Формални Езици и Езикови Процесори
ТУ, кат. КС, летен семестър 2012

Лекция 9

Тема:

Компилатори на Компилатори /Програма Lex/

Съдържание:

- LEX – генератор на сканери
- Вход
- Регулярни изрази
- Действия
- Неоднозначни правила
- Примери

Как се строят лексически анализатори /сканери/?

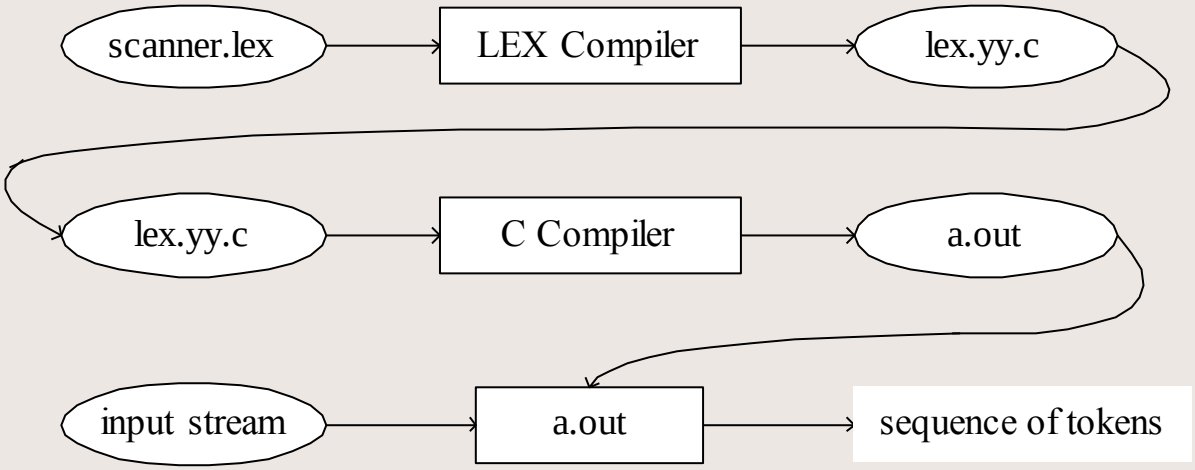
- Ръчно - според теорията на ЛА;
- Автоматизирано - utilities генерират първичния код на сканери съгласно описание на лексика във формат на РИ.

LEX е най-известният генератор на сканери (1975, Lesk & Schmidt, за UNIX). Използва се независимо или в комбинация с друга utility YACC (генератор на парсери). Двете програми са известни като компилатори на компилатори **compiler-compilers**.

Въведение в LEX

- LEX е генератор на лексически анализатори /сканери/.
- Нарича се още LEX компилатор:
  - Физическа схема на потока данни (диаграма);
  - Функционална и логическа схема (диаграма).

Физическа схема на потока данни



Функционална схема

Въвежда се описание на сканер в следния формат

Rule1 *{action1}*

Rule2 *{action2}*

Rule3 *{action3}*

Всяко правило $Rule_i$ е шаблон и всеки елемент $\{action_i\}$ е действие, което се изпълнява, ако лексема от входния поток бъде разпозната от шаблон $Rule_i$.

Шаблоните $Rule_i$ са регулярни изрази.

Действията $\{action_i\}$ са първични Си текстове.

Логическа схема

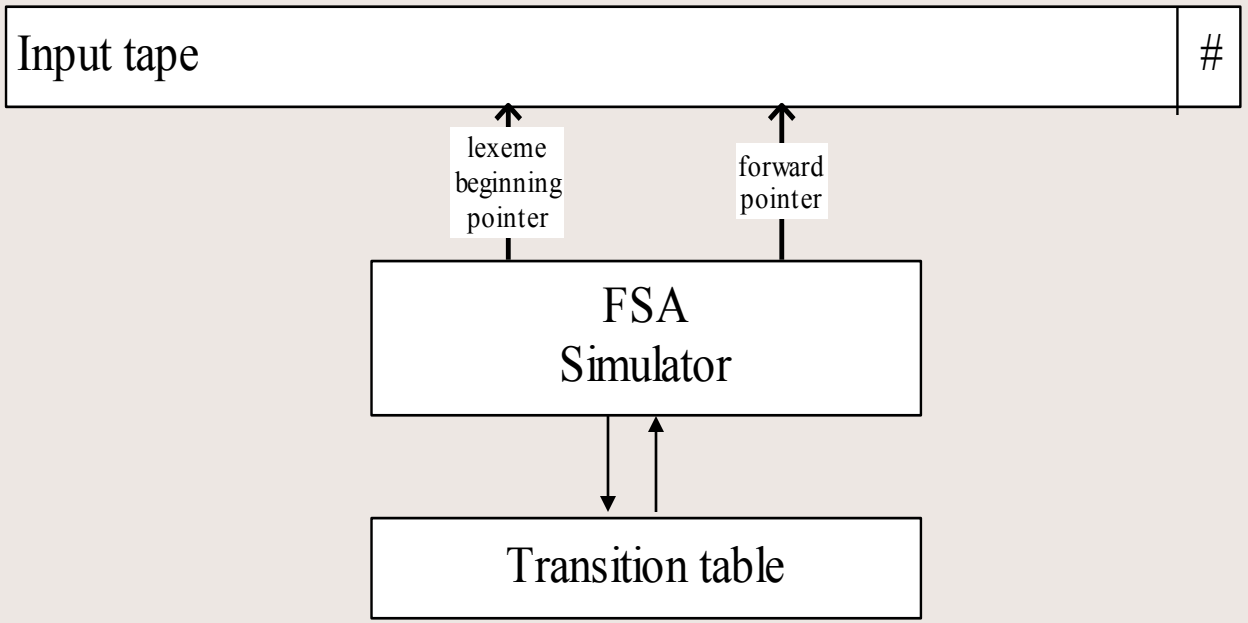
LEX compiler има за изход:

- Таблица на преходите на КА
- Функция `int yylex()`, използва таблицата на преходите за разпознаване на лексеми.

FSA/КА е естественият модел за създаване на сканери.

Такова е и поведението на продукта, генериран от LEX.

Функц./Лог. схема



Доп. пояснения

- LEX compiler генерира таблица на преходите за КА по данни от входа, описани като РИ.
- Генерираният сканер се състои от КА симулатор (Си функция *int yylex(void)*), която ползва таблицата на преходите за да търси и разпознава низове според РИ шаблоните.

LEX формат на входа

– LEX спецификация:

Definitions (declarations)	незадължителен
%%	задължителен
Translation rules	незадължителен
%%	незадължителен
Auxiliary (user defined) procedures	незадължителен

LEX формат на входа

- Абсолютният минимален вход за LEX:

%%

Без definitions, Без rules, Без user routines.

LEX създава програма, която единствено копира входните данни и ги изпраща на изхода без промяна.

С други думи, това е единственото действие, което винаги се изпълнява, ако не се намери съответствие по шаблон.

LEX translation rules

- Формат на LEX Translation Rules:

<Regular Expression> РИ	Поне 1 интервал	<action> действие
Описание на шаблон за разпознаване		Програмен код за изпълнение

РИ – Операции, м-ва символи /Character classes/, Optional изрази, Repeated изрази, Алтернация и Групиране.

LEX действия – единичен оператор или група оператори, обособени като блок /compound statement/.

Неоднозначни LEX правила.

Translation rules

Тази секция задава по-сложни обработки от просто копиране на входа в изхода. Най-общо LEX продуктът работи като програма филтър.

Извършва се филтриране на входа чрез търсене на символи или низове от символи, които се разпознават по специфицираните РИ, и се предизвиква изпълнение на съответни зададени действия.

Translation rules

Налице е разнообразие от формат на LEX РИ.

В най-прост вид РИ е низ. Този низ служи като шаблон за разпознаване на самия себе си.

orange	***;***
int	***printf("\n found keyword");***
colour	***printf("color");***
petrol	***gas***

Същинската цел и сила на РИ е да се опише шаблон за разпознаване на множество от символни низове, а не на отделен низ.

LEX РИ

- РИ разпознават множества от символни низове.
- РИ съдържат
  - Текстови символи /Text characters/. Буквите и цифрите са text characters.
  - Управляващи символи /Operator characters/ (задават повторение, избор и т.н.). Това са: “ **\ [ ] ^ - ? . * + | ( ) $ / { } %**

Примери за РИ в нотация LEX

- Регулярни Изрази с повторение /Repeated expressions/. Задават се с операция итерация:

a*

7*

a+

m+

Примери за РИ в нотация LEX

- За разпознаване на управляващи символи като текстови, те се предшестват от \ или се заграждат в “X“

abc\+ abc”+”

xyz”+*”

“xyz+*”

xyz\+*

[0-9]

\\

Примери за РИ в нотация LEX

- Character classes. Вътре в [ ] са валидни само следните три управляващи символи: \ - ^

[dgka]	[^dgka]
[abc]	[a-c]
[a-z]	[^a-z]
[a-zA-Z0-9*&#]	
[0-9]	[^0-9]
[abc]+	[0-9]+
[+\-]	[-+]

Примери за РИ в нотация LEX

- Character classes. Вътре в [ ] са валидни само следните три управляващи символи: \ - ^

[-+0-9]

[0-9+-]

[^abc] [^a-zA-Z]

\40 in octal

\x2a in hexadecimal

[\40-\176]

. Всички възможни символи с изключение на нов ред /new line/

Примери за РИ в нотация LEX

- Изрази, които се срещат 0 или веднъж /Optional expressions/

ab?c

x?

[+\-]?

[-+]?

Примери за РИ в нотация LEX

- Алтернация | и Групиране (…)

ab|cd

(ab|cd)

(abc)+

(ab|cd+)?(ef)*

(+|-)?

Примери за РИ в нотация LEX

- Контекстна зависимост

ab/cd

ab$

ab/\n

x$ ^x

[\t]+$

^[ ]

Примери за РИ в нотация LEX

- Повторения: { } фигурни скоби означава повторение, ако вътре се описват числа

a{1,5}

a{2,}

a{4}

Примери за РИ в нотация LEX

- Дефиниции: { } фигурни скоби означава разширение, ако вътре се описва име, което е зададено в първа секция на входа за LEX

```
digit  [0-9]
%%
{digit}+
```

РИ примери

[a-z]+

[a-zA-Z][a-zA-Z0-9]*

-[0-9]+

+?[0-9]+

-0\.[0-9]+

[0-9]+\.?

[0-9]*\.[0-9]+

LEX действия

- Translation rules във форма <*RE*> <*action*> са активни винаги.
- Едно действие винаги се изпълнява – LEX копира входа на изхода. Този copy process се изпълнява за всички входни низове, които остават неразпознати от зададените правила за превод.
- Указаните действия се изпълняват и заместват процеса копиране на входа в изхода.

Примери за LEX действия

```
[ \t\n]          ;
[a-z]+           printf(“A word accepted”);
[A-Za-z]+        printf(“A word %s accepted”,yytext);
[A-Za-z]+        ECHO;
[A-Za-z]+        { sbwords++;  sbletters += yyleng;}
“\n”             { linenum++; }
[Ee][Nn][Dd]          numend++;
```

Примери за LEX действия

```
rail[ ]+road        // Railroad is one word
rail[ \t\n]+road
G[A-Za-z]*          // word beginning with G
she             s++;
he              he++;
\n              ;
.               ;
```

Примери за LEX действия

```
she     {  s++;  REJECT;  }
 he     he++;
 \n     ;
 .      ;
```

Неднозначни LEX правила

- Когато входен низ се разпознава с повече от един шаблон (регулярен израз), LEX прави следния избор:
  - Когато два или повече шаблона сработват за успешно разпознаване, автоматът разпознавател избира най-дългата лексема. С други думи, дава се предпочитание на по-дългата (най-дългата) лексическа единица.

Пример:

- “<“ { return LT; }
- “<=“ { return LE; }

Неднозначни LEX правила

- Когато входен низ се разпознава с повече от един шаблон (регулярен израз), LEX прави следния избор:
  - Ако два или повече шаблона (РИ) разпознават най-дългата лексема, избира се лексемата, която се намира най-напред в списъка от правила. Дава се предпочитание на правилото, което се среща първо.

Пример:

- int { key word for integer }
- [a-z]+ { identifier }

LEX първични дефиниции

- Формат – разполагат се в началото преди първия %%:

```
%{
// препроцесорни директиви и дефиниции на глобални променливи и константи.
%}
// дефиниции – абревиатури
%%
```

Съкратени дефиниции

Предназначени са за LEX и се подават преди първия %% във входа:

<name> *<translation>*

А/ двете тела се разделят с поне 1 интервал

В/ *<name>* започва с буква

С/ <translation> съкратено се извиква в секция правила за превод, като името отляво се загражда с фигурни скоби като *{name}*.

Примери за абревиатури

- Дефиниция на буква, цифра и експонента

```
L       [a-zA-Z]
D       [0-9]
Exp     [EeDd][-+]?{D}+
%%
{L}({L}|{D})*              printf(“\nIdentifier”);
[a-zA-Z][a-zA-Z0-9]*       printf(“\nIdentifier”);
{D}+                       printf(”\nConstant”);
[0-9]+                printf(”\nConstant”);
```

Секция потреб. Деф. функции

- Пример

```
%%
[0-9]+          { myfunc(“integer”); }
%%
 void myfunc(char *p)
 {
    printf(“%s constant accepted:%s”, p, yytext);
 }
```

Работа с LEX

Изпълнение на сканер, генериран от LEX/FLEX с вградена функция *main()* взета от библиотеката LEX/FLEX.

```
flex  example1.lex
cc    lex.yy.c      -lfl
./a.out
```

Работа с LEX

Изпълнение на сканер, генериран от LEX/FLEX с вградена функция *main()*, съставена от потребителя като входни данни.

flex example2.lex

cc lex.yy.c

./a.out

Работа с LEX

Автоматизация с програма MAKE?

make -fmakefile1

Съдържание на входа makefile1 (2 правила):

```
a.out :        lex.yy.c
   cc  lex.yy.c
lex.yy.c       :       example1.lex
   flex  example1.lex
```

Работа с LEX

Автоматизация с програма MAKE?

make -fmakefile2

Съдържание на входа makefile2 (едно правило с две команди):

a.out : example1.lex

flex example1.lex

cc lex.yy.c

Допълнение LEX

- Какво още съдържа генерираният от LEX файл?
  - Таблица на преходите за детермииран КА
  - int yylex(),
  - char yytext[ ],
  - int yyleng,
  - yymore(),
  - yyless(n),
  - REJECT,
  - yywrap()

Демо програми

expr1.lex, expr1m.lex, expr2.lex,
expr3.lex, expr4.lex, expr5.lex,
expr6.lex, expr7.lex, expr8.lex,
expr9.lex, expr10.lex, expr11.lex,
ahoscan.lex,
scan1a.lex = lex1a.cpp,
scan1b.lex = lex1b.cpp.

Справочник: правила за LEX РИ

x	Явно задаване на текстов символ *x*
“x”	Явно задаване на символ *x*, текстов или служебен символ
\x	Явно задаване на символ *x*, текстов или служебен символ
[xy]	Символ *x* или символ *y*
[x-z]	Символ в диапазона от *x* до *z*, един от символите *x*, *y*, *z*
[^x]	Всеки символ с изключение на символ *x*
.	Всеки символ с изключение на символ нов ред (‘\n’)
^x	Символ *x* с фиксирана позиция в началото на ред
x$	Символ *x* с фиксирана позиция в края на ред
x?	Символ *x* или нищо, символ *x* срещнат 0 или 1 път

продължава →

Справочник: правила за LEX РИ

x*	Низ, съставен от 0 или повече символи *x* , ε, x, xx, xxx, ...,
x+	Низ, съставен от 1 или повече символи *x* , x, xx, xxx, ...,
x|y	Символ *x* или символ *y*
(x)	Символ *x*, означен като група (grouping)
x/y	Символ *x*, ако е следван от символ *y*
{x}	Преводът на символ *x* от секция дефиниции(декларации)
x{m,n}	Низ, съставен от символ *x*, повторен *m* до *n* пъти
x{m}	Низ, съставен от символ *x*, повторен *m* пъти
x{m,}	Низ, съставен от символ *x*, повторен *m* или повече пъти

Справочник: правила за LEX РИ

abc+ Множество низове *abc*, *abcc*, *abccc*, *abccc*, ...
abc\+ или abc"+" Низ *abc*+ единствено
(abc)+ Групиране: множество низове *abc*, *abcabc*, *abcabcabc*, ...
[abc] или [a–c] Един от символите *a*, *b*, *c*
[a–z] Един от символите *a*, *b*, *c*, ... *z*
[^a–z] Кой да е символ с изключение на символите *a*, *b*, *c*, ... *z*
[0–9] Кой да е цифров символ
[^0–9] Кой да е символ с изключение на цифровите
[abc]+ Непразен низ съставен от символите *a*, *b*, *c*
[0–9]+ Непразен низ, съставен от цифрови символи, константа без знак
ab/cd Низ *ab* само, ако е следван от низ *cd*
ab|cd Низ *ab* или низ *cd*
ab?c Низ *abc* или низ *ac*

Справочник: правила за LEX РИ

[+\–]?	Знак + или знак –, или нищо (празен низ)
или (+|–)?	
Или [–+]?	
[\t]+$	Серия от интервал и табулация до края на реда
^[ ]	Първият интервал в началото на реда
xyz"+*" xyz\+*	Низ *xyz+**
\\	Обратна наклонена черта като текстов символ
–[0–9]+	Отрицателна цяла константа
+?[0–9]+	Положителна (със знак или без знак) цяла константа
–0\.[0–9]+	Отрицателна реална константа *–0.D+*
[0–9]+\.?	Беззнакова цяла или реална константа без дробна част
[0–9]*\.[0–9]+	Реална константа *D*.D+*
[A–Za–z][A–Za–z0–9]*	Идентификатор *L(L|D)**

Благодаря
За
Вниманието