

Systèmes d'exploitation

Lex et Yacc : partie I (Lex)

Yves STADLER

Codasystem, UPV-M

13 mars 2013

1/10

Introduction

principe

- Écrire une liste d'expression
- Écrire une liste d'actions
- Générer un programme qui va parcourir un fichier et effectue les actions décrites pour les expressions choisies.
- Le fichier .lex donnera un fichier source à utiliser dans le langage choisi.

3/10

Lexical analyser

- Programme qui génère des analyseur lexical
- Appelle aussi parfois "lexer" ou "parser"
- Permet de "parser" un fichier, le scanner.

Historique

- Original by AT&T
- OpenSource fourni dans OpenSolaris et Plan9 des laboratoires Bell.
- On utilise maintenant flex (fast lex)

2/10

Structure d'un fichier Lex

Général

- Definition section
%%
Rules section
%%
language code section
- Langage cible : C (pour ce cours)
- Définition section : permet de donner des instructions pour le fichier source généré
- Rules section : productions de lex
- Language code section : code additionnel dans le langage cible.
- Le fichier aura l'extension .l, .lex la plupart du temps.

4/10

Définitions

- Définition des macros (#define)
- Includes (stdio.h etc)
- Options
- Des commentaires /* */
- Des notions non terminales

Syntaxe

- Tous code du langage cible est entouré de %{ et }%
- Les instructions Lex ne sont pas encadré ainsi
- Une notion non terminale est une association entre un nom et une expression régulière.
- `notion expression`
- `nombre [0-9]+`

5/10

Productions

- Dire à Lex quoi faire en rencontrant une notion
- (notion|expression) action
- Si action est absent, lex recopie les caractères tels quels sur la sortie standard
- Si il y a plus d'une instruction il faut mettre des { } autour des actions
- Pas besoin de mettre %{ et }% pour les actions
- En revanche on pourra mettre entre %{ et }% en début des productions une liste d'instruction que sera incluse dans le code de la fonction générée yylex()
- Il faut mettre les notions entre accolades

6/10

Productions

Exemple

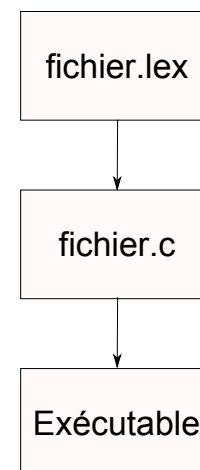
- `[0-9]+ {`
`/* yytext contient la chaîne matchée */`
`printf("Saw an integer: %s\n", yytext);`
`}`
- `. { /* Ignore all other characters. */ }`
- `\item \begin{verbatim}{nombre} {`
`/* yytext contient la chaîne matchée */`
`printf("Saw an integer: %s\n", yytext);`
`}`
- `. { /* Ignore all other characters. */ }`

7/10

Code additionnel

Principe

- Lex va générer un fichier C à partir d'un fichier lex
- Plutôt que de modifier ce fichier pour ajouter le main, on peut l'ajouter dans la partie code additionnel.
- On peut aussi y écrire d'autres fonctions bien sur !



8/10

Syntaxe

```
int main(void) {  
/* Du code */  
  
/* On appelle l'analyseur lexical */  
yylex();  
/* Du code */  
return 0;  
}
```

Options disponibles

- %pointer %array pour choisir le type de yytext (pointeur ou tableau externe)
- %noyywrap Par défaut yylex() appelle la fonction yywrap() pour continuer les traitements après avoir détecté une notion. Cela nous permettra de réaliser des analyse grammaticales. Si il n'y a pas de traitement supplémentaire, on précise cette option.

Variables speciales

- yyin : entrée standard de yylex (défaut : stdin)
- yyin = fopen("fichier", "r");
- yytext la chaîne de caractère mise en correspondance.