Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №1**

по дисциплине «Транслирующие системы»

«**Программирование лексического разбора на языке lex**»

Работу выполнил студент группы № 43501/3 Коренёк Григорий Андреевич

Работу принял преподаватель Цыган Владимир Николаевич

Санкт-Петербург

2018

**Цель работы**

Цель работы - изучение и получение навыков применения утилиты LEX для генерирования лексических анализаторов.

**Программа работы**

1. Ознакомиться с работой программы LEX.

2. Протестировать примеры.

3. Выполнить индивидуальное задание.

**Выполнение работы**

***Теоретическая информация:***

Lex - это генератор программ лексической обработки текстов. Основу исходной lex-программы составляет таблица регулярных выражений - "шаблонов'', и соответствующих им "действий'', которые задаются пользователем в виде фрагментов на языке C.

Общая форма исходного текста lex-программы:

*определения*

*%%*

*правила*

*%%*

*процедуры пользователя*

*Определения:*

Секция определений может содержать (в любой последовательности):

- макроопределения регулярных выражений, без отступов, в форме:

"name pattern"

{digit} [0-9]

- включаемый код на языке C, с отступом, в форме:

"code"

int count = 0 ;

- включаемый код на языке C, без отступов, в форме:

%{

code

%}

%{

#include <stdlib.h>

#define YY\_USER\_ACTION crap();

%}

- стартовые условия, без отступов, в форме:

%S cond1, cond2 ...

%S comment, newPage

- комментарии в стиле языка C, без отступов.

/\* macro, C-fragments, start conditions \*/

*Правила:*

Правила задаются без отступа, каждое в форме "шаблон действие".

Действие - это один оператор языка C; допускается составной оператор (т.е. последовательность простых операторов, заключенная в фигурные скобки), тогда действие может быть записано в нескольких строках.

Значения операторных литер:

"x" "x", даже если x - оператор

\x "x", даже если x - оператор

[xy] литера x или y

[x-z] литера в диапазоне от x до z

[^x] любая литера кроме x

. любая литера кроме новой строки

^x x в начале строки

<y>x x, если стартовое состояние y

x$ x в конце строки

x? необязательное x

x\* 0,1,2, ... экземпляров x

x+ 1,2,3, ... экземпляров x

x|y x или y

(x) x

x/y x, но только если за ним y

{xx} макроподстановка xx

x{m,n} от m до n появлений x

x{m,} m и более появлений x

x{m} m появлений x

Между начальным разделителем "%%'' и первым правилом может быть задан - с отступом - фрагмент на языке С. Он копируется в тот участок C-программы, который выполняется один раз после запуска; здесь также можно поместить определения локальных для yylex() переменных.

*Процедуры пользователя:*

Все, что следует после второго разделителя "%%'', переносится в C-программу без анализа и изменений; отступы роли не играют.

*Вспомогательные функции и макрокоманды:*

* yytext, yyleng

Входная последовательность, будучи распознана некоторым правилом, сохраняется в массиве yytext, а ее длина записывается в переменную yyleng.

Пользователь может изменять содержимое yytext в пределах первых yyleng позиций. В частности, последняя литера найденной строки доступна как yytext[yyleng-1].

* yymore()

Когда очередная лексема записывается в массив yytext, предыдущее содержимое yytext теряется. Функция yymore() временно отключает режим "перезаписи'' для следующего (одного) сопоставления, т.е. литеры следующей лексемы будут добавлены к текущему содержимому yytext.

* yyless(n)

Функция yyless(n) сокращает содержимое yytext до n первых литер, возвращая остальные во входной поток.

* input()

Чтение следующей литеры входного потока (в конце потока считывается null-литера);

* output(c)

Запись литеры "c'' в выходной поток;

* unput(c)

"запись'' литеры во входной поток.

* %start cond1, cond2, ...

Можно сократить "%start'' как "%s''. На условие в дальнейшем можно ссылаться в начале правил, задавая имя условия в угловых скобках. Такая запись как:

<cond>шаблон

означает, что правило распознается только тогда, когда текущее стартовое условие анализатора - cond. Для установки требуемого стартового условия (например, cond) используется макрокоманда:

BEGIN (cond);

(Скобки можно опустить.) Исходное (нулевое) стартовое условие восстанавливается следующим образом:

BEGIN (INITIAL);

Правило может быть активным при нескольких стартовых условиях, что задается префиксом в виде списка:

<cond1, ....., condN>

Правила без стартового условия активны всегда.

* Макроопределение YY\_USER\_ACTION, по умолчанию пустое, позволяет задать действие, которое выполняется перед действием любого правила.
* Макрокоманда YYSTATE возвращает целочисленное значение текущего стартового условия.
* макрокоманда REJECT

Выполняет следующие действия:

- возвращает принятую последовательность во входной поток;

- исключает правило, которым была распознана эта последовательность;

- возобновляет сопоставление.

**Выполнение примеров**

**1. Удаление пробелов и табуляций в начале строки**

%%

^[ \t]+ ;

%%

#ifndef yywrap

int yywrap() { return 1; }

#endif

main () { while (yylex()); }

Вход:

3 spaces

2 tabs

2 space + tab ;

the end

Выход:

3 spaces

2 tabs

2 space + tab ;

the end

**2. Подсчет числа строк**

Используем ключ –s, чтобы программа не выдавала на выход символы, не соответствующие ни одному шаблону.

int lineno = 0;

%%

\n lineno++;

. ;

%%

#ifndef yywrap

int yywrap() { return( 1 ); }

#endif

main()

{

while( yylex() );

printf( "%d lines\n", lineno );

}

Вход:

123513646

562346432672457

2355245

234525

141324234

Выход:

5 lines

**3. Подсчет и вывод знаковых целых чисел**

%{

int count = 0;

%}

%%

[-+]?[0-9]+ {

count++;

printf( "%d %s\n", count, yytext );

}

.|\n ;

%%

#ifndef yywrap

int yywrap() { return 1; }

#endif

main () { while (yylex()); }

Вход:

123as-123

as9876

-234-2345

Выход:

1 123

2 -123

3 9876

4 -234

5 -2345

**4. Вывод идентификаторов и беззнаковых целых чисел**

%%

[0-9]+ |

[a-zA-Z]+ { ECHO; printf( "\n" ); }

.|\n ;

%%

#include "yy.c"

Вход:

123as-123

as9876

-234-2345

Выход:

123

as

123

as

9876

234

2345

**5. Подсчет и вывод гистограммы длин слов**

int len[40], i;

%%

{

for( i = 0; i < 40; i++ )

len[i] = 0;

}

[a-z]+ len[yyleng]++;

.|\n ;

%%

#ifndef yywrap

int yywrap() { return 1; }

#endif

main()

{

while( yylex() );

for( i = 0; i < 40; i++ )

if( len[i] > 0 )

printf( "%5d%10d\n", i, len[i] );

}

Вход:

a

as

s

asd

as

s

asd

asdf

asd

as

as

Выход:

1 2

2 4

3 3

4 1

Контрольный вопрос: каким должен быть шаблон для выявления пробелов и табуляций в конце строки?

%%

[0-9]+([ \t]+)$ { ECHO; printf( "\n" ); }

.|\n ;

%%

#include "yy.c"

Вход:

1111

1111

1111

1111

1111

1111

1111

Выход:

1111

1111

1111

1111

1111

1111

1111

**6. Вывод строки наискосок при помощи yyless**

%%

(.)+ {

printf(">%s\n", yytext);

if (yyleng > 1) yyless(yyleng/2);

}

%%

Вход:

1234567

Выход:

>1234567

>4567

>67

>7

**7.1. Поиск конца комментария, заданного в стиле языка C**

%{

void skip\_comments();

%}

D [0-9]

H [0-9A-Fa-f]

L [\_A-Za-z]

%%

{L}({L}|{D})\* printf( "ident: %s\n", yytext );

0{H}+(H|h)? |

{D}{H}\*(H|h) printf( "hex: %s\n", yytext );

{D}+ printf( "decimal: %s\n", yytext );

"/\*" skip\_comments();

. ;

%%

void skip\_comments()

{

int c = '\*'; /\* not char! \*/

while( c != '/' ) {

while( input() != '\*' );

c = input();

if( c != '/' )

unput (c);

}

}

#include "yy.c"

Вход:

111/\*22

3333

4444444

55\*/6666

777

Выход:

decimal: 111

decimal: 6666

decimal: 777

В примере нет проверки конца входного потока, так что незакрытый комментарий приводит к зацикливанию в процедуре skip\_comments.

**8.1. Правильное решение, с проверкой завершения входного потока:**

%{

void skip\_comments();

%}

D [0-9]

H [0-9A-Fa-f]

L [\_A-Za-z]

%%

{L}({L}|{D})\* printf( "ident: %s\n", yytext );

0{H}+(H|h)? |

{D}{H}\*(H|h) printf( "hex: %s\n", yytext );

{D}+ printf( "decimal: %s\n", yytext );

"/\*" skip\_comments();

. ;

%%

void skip\_comments()

{

int c = '\*'; /\* not char! \*/

do {

while ((c = input()) != '\*' && c != EOF) ;

while ((c = input()) == '\*') ;

} while (c != '/' && c != EOF);

if (c == EOF) {

fprintf(stderr, "?-EOF in comment\n");

exit(1);

}

}

#include "yy.c"

8. Функция unput: реверсирование идентификаторов, начинающихся с '@':

int i, len;

char \*p;

%%

\@[A-Za-z]+ {

len = yyleng;

p = (char \*)strdup(yytext);

for( i = 1; i < len; i++ )

unput( p[i] );

}

%%

#include "yy.c"

Вход:

asd @asd asd asd

@asd@asd asd @asd@

Выход:

asd dsa asd asd

dsadsa asd dsa@

**9. Двусмысленный набор правил**

%%

read { printf( "operation: " ); ECHO; }

[a-z]+ { printf( "identifier: " ); ECHO; }

%%

#include "yy.c"

Вход:

ready

Выход:

identifier: ready

Вход:

read

Выход:

operation: read

**10. Неправильный шаблон для распознавания строки в кавычках**

%%

'.\*' ;

%%

#include "yy.c"

Вход:

'first' here, 'second' there

Выход:

'first' here, 'second'

**11. Правильный шаблон для распознавания строки в кавычках**

%%

'[^'\n]\*' ;

%%

#include "yy.c"

Вход:

'first' here, 'second' there

Выход:

here, there

**12. Использование переменной состояния**

int state;

%%

^1 { state = 1; ECHO; }

^2 { state = 2; ECHO; }

^3 { state = 3; ECHO; }

\n { state = 0; ECHO; }

magic { switch (state) {

case 1: printf("<first>"); break;

case 2: printf("<second>"); break;

case 3: printf("<third>"); break;

default : ECHO;

}

}

%%

#include "yy.c"

Вход:

1 asd magic

2 magic asd

3 asd magic asd

Выход:

1 asd <first>

2 <second> asd

3 asd <third> asd

Вход:

1 asd magic

2 magic asd

3 asd magic asd

Выход:

1 asd <first>

2 <second> asd

3 asd <third> asd

**13.1. Решение той же задачи при помощи стартовых условий**

%START c1 c2 c3

%%

^1 { ECHO; BEGIN c1; }

^2 { ECHO; BEGIN c2; }

^3 { ECHO; BEGIN c3; }

\n { ECHO; BEGIN 0; }

<c1>magic printf( "<first>" );

<c2>magic printf( "<second>" );

<c3>magic printf( "<third>" );

%%

#include "yy.c"

**13.2. Трассировка стартовых условий**

%START c1 c2 c3

%{

#define YY\_USER\_ACTION { fprintf(stderr, "<%d>", YYSTATE); }

%}

%%

^1 { ECHO; BEGIN c1; }

^2 { ECHO; BEGIN c2; }

^3 { ECHO; BEGIN c3; }

\n { ECHO; BEGIN 0; }

<c1>magic printf( "<first>" );

<c2>magic printf( "<second>" );

<c3>magic printf( "<third>" );

%%

#include "yy.c"

Вход:

1 asd magic

2 magic asd

3 asd magic asd

Выход:

<0><1><1><1><1><1><1><1>1 asd <first>

<0><2><2><2><2><2><2><2>2 <second> asd

<0><3><3><3><3><3><3><3><3><3><3><3>3 asd <third> asd

**14.1. Подсчет количества she и he без учета he внутри she**

int s = 0, h = 0;

%%

she s++;

he h++;

.|\n ;

%%

#ifndef yywrap

int yywrap() { return( 1 ); }

#endif

main()

{

while( yylex() );

printf( "she: %d times, he: %d times\n", s, h );

}

Вход:

he she he she she he he she he

Выход:

she: 4 times, he: 5 times

**14.2. Подсчет всех экземпляров she и he**

int s = 0, h = 0;

%%

she { s++; REJECT; }

he { h++; REJECT; }

.|\n ;

%%

#ifndef yywrap

int yywrap() { return( 1 ); }

#endif

main()

{

while( yylex() );

printf( "she: %d times, he: %d times\n", s, h );

}

Вход:

he she he she she he he she he

Выход:

she: 4 times, he: 9 times

**14.3. Подсчет she и he с использованием yyless**

int s = 0, h = 0;

%%

she { s++; yyless(1); }

he { h++; }

.|\n ;

%%

#ifndef yywrap

int yywrap() { return 1; }

#endif

main ()

{

while (yylex());

printf("she: %d times, he: %d times\n", s, h);

}

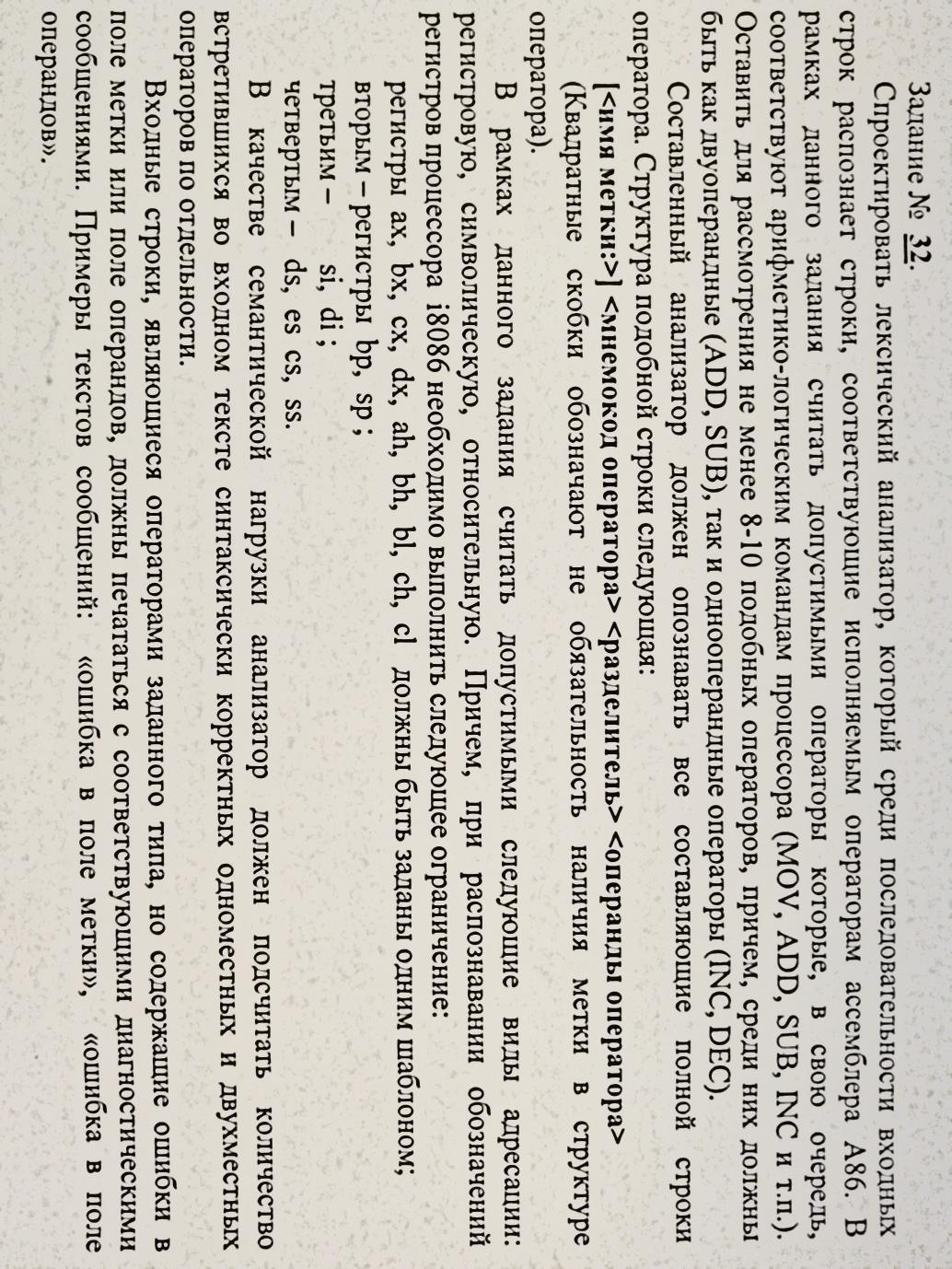
Вход:

he she he she she he he she he

Выход:

she: 4 times, he: 9 times

**Индивидуальное задание**



lbl [a-zA-Z]+[1-9]\*:

cmd1 (inc)|(dec)

cmd2 (mov)|(add)|(sub)

dbg {cmd2}

reg1 (ax)|(bx)|(cx)|(dx)|(ah)|(bh)|(bl)|(ch)|(cl)

reg2 (bp)|(sp)

reg3 (si)|(di)

reg4 (ds)|(es)|(cs)|(ss)

reg {reg1}|{reg2}|{reg3}|{reg4}

id [a-zA-Z]+[1-9]\*

addr {reg4}"["{reg3}"]"

int cmd1\_count = 0;

int cmd2\_count = 0;

void err\_lb(char\* str) {

if (str[strlen(str)-1] == '\n') {

printf("label error: '%s'\n", str);

}

}

void err\_op(char\* str) {

printf("operator error: '%s'\n", str);

}

%%

^(({lbl}[ \t])?{cmd1}[ \t]({reg}|{id}|{addr})[\n\0]) cmd1\_count = cmd1\_count + 1;

^(({lbl}[ \t])?{cmd2}[ \t]({reg}|{id}|{addr})[ \t]({reg}|{id}|{addr})[\n\0]) cmd2\_count = cmd2\_count + 1;

^((.)+[ \t]{cmd1}[ \t]({reg}|{id}|{addr})[\n\0]) err\_lb(yytext);

^((.)+[ \t]{cmd2}[ \t]({reg}|{id}|{addr})[ \t]({reg}|{id}|{addr})[\n\0]) err\_lb(yytext);

^(({lbl}[ \t])?({cmd1}|{cmd2})[ \t](.)+([ \t](.))?[\n\0]) err\_op(yytext);

.|\n ;

%%

#ifndef yywrap

int yywrap() { return 1; }

#endif

main () {

while (yylex());

printf("1-op commands: %i\n2-op commands: %i\n", cmd1\_count, cmd2\_count);

}

Вход:

m1: mov ax bx

1m: mov ax bx

label123: inc asd

sub ds[si] op1

add es[si] op2

mov es[si]

Выход:

label error: '1m: mov ax bx

'

operator error: 'mov es[si]

'

1-op commands: 1

2-op commands: 3

**Выводы**

В ходе работы были рассмотрены основные принципы работы с программой LEX. На примерах рассмотрена структура и синтаксис LEX-программы. Полученные знания были обобщены при работе над индивидуальным заданием. При написании программы индивидуального задания были использованы макроопределения регулярных выражений для большей компактности и лучшей читаемости текста программы. Для вывода отладочных сообщений использовался массив yytext.