**Master Informatique** **Compilation – J1IN7M13**
**Université Bordeaux 1** **2014 – 2015**

**TD $N^o$1 Ecrire un analyseur lexical en JFlex**

---

1. Vous pouvez choisir de travailler sous Eclipse ou sous Emacs.
   Si vous travaillez sous Emacs, créez un répertoire `TD1` dans lequel vous copierez le fichier `pgcd.c` et le fichier le fichier `build.xml` disponibles sur la page :
   `http://dept-info.labri.fr/~carrere/ENSEIGNEMENT/master.html`
   Créez un sous-répertoire `src` dans lequel vous copierez le fichier `lexer.jflex` ci-dessous. Compiler en tapant (dans le répertoire `TD1`) la commande `ant` qui va exécuter les directives du fichier `build.xml` (équivalent d'un makefile).
   Si vous choisissez de contruire le projet sous Eclipse, lancer *Eclipse* et créer un nouveau projet `Lexer` dans votre espace de travail dans lequel vous copierez le fichier `build.html` et le fichier `pgcd.c`. Créer un paquetage `src` dans lequel vous copierez le fichier `lexer.jflex` suivant :

Listing 1 – lexer.jflex

```
import java.io.*;

%%

%public
%class Lexer
%standalone
%8bit

%{
  StringBuffer str = new StringBuffer();
%}

LineTerminator = \r|\n|\r\n
InputCharacter = [^\n\r]
WhiteSpace = {LineTerminator}|[ \f\t]

%%

/* Keywords */
return   {System.out.println("KEYWORD: " +  yytext());}
break    {System.out.println("KEYWORD: " +  yytext());}

/* Operators */
"+"      {System.out.println("OPERATOR: " +  yytext());}
"-"      {System.out.println("OPERATOR: " +  yytext());}

/* Literals */

/* Comments and whitespace */
{WhiteSpace} {/*  ignore  */}
```

   Exécutez le fichier `build.xml` (Clic droit sur `build.xml`, puis `Run As` et selectionnez `Ant Build`).

2. Augmenter l'analyseur lexical de sorte qu'il puisse reconnaître les entités lexicales du langage $C$ qui sont les suivantes :

   (a) Les mots réservés :

   ```
   auto break case char
   const continue default
   do double else enum
   extern float for goto if
   int long register return
   short signed sizeof static
   struct switch typedef union
   unsigned void volatile while
   ```

   (b) Les identificateurs commencent par une lettre et contiennent des lettres, des chiffres ou le caractère `_`.

   (c) Les constantes entières sont des suites de chiffres.

   (d) Les constantes flottantes se composent d'une partie entière, d'un point décimal, d'une partie fractionnaire, d'un $e$ ou d'un $E$, d'un exposant entier eventuellement signé, et d'un suffixe de type eventuel, à savoir $f$, $F$, $\ell$ ou $L$. Chacune des parties entières et fractionnaires se compose d'une séquence de chiffres.
   On peut omettre la partie entière ou la partie fractionnaire (mais pas les deux), on peut aussi omettre le point décimal ou le e suivi de l'exposant mais pas les deux.

   (e) Les opérateurs sont :

   ```
   ++ -- & * + - ~ ! sizeof
    / % << >> < > <= >= == != ^ | && ||
   = *= /= %= += -= <<= >>= &= ^= |=
   ```

   (f) Les constantes de type `string` sont formées d'une suite de caractères entourés de guillemets. Elles peuvent contenir
   - Des facteurs de `\*` ou de `*/` sans pour autant contenir de commentaires
   - Les caractères échappés `\a`, `\b`, `\f`, `\n`, `\r`, `\t`, `\v`, `\\`, et `\" \'\, \?`
   - `\num` où num désigne un code octal sur trois chiffres
   - `\xnum` où num désigne un code hexadécimal sur deux chiffres

   (g) Les commentaires restreints à une seule ligne peuvent être précédés de `//`. Les autres commentaires sont précédés de `/*` et suivis de `*/`. Ils peuvent tout contenir. Exemple :

   ```
   /******
    Ceci est un commentaire
         *****/
   ```

3. Afin d'avoir une mesure objective de la quantité de code contenu dans le source, on augmente encore le projet pour qu'il affiche à la fin quelques données chiffrées :
   - Le nombre de lignes de code
   - Le nombre de mots clefs, d'opérateurs et d'identificateurs
   - Le nombre de lignes de commentaires
   - Le nombre de lignes blanches

   On souhaite enfin afficher un index des identificateurs utilisées dans le source (pour chaque identificateur, afficher les lignes où ils apparaissent).

4. Faire en sorte que l'analyseur lexical repère certaines erreurs :
   – un caractère non prévu (par exemple `@`),
   – un début de constante chaîne de caractères terminée par `<EOF>`
   – un début de commentaire `/*` terminée par `<EOF>`.
   L'analyseur devra afficher le numéro de ligne où l'erreur s'est produite, ainsi que le type d'erreur.