Санкт-Петербургский политехнический университет Петра Великого

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №1**

**Дисциплина**: Транслирующие системы

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

Выполнил студент гр. 3530901/70203 И.Д. Иванов

(подпись)

Преподаватель В.Н. Цыган (подпись)

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

Санкт-Петербург

2019

1. **Цели работы**

Изучение и получение навыков применения утилиты *LEX* для генерирования лексических анализаторов.

1. **Выполнение работы**

По каждому пункту работы представлен текст программы, тестовые данные и результаты выполнения программы.

2.1. Удаление пробелов и табуляций в начале строк

Рассмотрим программу, которая передает в выходной поток все литеры входного потока кроме пробелов и/или табуляций в начале строки.

Текст программы:

%%

^[ \t]+ ;

%%

#ifndef yywrap

int yywrap() {return 1; }

#endif

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

Была выполнена трансляция программы командой make ex1, а также её тестирование.

Тестовые данные:

fjvndfv

asdfasdfasdf

jkn

jndfkj sasAA

AA AAA

Результат выполнения программы:

fjvndfv

asdfasdfasdf

jkn

jndfkj sasAAScC

AA AAA

Пробелы и табуляции в начале строки были удалены, как и требовалось.

2.2. Пример секции определений

Общая структура исходного текста *lex*-программы:

определения

%%

правила

%%

процедуры пользователя

Обязательной является только секция правил, которая ограничивается парой разделителей «%%» (даже при отсутствии других секций).

Секция определений может содержать в любой последовательности: макроопределения регулярных выражений, без отступов, включаемый код на языке C, с отступом, включаемый код на языке C, без отступов, стартовые условия, без отступов, комментарии в стиле языка C. Пример секции определений:

{digit} [0-9]

int count = 0;

%{

#include <stdlib.h>

#define YY\_USER\_ACTION trace();

void skip\_comments();

%}

%S quotes, newPage

/\* macro, code, code, start conditions, comment\*/

2.3. Подсчет числа строк

Текст программы:

int lineno= 0;

%%

\n lineno++;

. ;

%%

#ifndef yywrap

int yywrap() { return(1);}

#endif

main()

{

while( yylex() );

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

}

Тестовые данные:

1

2

3

1234

ASDF

/

.

аaa

Результат выполнения программы:

11 lines

Результат полностью соответствует ожидаемому.

2.4. Подсчет и вывод знаковых целых чисел

Текст программы:

%{

int count = 0;

%}

%%

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

count++;

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

}

%%

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

-2

12

+123

asas

ss

Результат выполнения программы:

1 -2

2 12

3 +123

asas

ss

Каждый раз при обнаружении последовательности, обозначающей знаковое целое число, выводится текущее значение счетчика и текст лексемы.

2.5. Вывод идентификаторов и беззнаковых целых чисел

Текст программы:

%%

[0-9]+ |

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

.|\n ;

%%

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

sdasd\_\_

1212

\_fasdfa

\*1321

1212-12

Результат выполнения программы:

sdasd

1212

fasdfa

1321

1212

12

2.6. Подсчет и вывод гистограммы длин слов

Текст программы:

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] );

}

Тестовые данные:

aaaaaa

asdf

a

a

a

a

Результат выполнения программы:

1 4

4 1

6 1

Вывод программы осуществлен в виде гистограммы. 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*. Функция *yyless*(*n)* сокращает содержимое *yytext* до n первых литер, возвращая остальные во входной поток.

**Контрольный вопрос:**

9) цепочки букв латинского алфавита кроме гласных

Текст программы:

%%

.\*[aeiouyAEIPYY].\* ;

[b-df-hj-np-tv-xzB-DF-HJ-NP-TV-XZ]+ { ECHO; printf("\n");}

.|\n ;

%%

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

sdfgh

fdfdfa

wrt

.

123

ksaret

tpppq

Результат выполнения программы:

sdfgh

wrt

tpppq

Программа выводит строки, содержащие цепочки букв латинского алфавита кроме гласных. Можно сделать вывод о том, что программа работает корректно.

2.7. Вывод строки наискосок при помощи yyless

Текст программы:

%%

(.)+ {

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

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

}

%%

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

asdfgh

12345678

Результат выполнения программы:

>asdfgh

>fgh

>gh

>h

>12345678

>5678

>78

>8

Достигается этот вывод благодаря тому, что *yyless(n)* сокращает строку в *yytext* доnпервых литер,возвращаяостаток во входной поток.

2.8. Макросы и ввод-вывод низкого уровня

Текст программы:

%{

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);

}

}

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

ARTY\\..

asdfasdf

54361

/\*asnlkdanca

dvsdv dvldvmm ;scd;sm

\*/

2Dh

Результат выполнения программы:

ident: ARTY

ident: asdfasdf

decimal: 54361

hex: 2Dh

В рассмотренном примере определены правила для распознавания имен и чисел (десятичных и 16-ричных в стиле ассемблера а86). Для сокращения записи этих правил в разделе определений заданы макроопределения шаблонов, обозначающих буквы, десятичные и 16-ричные цифры; подстановки заданы именами макрокоманд в фигурных скобках.

2.9. Проверка конца входного потока при использовании input

Текст программы:

%{

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 != 0) ;

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

} while (c != '/' && c != 0);

if (c == 0) {

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

exit(1);

}

}

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

ARTY\\..

asdfasdf

/\*dhdhdhdh\*\*\*/

54361

/\*asnlkdanca

dvsdv dvldvmm ;scd;sm

\*/

2Dh

/\* this will not break the programm

Результат выполнения программы:

ident: ARTY

ident: asdfasdf

decimal: 54361

hex: 2Dh

Также, в стандартном потоке ошибок появилось сообщение, говорящее о том, что во время комментария достигнут конец файла:



Функция *input* используется для поиска конца комментария, заданного в стиле языка C — */\* \*/.* Также демонстрируются макроопределения 16-ричных цифр H, десятичных цифр D и букв L и их подстановки: {H}, {D} и {L}. Функция *unput* в *skip\_comments* предназначена для обработки частного случая */\*?\*\*/* (подряд более одной '\*' перед '/'). Была добавлена проверка конца входного потока, так что незакрытый комментарий больше не приводит к зацикливанию в процедуре *skip\_comments*.

2.10. Функция unput

Текст программы:

int i;

%%

\@[A-Za-z]+ {

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

unput( yytext[i] );

}

%%

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

@abcdbbbb

ccccddddaaaa

Результат выполнения программы:

abcddcba

ccccddddaaaa

Если представить себе входной поток в виде магнитофонной ленты, то функция *input* считывает ее при воспроизведении, а *unput* — это запись на

перемотке в начало. Задано реверсирование идентификаторов, начинающихся с '@'. Этот пример годится не для всех реализаций *lex*. Возможно, что функция *unput* будет изменять величину *yyleng* (уменьшать на 1) и содержимое *yytext* (удалять крайнюю литеру), что вполне логично. Поэтому лучше использовать копию *yytext* и *yyleng*.

2.11. Дублирование yytext и yyleng при работе с unput Текст программы:

int i, len;

char \*p;

%%

\@[A-Za-z]+ {

len = yyleng;

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

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

unput( p[i] );

}

%%

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

@abcdHHHH

abcdHHHHH

@123312

@AAbbccD

Результат выполнения программы:

HHHHdcba

abcdHHHHH

@123312

DdccbbAA

2.12. Двусмысленный набор правил

Текст программы:

%%

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

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

%%

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

read

re

rea

ready

asdfasdf

Результат выполнения программы:

operation: read

identifier: re

identifier: rea

identifier: ready

identifier: asdfasdf

Если при поиске лексемы входная последовательность может быть распознана несколькими шаблонами, то набор правил двусмысленный. В этой ситуации правило выбирается по следующей схеме:

1. Предпочтение отдается соответствию большей длины;

2. Если одна и та же последовательность соответствует нескольким правилам, предпочтение отдается тому правилу, которое задано раньше других.

2.13. Неправильный шаблон для распознавания строки в кавычках

Текст программы:

%%

'.\*' ECHO;

. ;

%%

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

'first', and there 'second' adadad

Hmm, it 'does' not 'work'

Результат выполнения программы:

'first', and there 'second'

'does' not 'work'

2.14. Правильный шаблон для распознавания строки в кавычках

Текст программы:

%%

'[^'\n]\*' ECHO;

. ;

%%

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

'first', and there 'second' adadad

And 'this' one 'is' 'working'

Результат выполнения программы:

'first''second'

'this''is''working'

Правильное решение формулируется так: между кавычками могут быть любые литеры кроме кавычки и конца строки.

2.15. Использование переменной состояния

Текст программы:

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;

}

}

%%

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

1 magic a

2 magic b

3 magic c

4 magic d

2 magic 1

1 magic aaaa

Результат выполнения программы:

1 <first> a

2 <second> b

3 <third> c

4 magic d

2 <second> 1

1 <first> aaaa

2.16. Решение при помощи стартовых условий

Текст программы:

%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>" );

%%

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

1 magic a

2 magic b

3 magic c

4 magic d

2 magic 1

1 magic aaaa

Результат выполнения программы:

1 <first> a

2 <second> b

3 <third> c

4 magic d

2 <second> 1

1 <first> aaaa

2.17. Трассировка стартовых условий

Текст программы:

%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>" );

%%

#ifndef yywrap

int yywrap() { return(1);}

#endif

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

Тестовые данные:

1 magic a

2 magic b

3 magic c

4 magic d

2 magic 1

1 magic aaaa

Результат выполнения программы:

1 <first> a

2 <second> b

3 <third> c

4 magic d

2 <second> 1

1 <first> aaaa

Также, в стандартном потоке ошибок появились сообщения, говорящие о том, какое стартовое условие имело место быть в тот или иной момент:



На уровне реализации стартовые условия — это целые числа (в частности, *INITIAL* = 0).Это обстоятельство позволяет проводить трассировкустартовых условий.

Макроопределение *YY\_USER\_ACTION*, по умолчанию пустое, позволяет задать код, который выполняется перед действием любого правила. Макрокоманда *YYSTATE* возвращает численное значение текущего стартового условия.

2.18. Подсчет количества she и he без учета he внутри she

Текст программы:

int s = 0, h = 0;  
  
%%  
she s++;  
he h++;  
.|\n ;  
%%  
  
#ifndef yywrap  
int yywrap() { return( 1 ); }  
[#endif](https://vk.com/im?sel=22702508&st=%23endif)  
  
main()  
{  
while( yylex() );  
printf( "she: %d times, he: %d times\n", s, h );  
}

Тестовые данные:

he

she

he she

shesheshe

hehehe

she she he

heheshe

aaaa

Результат выполнения программы:

she: 8 times, he: 8 times

Эта программа не выявляет экземпляры "*he''* внутри "*she''*, т. к. после распознавания "*she''* эти литеры уходят из входной последовательности.

2.19. Подсчет всех экземпляров she и he

Текст программы:

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 );

}

Тестовые данные:

he

she

he she

shesheshe

hehehe

she she he

heheshe

aaaa

Результат выполнения программы:

she: 8 times, he: 16 times

При обнаружении "*she''* увеличивается счетчик s, команда *REJECT* отвергает правило и возвращает "*she''* на вход. Затем предпринимается попытка заново сопоставить тот же ввод с оставшимися шаблонами. В этом примере можно учесть то, что "*she''* включает в себя "*he''*, но не наоборот, и убрать *REJECT* из второго действия. Но когда в шаблонах задано повторение, невозможно предугадать, сколько литер каким правилом будет распознано.

2.20. Подсчет she и he с использованием yyless

Текст программы:

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);

}

Тестовые данные:

he

she

he she

shesheshe

hehehe

she she he

heheshe

aaaa

Результат выполнения программы:

she: 8 times, he: 16 times

**3. Выводы**

В ходе выполнения лабораторной работы были получены навыки использования утилиты для построения лексических анализаторов (задание функционала анализаторов происходит с помощью языка С). Также, были получены навыки использования макрокоманд *ECHO*, *REJECT* и других, избавляющих разработчика от необходимости составления дополнительных алгоритмов, что, в свою очередь, уменьшает размер исходного кода и делает его легко читаемым.