MODULE : Techniques de Compilation

DEPARTEMENT : GLSI - ISI

NIVEAU : 1ère Ingénieur

TP1 - Analyse lexicale (Outil Flex)

Rappel

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 :

*exp*1 **{** *action*1**}**
exp2 **{** *action*2**}**
… …
expn **{** *actionn***}**

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

Exercice 1 – 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 : *binaire.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 binaire.l.

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

Exercice 2

Ecrire un analyseur lexical qui permettra de :

- Compter le nombre de mots.
- Compter le nombre de ligne.
- Compter le nombre de caractères.
- Retourner la somme des nombres lus.

L'exécution doit être effectuée sur un fichier texte (en entrée) et le résultat est redirigé vers un fichier **resultat.txt.**

Exercice 3

On considère un répertoire téléphonique où chaque ligne est écrite selon le format suivant :
Channouf belgacem @ 04 25 32 15 88
Ben salah jalel @85 75 89 56 56
Soufi mohamed amine@ 45 78 12 37 57
Touons omar chatty @78 65 42 15 19
Brun leon paul@54 89 67 32 45

Donner et exprimer en Flex les définitions régulières pour décrire nom, prénom et téléphone.

a) On s'est restreint aux noms et prénoms exclusivement composés de caractères alphanumériques.
b) Le nom doit commencer par une lettre majuscule.
c) On pourra avoir plusieurs prénoms (écrits qu'en minuscules) séparés par un nombre quelconque de blancs.
d) Le numéro de téléphone est composé de 5 paires de digits séparées par un blanc.

Exercice 4

Ecrire un analyseur lexical permettant d'analyser la requêtes SQL suivante :

```
CREaTE TaBLE utilisateur
(
iduser int PRIMARY KEY,
nom varchar(100),
prenom varchar(100),
date_naissance DATE,
ville varchar(255),
code_postal varchar(5)
)
```

Le résultat d'analyse demandée est :

```
CREaTE          CREATE
TaBLE           TABLE
utilisateur             IDENTIFIER
(               (
iduser          IDENTIFIER
int             DATATYPE
PRIMARY KEY             PRIMARY_KEY
,               COMMA
nom             IDENTIFIER
varchar         DATATYPE
(               (
100             NUMBER
)               )
,               COMMA
prenom          IDENTIFIER
varchar         DATATYPE
(               (
100             NUMBER
)               )
,               COMMA
date            DATATYPE
_naissance              IDENTIFIER
DATE            IDENTIFIER
,               COMMA
ville           IDENTIFIER
varchar         DATATYPE
(               (
255             NUMBER
)               )
,               COMMA
code            IDENTIFIER
_postal         IDENTIFIER
varchar         DATATYPE
(               (
5               NUMBER
)               )
)               )
```

Exercice 5

Ecrire à l'aide de Flex un analyseur lexical qui reconnaît les unités lexicales suivantes:

ENTIER : une constante entière
REEL : une constante réelle
IDENT : un identificateur type C
OP-ARTHM : + , - , * , /
OP-REL : < , > , <= , >= , == , !=
AFFECT: =
MOTCLE: if, else, while
CHAINE : une chaîne de caractères entre guillemets
COMMENTAIRE : une ligne commençant par #

a. Votre analyseur devra ignorer les espaces, tabulations, passages à la ligne et les lignes vides.
b. Afficher pour chaque lexème, son type (ENTIER, REEL,…etc.) et sa valeur.
c. Traiter quelques cas d'erreurs en affichant le numéro de la ligne, le type de l'erreur et la chaîne qui pose le problème.

Par exemple, si le fichier en entrée est le suivant :

```
" fichier exemple "

if a >b a=a+2
"chaîne deux " # commmentaire#
3.5  5.3E2   +4.2
-.2e2     2.1et
truc1  x1  1x
"chaîne incomplète
-t
#fin
```

L'analyseur doit indiquer :

CHAINE : " fichier exemple "
MOTCLE: if
IDENT : a
OP-REL : >
IDENT : b
IDENT : a
AFFECT : =
IDENT : a
OP- ARTHM: +
ENTIER : 2
CHAINE : "chaîne deux "
COMMENT : # commmentaire

REEL : 3.500000
……
REEL : -20.00000
…
Détection des erreurs lexicales avec le numéro de ligne
"Chaîne incomplète ** Erreur : ligne 6 **
fin de chaîne attendue
2.1e exposant attendu
1x identificateur mal formé
-x nombre attendu
#fin #attendu

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
c	le caractère unique non-opérateur c	a
$\backslash c$	le caractère c littéralement	*
"s"	la chaîne s littéralement	"**"
.	tout autre caractère qu'une fin de ligne	a.*b
^	le début d'une ligne	^abc
$	la fin d'une ligne	abc$
$[s]$	un caractère quelconque de la chaîne s	[abc]
$[\hat{}s]$	un caractère quelconque ne faisant pas partie de la chaîne s	[^abc]
$r*$	zéro, une ou plusieurs chaînes reconnues par r	a*
$r+$	une ou plusieurs chaînes reconnues par r	a+
$r?$	zéro ou une fois r	a?
$r\{m,n\}$	entre m et n occurrences de r	a{1,5}
r_1r_2	r_1 suivi par r_2	ab
$r_1 \mid r_2$	r_1 ou r_2	a|b
(r)	mêmes chaînes que r	(a|b)
r_1/r_2	r_1 lorsqu'il est suivi par r_2	abc/123