Université Côte d'Azur
Automates & Langages

L3 Info, L3 MI, L3 ST
2023–24

TP n°1

Génération automatique d'analyseurs lexicaux

Le principe

L'*analyse lexicale* est la première tâche effectuée par un compilateur, celle où un programme est décomposé en *lexèmes*. Chaque lexème appartient à un langage rationnel décrit par une expression régulière. Parmi ces langages, on trouve les identifiants, l'ensemble des mots-clefs, les séparateurs, les opérateurs, telle ou telle sorte de nombres *etc*.

FLEX est un générateur automatique d'*analyseurs lexicaux*, la version GNU du programme LEX. Plus récemment, SLY pour SLY LEX YACC est l'équivalent en PYTHON 3.6 assorti du générateur d'*analyseurs syntaxiques* YACC [1]. L'analyseur lexical est généré à partir d'un ensemble d'expressions régulières et fonctionne comme un super-automate pour analyser lexicalement un texte et effectuer éventuellement des opérations à chaque lexème reconnu.

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`) ou bien celle de LEX (son nom est `l`). 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 de 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. Après identification d'un mot comme lexème, ses lettres sont consommées dans le sens qu'elles sont retirées du flux d'entrée `yyin`. Elles ne peuvent pas être considérées pour le lexème suivant. 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.

1. Le manuel de FLEX est en ligne à `https://westes.github.io/flex/manual/`.

Les expressions régulières

Une expression régulière FLEX est constituée d'une suite de caractères et d'opérateurs parmi :

`""`	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 lexème 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

Premier exemple et exercices

Exercice 0) Recopiez le petit exemple suivant dans le fichier suffixé par `.lex`, par exemple `ex.0.lex`. Faites générer l'analyseur correspondant par FLEX puis testez l'exécutable produit en utilisant l'entrée standard ou en la redirigeant sur un fichier à analyser. Comprenez ce qu'il fait et n'oubliez pas d'aller visiter le fichier `lex.yy.c` produit.

```
motif   (bla)+
     // déclarations en C
     int i = 0;
%%
{motif}    i++;
\n         ;
.          ;
%%
int main (int argc, char *argv[]) {
  yylex();
  printf("%d\n", i);
  return 0;
}
```

Exercice 1) Récupérez sur le site MOODLE du cours le fichier `ex.1.lex` et examinez son contenu. Testez l'analyseur obtenu afin de comprendre précisément son usage ainsi que son fonctionnement. Par rapport à l'exercice précédent, notez la différence entre ce que nous avons appelé `mot` et `motif`.

Exercice 2) On souhaite disposer d'un programme interactif pour décider si un mot est une forme verbale du verbe *être* en anglais. Ecrivez et mettez en œuvre le source FLEX adéquat, une exécution pourrait être la suivante :

```
as
as: Sorry, I can't recognize ...

is
is: To be !
```

Exercice 3) Ecrivez un source FLEX permettant de reconnaître et d'afficher tous les mots contenus dans un texte. L'exécutable doit se comporter comme la commande `cat` d'UNIX.

Exercice 4) Le mot-clef `REJECT` de FLEX, introduit après les actions en C, a pour effet de refuser le *lexème* courant et de rechercher le *suivant*. Ce dernier peut alors être capturé par la même expression régulière (il est alors plus court) ou par une autre expression régulière que celle du lexème rejeté.

Vous pouvez vous faire une idée de son fonctionnement grâce au petit exemple suivant exécuté sur le mot `abracadabra` :

```
%%
[a-zA-Z]+     printf("%s\n", yytext); REJECT ;
\t | \n       ;
.             ;
```

Ecrivez un fichier FLEX de sorte à compter le nombre de voyelles, minuscules, majuscules ainsi que le nombre de lettres total d'un texte en utilisant les possibilités de `REJECT`.

Exercice 5) 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.

Exercice 6) *(Facultatif)* Ecrire un programme permettant d'extraire d'un texte et de recopier les mots à condition qu'ils soient de longueur croissante et aussi de faire la somme des nombres entiers apparaissant dans ce texte.

Petit analyseur lexical

Exercice 7) Nous allons engendrer un petit analyseur lexical forcément incomplet pour un sous-ensemble du langage PYTHON dont la partie lexicale est détaillée dans :

https://docs.python.org/3/reference/lexical_analysis.html

Voici un petit échantillon de ses unités lexicales :

— les mots-clefs du langage sont :

```
False   await     else     import    pass
None    break     except   in        raise
True    class     finally  is        return
and     continue  for      lambda    try
as      def       from     nonLocal  while
assert  del       global   not       with
async   elif      if       or        yield
```

— les opérateurs sont :

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

— les délimiteurs sont :

```
(    )    [     ]     {     }    ,
:    .    ;     @     =     ->   +=
-=   *=   /=    //=   %=    @=   &=
|=   ^=   >>=   <<=   **=
```

— un identificateur est une suite de caractères hors mots-clefs pris parmi les lettres en minuscules, les lettres en majuscules, l'*underscore* et les chiffres exceptés pour le premier caractère ;
— les chaînes de caractères sont comprises entre deux guillemets ou *quote* ;
— les nombres entiers (décimaux, binaires, octaux, hexadécimaux) ;
— les nombres flottants ;
— les commentaires délimités par des ♯ ;
— *etc*

L'analyse lexicale a pour but de transformer un programme ou un texte en *tokens* c'est-à-dire en unités lexicales correspondant à chaque lexème. Dans un compilateur, l'étape d'après serait l'*analyse syntaxique* du texte.

Produisez un analyseur lexical permettant de découper un programme `Python` en lexèmes et indiquant pour chacun son *token* d'appartenance. L'idée n'est pas de générer un analyseur complet mais plutôt de bien comprendre les mécanismes de l'analyse lexicale.

1. Trouvez les petites expressions régulières de chaque langage ;
2. Procédez à l'analyse lexicale automatique d'un programme : on affichera les unités lexicales reconnues à raison d'une par ligne sous la forme :

« unité : lexème »