Université de Nice-Sophia Antipolis — Licence 3 Informatique
UE Automates & Langages — 2011–2012

TP n°1

FLEX

Le principe de FLEX

FLEX[1] est un générateur automatique d'analyseurs lexicaux. Autrement dit, il produit un super-automate à partir d'un ensemble d'expressions régulières donné.

Le programme produit est écrit en langage C et stocké dans un fichier appelé `lex.yy.c`. Il contient le sous-programme `yylex()`. Il faut appeler le compilateur C pour obtenir un exécutable, en faisant l'édition de liens avec la bibliothèque de FLEX (le nom de la bibliothèque est **fl**). Les commandes UNIX correspondantes sont les suivantes :

```
flex fichierSource.lex
gcc -o fichierExecutable lex.yy.c -lfl
```

L'exécutable obtenu peut *analyser* l'entrée standard d'UNIX, ce qui veut dire qu'il peut la parcourir et identifier la nature des mots qui la composent. On rappelle que la redirection de l'entrée standard est possible grâce au signe '<'. Pour tester l'analyseur lexical produit par FLEX sur un fichier test `fichierTest`, la commande à utiliser est la suivante :

```
./fichierExecutable < fichierTest
```

L'analyseur peut être utilisé avec l'entrée standard qui se termine avec `Ctrl-d` et peut éventuellement recevoir des arguments.

Un fichier en entrée de FLEX doit respecter la forme suivante :

```
... /* règles de définition */
... /* déclarations C éventuelles (lignes débutant par des espaces) */
%%
... /* règles de reconnaissance de la forme */
expressionsRégulières     actionsEcritesEnC
%%
... /* sous-programmes de l'utilisateur écrits en C */
```

Les règles représentent les décisions de contrôle de l'utilisateur. Les actions sont effectuées dès lors qu'un mot correspondant à une des expressions régulières est trouvé. Si plusieurs mots préfixes de l'entrée courante peuvent être reconnus, FLEX choisit le plus long. En outre, si plus d'une expression régulière permet de reconnaître un même mot, FLEX choisit la première. Donc, l'ordre dans lequel les expressions régulières sont données est significatif.

Les expressions régulières

Une expression régulière pour FLEX est constituée d'une suite de caractères et d'opérateurs dont voici la liste :

[1] Le manuel de FLEX est en ligne à `http ://flex.sourceforge.net/manual/`

`""`	le texte compris entre les guillemets n'est pas interprété
`\`	le caractère suivant est interprété tel quel
`[ ]`	pour définir un ensemble de caractères ; à l'intérieur des crochets, les opérateurs sont ignorés sauf `-` (pour les définitions d'intervalle), `^` (pour le complémentaire de l'ensemble) et `\` (séquence d'échappement ordinaire)
`?`	élément facultatif
`.`	n'importe quel caractère (sauf la fin de ligne)
`*`	0 ou plusieurs fois
`+`	1 ou plusieurs fois
`|`	alternative
`( )`	pour grouper
…	…

Des *définitions* permettent d'écrire des expressions régulières de façon plus concise. On les déclare de la façon suivante :

```
nom traduction
```

Pour y faire référence dans une expression régulière, il suffit d'entourer le nom de la définition par des accolades :

```
{nom}
```

Les actions

Lorsqu'un mot est identifié, l'action par défaut est celle qui consiste à copier l'entrée standard sur la sortie standard. On peut lui substituer d'autres actions écrites en C et pouvant utiliser les variables et fonctions suivantes :

`yytext`	tableau de caractères contenant la chaîne reconnue pour l'expression régulière en cours
`yyleng`	nombre de caractères de la chaîne reconnue
`yymore()`	indique qu'il faut rajouter la prochaine entrée reconnue à cette entrée
`yyless(n)`	n caractères à retenir de l'entrée courante
`input()`	prochain caractère lu
`output(c)`	écrit le caractère c sur la sortie
`unput(c)`	retourne le caractère c sur le flot d'entrée

Des exemples

Ne les recopier pas, ce sont juste des exemples ... de vrais exercices suivent.

1. Si on donne le fichier suivant en entrée à FLEX, que pensez-vous que FLEX produise ?

```
mot     L3info
        int i = 0;
%%
{mot}    i++;
\n       ;
.        ;
%%
int main (int argc, char *argv[]) {
  yylex();
  printf("%d\n", i);
  return 0;
}
```

2. A présent, quel est la finalité du programme suivant ?

```
mot     [A-Za-z]+
        #include <strings.h>
        char *mot = NULL;
        int i = 0;
%%
{mot}    if (strcmp(mot, yytext) == 0) i++;
\n       ;
.        ;
%%
int main (int argc, char *argv[]) {
  if (argc != 2) {
    fprintf(stderr, "usage: %s [a-zA-Z]+\n", argv[0]);
    exit(2);
  }
  mot = argv[1];
  yylex();
  printf("%d\n", i);
  return 0;
}
```

Les exercices

Ex. 1) Le but est de disposer d'un programme décidant si un mot est une forme verbale du verbe *être* en anglais. Trouvez et mettez en œuvre le source FLEX adéquat. N'oubliez pas d'aller visiter le fichier `lex.yy.c` produit.

Ex. 2) Ecrivez un source FLEX permettant de reconnaître et d'afficher n'importe quel mot. Quel est l'effet du mot-clé `REJECT` ajouté après l'instruction `printf`. Exécutez-le par exemple sur le mot `Abracadabra` afin de comprendre ce qu'il se passe.

Ex. 3) Un exécutable similaire à la commande `wc` d'UNIX peut être créé par FLEX. Cette commande affiche sur la sortie standard le nombre de lignes, de mots et de caractères du texte passé en paramètre. Trouvez le fichier source à donner en entrée à FLEX pour qu'il produise le programme adéquat.

Ex. 4) Petit analyseur lexical Nous allons engendrer un analyseur lexical pour un petit langage à la PASCAL dont les *lexèmes* sont les suivants :

- un identificateur débute obligatoirement par une lettre, suivie de lettres ou de chiffres ;
- les mots réservés du langage sont *programme*, *debut*, *fin*, *si*, *alors*, *sinon*, *finsi* ;
- les chaînes de caractères sont comprises entre deux guillemets ;
- les nombres entiers décimaux ;
- les opérateurs sont : '$+$', '$-$', '$*$', '$/$', '*div*', '*mod*' ;
- le signe de l'affectation est ' :=' ;
- le séparateur d'énoncés est le ' ;' ;

Produisez un analyseur lexical permettant de découper l'entrée en unités lexicales et indiquant pour chacune son lexème d'appartenance. Dans un compilateur, l'étape d'après serait l'*analyse syntaxique* du texte, mais on en est pas encore là. Comment s'y prendre ?

1. Trouver les expressions régulières de chaque lexème.
2. Produire automatiquement l'analyseur qui affichera les mots reconnus sous la forme :

   « lexème : mot »

   à raison d'un mot par ligne.
3. *(facultatif et plus technique ...)* Ajoutez le traitement des commentaires sachant qu'ils sont compris entre (* et *).