littéraux de type `char` (chaînes entre `'` dans le code) et de type `char*` (chaînes entre `"` dans le code).
- Ces littéraux se terminent dès qu'on rencontre à nouveau le caractère `'` ou `"` respectivement, sauf si ce caractère précédent est un `\`.

1.1 Créez un fichier `cleanup_c.y`, dans lequel vous déclarerez les *tokens* représentant les différents éléments de syntaxe en jeu : les caractères, les caractères précédés de `\`, les apostrophes simples et doubles, les débuts et les fins de commentaires.

1.2 Écrivez un fichier `cleanup_c.l`, qui va contenir les expressions régulières et les règles pour générer ces *tokens*.

1.3 Complétez le fichier `cleanup_c.y` en y ajoutant la grammaire et les règles nécessaires pour supprimer uniquement les commentaires.

note : Vous allez avoir besoin de plusieurs symboles non terminaux supplémentaires, en particulier pour gérer les littéraux de type `char`, ceux de type `char*` et les commentaires. Ces éléments de syntaxe sont des suites de `tokens` vérifiant certaines règles, que vous devrez traduire sous forme d'une grammaire récursive à gauche (pourquoi ?).

1.4 Appelez `bison` sur `cleanup_c.y`, puis `flex` sur `cleanup_c.l`, et enfin `gcc` sur les différents fichiers `.c` générés. Une fois l'exécutable produit, testez-le intensivement.

Exercice 2 - Expressions arithmétiques et booléennes

2.1 Reprendre l'exemple de la calculatrice vu en cours, et ajouter le support des expressions booléennes : constantes `true` et `false`, opérateurs logiques `not`, `or` et `and`.

note : Comme `not` n'est pas un opérateur binaire, on utilisera `%nonassoc` (au lieu de `%left` ou `%right`) pour définir sa priorité.

2.2 Ajouter le support des opérateurs de comparaisons (`=`, `<` et `>`), qui doivent être utilisés avec une expression numérique à gauche et une autre à droite.

2.3 Vérifier que l'expression `not (2*3 < 5 and 7 = 3+4)` est valide et s'évalue à `true`.

2.4 Faire en sorte qu'une ligne avec une erreur de syntaxe soit ignorée et n'impacte pas l'évaluation des lignes suivantes.

aide : Il faut utiliser le symbole spécial `error` de `bison`. La grammaire pourra commencer par :

```
S    →  S LINE | ε
LINE →  NUM_EXPR EOL | BOOL_EXPR EOL | error EOL
```