Analyse lexicale: Outil Flex

Analyse lexicale: outil flex

- Description de l'outil Flex
- Format d'un programme Flex
- Les motifs
- Fonctionnement des analyseurs Flex
- Les actions prédéfinies

Ecriture d'un analyseur lexical

Il y a 3 méthodes pour développer un analyseur lexical :

- Méthode manuelle : on écrit à la main un analyseur lexical comme un programme en utilisant un langage de programmation évoluée comme *C*, *C* + + ou *Java*.
- Méthode par automate : on construit un automate fini déterministe qui représente le lexique du langage source, puis on le muni d'actions d'analyse lexicale.
- Méthode automatique : on écrit une spécification d'un analsyeur lexical à l'aide d'un programme qui génère automatiquement l'analyseur lexical spécifiée. Cette spéecification est faite par un ensemble d'expressions régulieres enrichies par des actions d'analyse lexicale.

Flex: Fast Lexer

- **Flex** est un outil de **génération automatique d'analyseurs lexicaux**.
- Un fichier *Flex* contient la description d'un analyseur lexiacl à générer.
- Cette description est donnée sous la forme d'expressions régulières etendues et du code écrit en langage *C* (ou *C* + +).
- *Flex* génère comme résultat un fichier contenant le code *C* du future analyseur lexical.
- Ce fichier est nommé **lex.yy.c**.
- La compilation de ce fichier par un compilateur *C*, génère finalement le code executable de l'analyseur lexical en question.

Flex: Fast Lexer

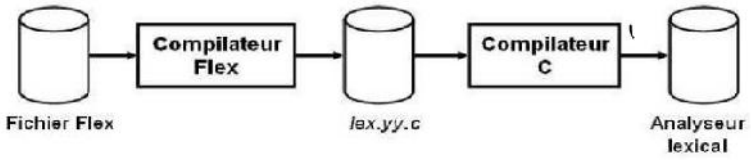

Figure: Processus de création d'un analyseur lexcical avec *Flex*

Flex: Fast Lexer

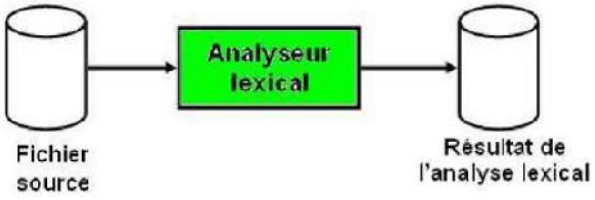

Figure: Utilisation d'un analyseur lexcical généré avec *Flex*

Flex: Fast Lexer

- Lorsque l'exécutable est mis en œuvre, il analyse le fichier source pour chercher les occurrences d'expressions régulières.
- Lorsqu'une chaîne est reconnue par une expression régulière, il exécute le code C correspondant.

Flex: Fast Lexer

Les étapes à suivre pour obtenir un analyseur avec *Flex* :

1. Ecrire, puis enregistrer votre analyseur dans un fichier portant une extension ".flex" ou ".lex". Par exemple, "scan.flex".
2. Compiler votre analyseur par la commende `flex` :

```
flex scan.flex
```

3. Compiler le fichier par la commande `gcc` le fichier `lex.yy.c` produit par l'etape précedente :

```
gcc -o lex.yy.c scan -lfl
```

4. Lancer l'analyseur en utilisant le nom de celui-ci : `scan`

Flex: Fast Lexer

- Par defaut, le texte à analyser est lu à partir de l'entrée standard (le clavier) et le resultat de l'analyse lexical est affiché sur la sortie standard (l'ecran).
- On peut indiquer au compilateur le fichier source à analyser :

scan < input.txt

- On peut indiquer au compilateur d'afficher le résultat de l'anayse dans un fichier :

scan > output.txt

- On peut combiner les deux :

scan < input.txt > output.txt

Flex: Fast Lexer

Exemple 1:

Ecrire un analyseur lexical qui remplace toute occurrence de lettre ”a” par une lettre ”z”, et inversement.

Solution:

```
%%
”a” printf(”z”);
”z” printf(”a”);
```

Flex: Fast Lexer

Fonctionnement

- Les deux symboles %% marquent le debut de la **section des règles**. Ils doivent être écrits en première colonne.
- Le code :
  ”a” printf(”z”)
  est une **règle**.
- Chaque règle contient deux éléments :
  une expression regulière;
  et du code C.
- Dans le code précedent, l'expression regulière est "a" et le code C est l’instruction printf(”z”).

Flex: Fast Lexer

Fonctionnement

- Signification du code : lorsque l'analsyeur renconnaît une lettre ”a” dans le fichier source, il va la remplacer par une lettre ”z”.
- On a un résultat similaire avec la seconde règle :
  ”z” printf(”a”)
  par laquelle l’analyseur remplacera chaque lettre ”z” par une lettre ”a”.

Flex: Fast Lexer

Remarque:

- Une expression régulière doit toujours commencer en première colonne.
- Il faut laisser au moins un caractères blanc (espace ou tabulation) entre l'expression régulière et le code C associé.
- Le code C peut contenir plusieurs instructions. Les acollades { et } doivent être utilisées si c'est le cas.

Flex: Fast Lexer

Exemple 2:

Ecrire un analyseur lexical qui calcule le nombre de lignes contenu dans le fichier source.

```
%{
  int nblignes = 0;
%}
%%
\n nb_lignes++;
%%
main() {
  yylex();
  printf(”nombre de lignes : %d”, nblignes); }
```

Flex: Fast Lexer

Fonctionnement

- La zone entre les symboles %{ et %} représente la **section des déclarations**.
- Dans cette section, on peut :
  - declarer des variables globales.
  - declarer des fonctions globales.
  - inclure des fichiers.
- Après le deuxième %%, c'est la section du **code utilisateur** : la fonction **main** et les autres fonctions.
- La règle :

\n nblignes- - ;

signifie : à la rencontre d'un caractère de saut de ligne "\n" dans le fichier source, incrémenter la variable nblignes.

Flex: Fast Lexer

Fonctionnement

- La fonction "main" contient deux instructions :
  - `yylex` : c'est une routine qui effectue l'appel explicite de l'analyseur lexical.
  - une instruction *printf* par laquelle on affiche le nombre de lignes calcule.

Flex: Fast Lexer

Format d'un fichier Flex

- Un fichier Flex contient trois sections :
  - La section des **déclarations** et des **définitions**.
  - La section des **règles**.
  - La section du **code utilisateur**.
- Chaque section est separée de la suivante par une ligne contenant juste les deux symboles %%.

Flex: Fast Lexer

Format d'un fichier Flex

Un fichier *Flex* se présente comme suit:

```
%{
/* Déclarations */
%}
/* Définitions */
%%
/* Règles */
%%
/* Code utilisateur */
```

Flex: Fast Lexer

Format d'un fichier Flex

- Une **définition** permet de **nommer** une expression régulière.
- Une définition est un couple formé d'un **identificateur** et d'une **expression régulière**, séparée par au moins un caractère espace (ou tabulation).
- Une expression régulière peut contenir une **référence** à un identificateur déjà défini. Dans ce cas, l'identificateur doit être écrit entre les symboles ” {” et ”}”
- Une définition doit être écrite sur une seule ligne.

Flex: Fast Lexer

Exemple:

- **CHIFFRE** [0 - 9]
- **ENTIER** (+|-)? {**CHIFFRE**}$^{+}$

Flex: Fast Lexer

- Une **règle** Flex est la donnée :
  d'une expression régulière **e**.
  et d'une action (celle qui sera exécutée lorsqu'une séquence d'entrée est reconnue par **e**).

  Exemple:
  "int" {printf(”Mot clé int”);}
  A chaque fois que la chaîne de caractères "int" sera reconnue, le message ”Mot clé int” sera affichée sur la sortie standard.

Flex: Fast Lexer

Les règles:

- Une expression régulière spécifie une unité lexicale du langage source.
- *Flex* utilise les **expressions régulières étendues**.
- Dans *Flex*, les expressions régulieres sont construites à partir des caractères et d'opérateurs.
- Les opérateurs de *Flex* sont :
  ” \ [] ~ - ? . * + | ()$ / {} % <>
- Pour utiliser ces opérateurs comme caractères ordinaires, il faut les protéger en les plaçant dans une chaîne entourée de double-quotes (”) ou en les plaçant après un caractère \.
- Par exemple, \n, \t et *b* correspondent comme en *C* au saut de ligne, à la tabulation et au retour en arrière.

Flex: Fast Lexer

Specification d'expressions régulières avec Flex:

- ” : la chaîne elle-même.
  ”abc” : specifie la chaîne abc
- [ ] : un des elements de l'ensemble.
  [abc] : a, b ou c
  [a - z] : toutes les lettres minuscules
- . : tout caractere sauf \n.
- | : l'alternance (l'union).
  (ab|bc) : la chaîne ab ou la chaîne bc.
- * : zero ou plusieurs fois (l'etoile).
  (x|y)* : 0 ou plusieurs caractères x ou y.
- + : un ou plusieurs fois (l'etoile positive).
  [a z]+ : 1 ou plusieurs lettres minuscules.

Flex: Fast Lexer

Specification d'expressions régulières avec Flex:

- ? : operateur d'occurrence 0 ou 1 fois. ab?c : la chaîne abc ou ac.
- / : condition de reconnaissance.
  ab/cd : la chaîne ab seulement si elle est suivie de la chaîne cd.
- $: reconnaissance en fin de ligne.
  ab$: la chaîne ab seulement si elle est en fin de ligne.
- ^ : reconnaissance en debut de ligne.
  ^ ab : la chaîne ab seulement si elle est en debut de ligne.
- { } : repetition bornée.
  a{1,5} : les chaînes a, aa, aaa, aaaa ou aaaaa.
  a{2,} : les chaînes aa, aaa, aaa, ...etc.

Flex: Fast Lexer

Résolution des conflits

- En cas de conflit, Flex choisit toujours la règle qui produit le plus long lexème.

Exemple:

- "prog" action1
- "program" action2

- La deuxième règle sera choisie en cas de conflit.

Flex: Fast Lexer

Résolution des conflits

- Si plusieurs règles donnent des lexèmes de mêmes longueurs, Flex choisit la première.

Exemple:

- "prog" action1
- [a — z]+ action2

La première règle sera choisie en cas de conflit de lexèmes ayant mêmes longueurs.

Flex: Fast Lexer

Résolution des conflits

- Si aucune règle ne correpond au flot d'entrée, Flex choisit sa règle par defaut implicite :

.|\n {ECHO }

- Cette règle par defaut, **recopie** le flot d'entrée sur le flot de sortie.

Flex: Fast Lexer

Les actions:

- Une **action** est un bloc d'instructions qui est exécutée lorsque la chaîne de caractères lue correspond à la chaîne spécifiée avant l'action.
- Flex dispose de quelques **variables** et **actions prédéfinies**.

Flex: Fast Lexer

Quelques variables de Flex:

- **yytext** : la chaîne reconnue par l'expression regulière (le lexème courant).
- **yyleng** : la longueur de yytext.
- **yyin** : le fichier d'entrée (c'est un **FILE***).
- **yyout** : le fichier de sortie (c'est un **FILE***).

Flex: Fast Lexer

Quelques macros de Flex:

- **ECHO** : recopie le lexème courant.
  [*a-z*]+ **ECHO**;
  ce qui éequivaut à {printf(”%s”, *yytext*); }
- **REJECT** : permet d’envisager la "deuxieme meilleure” règle pour laquelle le flot d'entréee (ou un de ses préfixes) est reconnu.
  foo {*f* (); **REJECT**; }
  [^\t\n]+ { --word_count; }
  ce qui permet de compter le nombre totat des caractères et exécuter la fonction *f*() chaque fois que la chaîne foo est rencontrée.

Flex: Fast Lexer

Quelques directives de Flex:

- **yymore**() : indique que lors de la prochaîne application d'une règle, la chaîne reconnue doit être cancaténée à *yytext* qui ne doit donc pas écraser *yytext*.

Exemple:

mega- { **ECHO***; yymore*(); }

octets **ECHO***;*

- *Pour l'entrée mega-octets, l'analyseur fournira mega-mega-octets.*

Flex: Fast Lexer

Quelques directives de Flex:

- **yyless**(n) : indique que les *n* premiers caratères de yytext doivent être pris en compte lors de la prochaine application d'une règle.

Exemple:

foobar { **ECHO***; yyles*(3)*;* }

[*a-z*]+ **ECHO***;*

Pour l'entrée foobar, l'analyseur fournira foobarbar.

Flex: Fast Lexer

Quelques directives de Flex

- **yymore**() : indique que lors de la prochaîne application d'une règle, la chaîne reconnue doit être cancaténée à *yytext* qui ne doit donc pas écraser *yytext*.

  Exemple:

  mega- { **ECHO***; yymore(); }* *octets* **ECHO***;*

  - *Pour l'entrée mega-octets, l'analyseur fournira mega-mega-octets.*

Flex: Fast Lexer

Quelques directives de Flex

- **yymore**() : indique que lors de la prochaîne application d'une règle, la chaîne reconnue doit être cancaténée à *yytext* qui ne doit donc pas écraser *yytext*.

  Exemple:

  mega- { **ECHO***; yymore(); }* *octets* **ECHO***;*

  - *Pour l'entrée mega-octets, l'analyseur fournira mega-mega-octets.*

Flex: Fast Lexer

Quelques directives de Flex

- **yymore**() : indique que lors de la prochaîne application d'une règle, la chaîne reconnue doit être cancaténée à *yytext* qui ne doit donc pas écraser *yytext*.

  Exemple:

  mega- { **ECHO***; yymore*()*;* } *octets* **ECHO***;*

  - *Pour l'entrée mega-octets, l'analyseur fournira mega-mega-octets.*