Analyse Syntaxique - Annee 2010-2011
Feuille 2

Construction d'analyseurs lexicaux avec `lex`

Exercice 2.1 *Premier exemple*

1. Un petit exemple de programme lex, dans un fichier que l'on nomme *lc.l* :

```
%{
#include <stdio.h>

      int num_lines = 0;
%}
%option noyywrap

%%
\n      ++num_lines;

%%
int main(int argc, char *argv[]){
        yylex();
        printf("number of lines = %d \n",num_lines);
        }
```

   Compiler et tester ce pogramme.

2. Ecrire un analyseur lexical *lwc* qui simule *wc*, c'est à dire qui compte les lignes, les mots, et les caractères d'un texte. Essayer plusieurs définitions raisonnables pour la notion de *mot*.

   Appliquer *lwc* au fichier source *lwc.l*, et comparer avec les résultats du programme standard *wc*.

Exercice 2.2

1. Écrire un source `lex` n'utilisant pas les états, permettant de retirer d'un programme C :
   - les commentaires avec la syntaxe `//...` ;
   - les commentaires avec la syntaxe `/*...*/` ;
2. Écrire un autre source permettant d'arriver au même résultat mais en utilisant les états.
3. Modifier votre analyseur lexical pour qu'il compte le nombre de lignes de commentaires supprimées.

Exercice 2.3

En Java, il existe un autre type de commentaires : les commentaires de documentation. Ils

sont encadrés par les motifs `/**` et `*/`, ils peuvent se terminer par une liste de motifs spéciaux, appelés `tags`. Le nom d'un `tag` commence par le caractère arobase (`@`) et figure nécessairement en début de ligne (en ignorant les espaces et les `*`). Si des mots commençant par `@` figurent ailleurs qu'en début de ligne, ils ne seront pas considérés comme des `tags`. Chaque `tag` a un texte associé (éventuellement sur plusieurs lignes) qui se termine au `tag` suivant ou au motif final `*/`. Exemple :

```
/**
* Returns the character at the specified index.
*
* @param     index  the index of the desired character.
* @return    the desired character.
* @exception StringIndexOutOfRangeException
*              if the index is not in the range <code>0</code>
*              to <code>length()-1</code>.
*/
```

Ecrire un analyseur lexical, dans le format reconnu par `lex`, qui élimine tous les commentaires du fichier java et affiche dans un fichier nommé `docfile` (on utilisera `fprintf`) les `tags` `@param`, `@return` et `@exception` des commentaires de documentation figurant dans le fichier java, avec leur texte associé.

Exercice 2.4

En langage C, une chaîne de caractères littérale est délimitée par des apostrophes doubles (caractère ") ; à l'intérieur d'une chaîne, $\backslash x$ désigne un caractère spécial, pour $x = t, n, \ldots$ et désigne 'x' lui-même dans les autres cas (y compris si x=").
Une chaine peut être écrite sur plusieurs lignes à condition que chaque ligne intermédiaire se termine par \. Ecrire un analyseur lexical qui isole les chaînes de caractères littérales dans un programme C.

Exercice 2.5

On donne la grammaire suivante (les terminaux sont en fonte : `terminal`, les non terminaux en SMALL CAPS, ε désigne le mot vide) :

1	PROGRAMME	$\longrightarrow$	`début` LISTE_INSTRUCTIONS `fin`
2	LISTE_INSTRUCTIONS	$\longrightarrow$	INSTRUCTION FIN_LISTE_INSTR
3	FIN_LISTE_INSTR	$\longrightarrow$	INSTRUCTION FIN_LISTE_INSTR
4	FIN_LISTE_INSTR	$\longrightarrow$	ε
5	INSTRUCTION	$\longrightarrow$	`id :=` EXPRESSION `;`
6	INSTRUCTION	$\longrightarrow$	`lire (` LISTE_ID `) ;`
7	INSTRUCTION	$\longrightarrow$	`écrire (` LISTE_EXPR `) ;`
8	LISTE_ID	$\longrightarrow$	`id` FIN_LISTE_ID
9	FIN_LISTE_ID	$\longrightarrow$	`, id` FIN_LISTE_ID
10	FIN_LISTE_ID	$\longrightarrow$	ε
11	LISTE_EXPR	$\longrightarrow$	EXPRESSION FIN_LISTE_EXPR
12	FIN_LISTE_EXPR	$\longrightarrow$	`,` EXPRESSION FIN_LISTE_EXPR
13	FIN_LISTE_EXPR	$\longrightarrow$	ε
14	EXPRESSION	$\longrightarrow$	EXPR_SIMPLE FIN_EXPR_SIMPLE
15	FIN_EXPR_SIMPLE	$\longrightarrow$	OP EXPR_SIMPLE FIN_EXPR_SIMPLE

16 FIN_EXPR_SIMPLE $\longrightarrow$ ε
17 EXPR_SIMPLE $\longrightarrow$ `(` EXPRESSION `)`
18 EXPR_SIMPLE $\longrightarrow$ `id`
19 EXPR_SIMPLE $\longrightarrow$ `int`
20 OP $\longrightarrow$ `+`
21 OP $\longrightarrow$ `-`

On suppose qu'un identificateur est une suite de lettres ou de chiffres commençant par une lettre et un entier une suite de chiffres.

1. Quels lexèmes l'analyseur lexical devra-t-il retourner à l'analyseur syntaxique ?
2. Construire un analyseur lexical pour cette grammaire en utilisant `lex`. L'analyseur affichera le flot de léxèmes.

Exercice 2.6

Ecrire un programme qui évalue des expressions postfixées, en *incluant* un analyseur lexical qui identifie les constantes numériques réelles et les opérateurs. Lorsque l'analyseur lexical identifie un nombre, celui-ci doit être empilé. Lorsque l'analyseur lexical identifie un opérateur, l'opération doit être exécutée, en prenant pour opérandes les deux nombres en sommet de pile ; le résultat de l'opération remplace, sur la pile, les opérandes. En fin d'analyse, le progamme affichera la valeur de l'expression analysée.