Théorie des Langages et de la Compilation (Partie pratique)
Énoncés de la séance 11 (LEX)

S. Collette G. Geeraerts

Année académique 2006–2007

Exercices

1. Écrire un filtre qui compte le nombre de caractères, de mots et de lignes d'un fichier.
2. Écrire un filtre qui numérote les lignes d'un fichier (hormis les lignes blanches).
3. Écrire un filtre qui n'imprime que les commentaires d'un programmes. Ceux-ci sont compris entre `{ }`.
4. Écrire un filtre qui transforme un texte en remplaçant le mot `compilateur` par `beurk` si la ligne débute par `a`, par `schtroumpf` si la ligne débute par `b` et par `youpi ! !` si la ligne débute par `c`.
5. Écrire une *fonction d'analyse lexicale* à l'aide de LEX. Les *tokens* reconnus seront :
   - Les nombres décimaux (en notation scientifique) ;
   - Les identifiants de variables ;
   - Les opérateurs relationnels ($<$, $>$, $\leq$, *etc*) ;
   - Les mots-clefs `si`, `sinon` et `alors`.

   Cette fonction a pour but d'être utilisée dans un analyseur syntaxique écrit en YACC.

Exercice supplémentaire

Écrire un petit programme qui *embellit* du code C à l'aide de `(f)lex`. Ce programme lira un fichier C et l'affichera ainsi :

- Indenter correctement le code ;
- Les mots clef en gras (`while`, `for`, ...) ;
- Les *strings* (délimités par des `"`) en vert ;
- Les entiers en bleu ;
- Les commentaires (délimités pas `/*` et `*/`, ou bien les lignes commençant par `//`) en noir sur fond blanc (vidéo inverse).

Pour ce faire, vous pouvez vous aider de la fonction `void textcolor(int attr, int fg, int bg)`

- `attr` permet de mettre en gras (valeur `BRIGHT`), en vidéo inverse (`REVERSE`) ou en normal (`RESET`).
- `fg` et `bg` permettent de préciser la couleur du texte et du fond (valeurs `GREEN`, `BLUE`, `WHITE`, `BLACK`...)

Rappels

Un *filtre* est un programme qui lit un texte sur l'*input* standard et affiche sur l'*output* standard ce texte modifié. Par exemple, un filtre qui remplace tous les `a` par des `b` et qui reçoit `abracadabra` sur *input*, écrira `bbrbcbdbbra` sur *output*.

Format de la spécification

Une spécification LEXcomporte trois parties, séparées par des `%%` :

- **Partie 1** : Déclaration de variables, de constantes et de définitions régulières ;
  - Les définitions régulières sont utilisées comme des « macros » dans les actions ;
- **Partie 2** : Règles de traduction, de la forme : `ExpReg  {Action}`
  - `ExpReg` est une *expression régulière étendue* ;
  - `Action` est un *fragment de code C*, qui sera exécuté chaque fois qu'un *token* satisfaisant `ExpReg` est rencontré.
  - Les actions peuvent faire appel aux expressions régulières de la partie 1 grâce aux { } ;
- **Partie 3** : Procédures de l'utilisateur.
  - Par exemple : `main()` si le *scanner* n'est pas utilisé avec YACC ou BISON.

Variables et actions spéciales

- Dans les *actions*, on peut accéder à certaines variables spéciales :
  - `yyleng` : contient la *taille* du *token* reconnu ;
  - `yytext` : est une variable de type `char*` qui pointe vers la *chaîne de caractères reconnue* par l'expression régulière.
  - `yylval` : qui permet de *passer des valeurs* entières à YACC...
- Il existe aussi une action spéciale : `ECHO` qui équivaut à `printf("%s",yytext)`.

Compilation

Pour obtenir l'exécutable du *scanner* :

- On commence par compiler le fichier `lex.yy.c` grâce à `gcc -c lex.yy.c`, ce qui crée le fichier `lex.yy.o` ;
- On compile les autres fichiers `.c` ;
- On *linke* le tout *avec la librairie* `libl` (`libfl` pour FLEX) : `gcc -o executable fichier1.o ... fichierN.o lex.yy.o -ll`

Exemple

```
chiffre [0-9]
lettre [a-z]|[A-Z]
%%
{lettre}({chiffre}|{lettre})*   {
        printf("J'ai reconnu l'identifiant: %s de longueur %d\n", yytext, yyleng) ;
        }
({chiffre})+    {
        printf("J'ai reconnu un nombre\n") ; ECHO ;
        }
%%
main(){ yylex() ; }
```