

TP1 - Analyse lexicale Mini projet

A : Mettre en œuvre un analyseur lexical en C pour le langage C

(Voir Exercice 8 TD 1.) Veuillez choisir, seulement, cinq unités lexicales dont nombre et identificateur.

B : Comment utiliser l'outil Flex ?

Rappel outil Flex

La tâche principale d'un analyseur lexical est de lire un texte source (suite de caractères) et de produire comme résultat une suite d'unités lexicales. La reconnaissance des unités lexicales est basée sur la notion d'expressions régulières. Théoriquement, la construction d'un analyseur lexical consiste à :

- Définir les unités lexicales.
- Modéliser chaque unité lexicale par une expression régulière.
- Représenter chaque expression régulière par un diagramme de transition (automate).
- Construire le diagramme global.
- Implémenter à la main le diagramme obtenu.

Généralement, l'implémentation à la main d'un diagramme avec un grand nombre d'états est une tâche qui n'est pas assez facile. En outre, si on a besoin d'ajouter ou modifier une unité lexicale, il faut parcourir tout le programme pour effectuer les modifications nécessaires. Plusieurs outils ont été bâtis pour simplifier cette tâche. (Flex par exemple)

Flex :

L'outil Flex (version GNU de LEX) est un générateur d'analyseurs lexicaux. Il accepte en entrée des unités lexicales sous formes d'expressions régulières et produit un programme écrit en langage C qui, une fois compilé, reconnaît ces unités lexicales. L'exécutable obtenu lit le texte d'entrée caractère par caractère jusqu'à l'identification de plus long préfixe du texte source qui concorde avec l'une des expressions régulières. Un fichier de spécifications Flex se compose de quatre parties :

```
%{  
  Les déclarations en c  
}%  
Déclaration des définitions régulières  
%%  
Règles de traduction  
%%  
Bloc principal et fonctions auxiliaires en C
```

Définitions régulières :

Une définition régulière permet d'associer un nom à une expression régulière et de se référer par la suite dans la section de règles à ce nom, plutôt qu'à l'expression régulière.

Règles de traduction :

```
exp1 { action1}  
exp2 { action2}  
... ...  
expn { actionn}
```

Chaque *exp_i* est une expression régulière qui modélise une unité lexicale. Chaque *action_i* est une suite d'instructions en C.

premiers pas en Flex

Partie 1

1) Ecrire le programme Flex suivant permettant de dire si une chaîne en entrée est un nombre binaire ou non. Le fichier doit être enregistré avec l'extension **.l** (exemple : *binair.l*)

```
%%  
(0|1)+ printf ("c'est un nombre binaire");  
. * printf ("ce n'est pas un nombre binaire");  
%%  
int yywrap(){return 1;}  
main ()  
{  
  yylex ();  
}
```

2) Créer un nouveau répertoire et y placer le fichier *binair.l*.

3) Lancer l'invite de commande propre à Flex Windows et compiler votre fichier « *binair.l* » à l'aide de la commande: **flex binair.l**. S'il y aura pas de problème, un fichier *lex.yy.c* sera créé.

Ensuite on va compiler le fichier *lex.yy.c* et générer un fichier C *.exe* à l'aide de la commande : **gcc lex.yy.c -o prog**

Enfin, vous pouvez appeler le programme *prog.exe* à travers la commande **prog.exe**

4) Créer un fichier texte dans votre répertoire de travail et y éditer quelques lignes.

Rajouter le code suivant dans le *main()* de votre programme.

```
int main(int argc, char *argv[])  
{  
  ....  
  ++argv, --argc;
```

```

    if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
    else
        yyin = stdin;
    yylex();
}

```

Modifier votre programme pour rediriger la sortie standard vers un fichiers **resultat.txt**.

Partie 2

1. Ecrire et compiler le fichier de spécifications suivant :

```

pairpair (aa|bb)*((ab|ba) (aa|bb)*(ab|ba) (aa|bb)*)*
%%
{pairpair} printf ("[%s]: nombre pair de a et de b\n", yytext);
a*b* printf ("[%s]: des a d'abord et des b ensuite\n", yytext);
.
%%
int yywrap() {return 1;}
main()
{
    yylex();
}

```

2. Tester les entrées babbaaab abbb aabb baabbbb bbaabbba baabbbbab aaabbbba.

3. Même question en permutant les deux lignes :

```

a*b* printf ("[%s]: des a d'abord et des b ensuite \n",yytext);
(pairpair) printf ("[%s]: nombre pair de a et de b \n",yytext);

```

4. Y'a t-il une différence ? Laquelle ? Pourquoi ?

5. On considère l'unité lexicale **id** définie comme suit : un identificateur est une séquence de lettres et de chiffres. Le premier caractère doit être une lettre. Ecrire à l'aide de Flex un analyseur lexical qui permet de reconnaître à partir d'une chaîne d'entrée l'unité lexicale **id**.

6. Modifier l'exercice précédent pour que l'analyseur lexical reconnaisse les deux unités lexicales **id** et **nb** sachant que **nb** est une unité lexicale qui désigne les entiers naturels.

Annexe et référence

Référence **gnu** : <http://www.gnu.org/software/bison/manual/>

Variables flex:

yyin fichier de lecture (par défaut: *stdin*)

yyout fichier d'écriture (par défaut: *stdout*)

char yytext [] : tableau de caractères qui contient le lexème accepté.

int yyleng : la longueur du lexème accepté.

Fonctions :

int yylex () : fonction qui lance l'analyseur.

Int yywrap () : fonction qui est toujours appelée à la fin du texte d'entrée. Elle retourne 0 si l'analyse doit se poursuivre et 1 sinon.

Expressions régulières abrégées

| EXPRESSION | CHAÎNES RECONNUES | EXEMPLE |
|--------------------------------------|---|----------------|
| <i>c</i> | le caractère unique non-opérateur <i>c</i> | <i>a</i> |
| <i>\c</i> | le caractère <i>c</i> littéralement | <i>*</i> |
| <i>"s"</i> | la chaîne <i>s</i> littéralement | <i>"**"</i> |
| <i>.</i> | tout autre caractère qu'une fin de ligne | <i>a.*b</i> |
| <i>^</i> | le début d'une ligne | <i>^abc</i> |
| <i>\$</i> | la fin d'une ligne | <i>abc\$</i> |
| <i>[s]</i> | un caractère quelconque de la chaîne <i>s</i> | <i>[abc]</i> |
| <i>[^s]</i> | un caractère quelconque ne faisant pas partie de la chaîne <i>s</i> | <i>[^abc]</i> |
| <i>r*</i> | zéro, une ou plusieurs chaînes reconnues par <i>r</i> | <i>a*</i> |
| <i>r+</i> | une ou plusieurs chaînes reconnues par <i>r</i> | <i>a+</i> |
| <i>r?</i> | zéro ou une fois <i>r</i> | <i>a?</i> |
| <i>r{m,n}</i> | entre <i>m</i> et <i>n</i> occurrences de <i>r</i> | <i>a{1,5}</i> |
| <i>r₁r₂</i> | <i>r₁</i> suivi par <i>r₂</i> | <i>ab</i> |
| <i>r₁ r₂</i> | <i>r₁</i> ou <i>r₂</i> | <i>a b</i> |
| <i>(r)</i> | mêmes chaînes que <i>r</i> | <i>(a b)</i> |
| <i>r₁/r₂</i> | <i>r₁</i> lorsqu'il est suivi par <i>r₂</i> | <i>abc/123</i> |