

Compilation

Analyseurs lexicaux

PAR DAVID TELLER

David.Teller@univ-orleans.fr

Au cours des semaines qui suivent, nous allons chercher à définir un langage ressemblant à l'extrait suivant :

```
function f : (x,y,a,b) ->
  when x < y -> a
  when x >=y -> b
end
```

```
function g : (x) ->
  print(f(x,1-x,0,1))
end
```

Pour ce TP, nous utiliserons le logiciel FLEX, dont le manuel est disponible en français à l'adresse <http://www.delafond.org/traducmanfr/man/man1/flex.1.html>. Ce logiciel sert à convertir un flux (c'est-à-dire, typiquement, un fichier) en une liste de symboles utilisables pour l'analyse syntaxique.

Dans le langage que nous souhaitons définir, les symboles à reconnaître sont :

- les mots-clé **function**, **end**, **when**
- les opérateurs **<**, **<=**, **>**, **>=**, **<>**, **=**, **+**, **-**, **/**, *****
- les nombres, contenant éventuellement un séparateur décimal .
- les noms de variables ou de fonctions (toute suite non vide de lettres à part les mots-clés)
- les symboles additionnels **:**, **->**, **(**, **,**, **)**
- la fin du fichier.

de plus, cet analyseur doit ignorer les espaces, retours à la ligne et tabulations.

Reconnaissance de base

Pour le moment, nous allons construire un reconnaiseur pur, c'est-à-dire un programme qui va juste afficher **FUNCTION** à chaque fois qu'il rencontre le mot clé **function**, **NUMBER** à chaque fois qu'il rencontre un nombre, etc.

L'extrait qui suit est un reconnaiseur minimaliste, qui ne gère presque rien.

```
%{
#include <stdio.h> //Inclure stdio.h pour pouvoir utiliser printf
%}

%%
"function"      printf("%s ", "FUNCTION");
"end"           printf("%s ", "END");
[ \n \t]+      /* ignorer les espaces, retours chariots... */;
%%
```

pour utiliser ce reconnaiseur, sauvegardez-le sous le nom **reconnaiseur.yy** puis

```
flex reconnaiseur.yy -o reconnaiseur.lex.c
```

```
gcc reconnaisseur.lex.c -lfl -o reconnaisseur
./reconnaisseur < un_nom_de_fichier
```

Exercice 1. Ajoutez à ce reconnaisseur les autres mot-clés, opérateurs et symboles additionnels.

Exercice 2. Ajoutez à ce reconnaisseur les noms de variables/fonctions – lorsqu’un nom est reconnu, affichez ce nom. Pour ce faire, utilisez la variable spéciale `yytext`.

Exercice 3. Ajoutez à ce reconnaisseur les nombres. Les règles nécessaires se trouvent dans le manuel. Vous utiliserez `atof` pour convertir une chaîne de caractères en nombre flottant.

À l’issue de cet exercice, appliquer le reconnaisseur à l’exemple donné plus haut doit produire un résultat similaire à

```
FUNCTION f COLUMN OPEN x COMMA y COMMA a COMMA b CLOSE MAPS TO WHEN x LESS y THEN
a WHEN x GREATEREQ y THEN b END FUNCTION g COLUMN OPEN x CLOSE MAPS TO print OPEN
f OPEN x COMMA 1.000000 MINUS x COMMA 0.000000 COMMA 1.000000 CLOSE CLOSE END EOF
```

Préparation de l’analyse syntaxique

L’objectif de cette partie est de passer d’un reconnaisseur à un analyseur lexical, qui renverra cette fois des informations utilisables par l’analyseur syntaxique.

L’extrait qui suit contient quelques bribes de la chose.

```
%{
#include <stdio.h> //Inclure stdio.h pour pouvoir utiliser printf
#include "toy.yacc.h" //...ce fichier définit toutes les constantes

YYSTYPE yylval; //...

}%
%%
"function" return FUNCTION;
"end" return END;
([0-9])+."([0-9])* yylval.float_value = atof(yytext); return NUM;
[ \n \t]+ /* ignorer les espaces, retours chariots... */;
%%
```

Exercice 4. Compléter l’extrait précédent. La définition de `yylval` et `yytext` se trouve dans la documentation de FLEX.