Санкт-Петербургский политехнический университет Петра Великого
Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе

Дисциплина: Транслирующие системы

Тема: Программирование лексического разбора на языке *lex*

Выполнил студент гр. 43501/3 Мальцев М.С.
Преподаватель Цыган В.Н.

Санкт-Петербург
22 марта 2019 г.

1 Цель работы

- Познакомиться с генератором программ лексической обработки текстов Lex.
- Выполнить трансляцию предложенных программ на языке Lex. Протестировать их работу.

2 Ход работы

2.1 Удаление пробелов и табуляций в начале строк

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

```
%%
^[ \t]+     ;
%%

#ifndef yywrap
int yywrap() { return 1; }
#endif

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

Тестовые данные, подаваемые на вход, приведены ниже:

```
   3 spaces
        2 tabs
    2 space + tab           ;
the end
```

В результате получили:

```
3 spaces
2 tabs
2 space + tab           ;
the end
```

Программа отработала корректно. Пробелы и символы табуляции были заменены.

2.2 Подсчет числа строк

Программа подсчитывающая количество строк:

```
    int lineno = 0;
%%
\n  lineno++;
.   ;
%%

#ifndef yywrap
int yywrap() { return( 1 ); }
```

```
#endif

main()
{
    while( yylex() );
    printf( "%d lines\n", lineno );
}
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
1 string
string 2
3 string
string 4
```

В результате получили:

```
4 lines
```

Программа отработала корректно. Количество строк было подсчитано верно.

2.3 Подсчет и вывод знаковых целых чисел

Программа подсчитывающая и выводящая количество знаковых чисел:

```
%{
int count = 0;
%}

%%
[-+]?[0-9]+   {
                  count++;
                  printf( "%d %s\n", count, yytext );
              }
%%

#include "yy.c"
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
fkgnsdfngsd ladsmfka s d sdf kas      efgdf123
+123 dfg -987 sdfg sdf --456
456
123
sdfsd
```

В результате получили:

```
fkgnsdfngsd ladsmfka s d sdf kas      efgdf1 123

2 +123
 dfg 3 -987
 sdfg sdf -4 -456

5 456

6 123

sdfsd
```

Программа отработала корректно. Видно, что при обнаружении целого числа программа добавляет к нему индекс.

2.4 Вывод идентификаторов и беззнаковых целых чисел

Используемая программа:

```
%%
[0-9]+      |
[a-zA-Z]+   { ECHO; printf( "\n" ); }
.|\n        ;
%%

#include "yy.c"
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
jfdshf asdjewh23 123+ + 45324 +3425 -rg455 -x3 +4
231  -3 --3 x2 x mk3 20 4
```

В результате получили:

```
jfdshf
asdjewh
23
123
45324
3425
rg
455
x
3
4
231
3
3
x
2
x
mk
3
20
4
```

Программа отработала корректно. Все идентификаторы и беззнаковые целые числа были отдельно выведены.

2.5 Подсчет и вывод гистограммы длин слов

Текст используемой программы:

```
    int len[40], i;

%%
        {
          for( i = 0; i < 40; i++ )
            len[i] = 0;
```

```
7           }
8   [a-z]+  len[yyleng]++;
9   .|\n    ;
10  %%
11
12  #ifndef yywrap
13  int yywrap() { return 1; }
14  #endif
15
16  main()
17  {
18      while( yylex() );
19      for( i = 0; i < 40; i++ )
20          if( len[i] > 0 )
21              printf( "%5d%10d\n", i, len[i] );
22  }
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
1  df g qg gtr gg allin yoloyolo
2  metro Exodus RTX
```

В результате получили:

```
1      1          1
2      2          3
3      3          1
4      5          3
5      8          1
```

Программа отработала корректно. Была выведенна длинна слов и их количество.

2.6 Дополнительное задание

Цепочки из 0 и 1 не больше, чем 2 единицы подряд.

Разработанная программа:

```
1   int is_true = 1;
2
3   %%
4   (1){3} { printf("11\n1");}
5   \n ;
6   %%
7
8   #ifndef yywrap
9   int yywrap() { return( 1 ); }
10  #endif
11
12  main() { while( yylex() ); }
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
1  000010110111001100010
2  111000111000111100110
```

В результате получили:

```
00001011011
100110001011
100011
100011
1100110
```

Программа отработала корректно.

2.7 Вывод строки наискосок при помощи yyless

Текст используемой программы:

```
%%
(.)+    {
            printf(">%s\n", yytext);
            if (yyleng > 1) yyless(yyleng/2);
        }
%%

/*
%%
(.)+        {
            printf(">%s\n", yytext);
            if (yyleng > 1) yyless(yyleng/2);
        }
(.|\n)      ;
%%
*/

#include "yy.c"
```

- Тест 1

  Тестовые данные, подаваемые на вход, приведены ниже:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
```

  В результате получили:

```
>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
>consectetur adipiscing elit.
>ipiscing elit.
>g elit.
>lit.
>t.
>.
```

- Тест 2

  Тестовые данные, подаваемые на вход, приведены ниже:

```
Nunc consectetur diam neque, at congue lectus congue euismod. Quisque
    faucibus non arcu eu lobortis. Aliquam mollis nisl nisl, ac pretium
    velit commodo in.
```

  В результате получили:

```
>Nunc consectetur diam neque, at congue lectus congue euismod. Quisque faucibus non arcu eu lobortis. Aliquam mollis nisl nisl, ac pretium velit commodo in.
> non arcu eu lobortis. Aliquam mollis nisl nisl, ac pretium velit commodo in.
>isl nisl, ac pretium velit commodo in.
>m velit commodo in.
>mmodo in.
> in.
>n.
>.
>
```

Программа отработала корректно.

2.8 Макросы и ввод-вывод низкого уровня

Программа, приведенная ниже, идентифицирует буквы и числа в 10-м и 16-м фарматах. Программа игнорирует комментарии.

```
%{
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"
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
int main()
{
    unsigned int n;
    unsigned long long factorial = 1;
```

```
5
6       /*
7       unsigned int x;
8       factorial
9       **/
10
11      int hexVar = 7Ah;
12
13      return 0;
14  }
```

В результате получили:

```
1  ident: int
2  ident: main
3
4
5  ident: unsigned
6  ident: int
7  ident: n
8
9  ident: unsigned
10 ident: long
11 ident: long
12 ident: factorial
13 decimal: 1
14
15
16
17
18 ident: int
19 ident: hexVar
20 hex: 7Ah
21
22
23 ident: return
24 decimal: 0
```

Программа отработала корректно.

2.9 Проверка конца входного потока при использовании input

Настоящая программа, является дополнением предыдущей. В качестве нововведения добавлена проверка конца входного потока при использо-вании input. Проблема предшествующей программа заключалась в том, что происходило зацикливание при незакрытом комментарии. Код модифицированной программы представлен ниже:

```
1  %{
2  void skip_comments();
3  %}
4
5  D    [0-9]
6  H    [0-9A-Fa-f]
7  L    [_A-Za-z]
8
```

```
%%
{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"
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
int main()
{
    unsigned int n;
    unsigned long long factorial = 1;

    /*
    unsigned int x;
    factorial
    **/

    int hexVar = 7Ah;
    return 0;

    /*
        some text
}
```

В результате получили:

```
ident: int
ident: main


ident: unsigned
ident: int
ident: n

ident: unsigned
ident: long
ident: long
ident: factorial
decimal: 1


```

```
16
17
18 ident: int
19 ident: hexVar
20 hex: 7Ah
21
22 ident: return
23 decimal: 0
```

Программа отработала корректно.

2.10 Функция unput

Приведенная ниже программа производит реверсирование идентификаторов, начинающихся с символа @:

```
1     int   i, len;
2     char  *p;
3
4 %%
5 \@[A-Za-z]+     {
6                   len = yyleng;
7                   p = (char *)strdup(yytext);
8
9                   for( i = 1; i < len; i++ )
10                      unput( p[i] );
11                  }
12 %%
13
14 #include "yy.c"
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
1 But I @must
2 @
3 explain @to you how
4 all this @mistaken idea
5 @of denouncing pleasure
```

В результате получили:

```
1 But I tsum
2 @
3 explain ot you how
4 all this nekatsim idea
5 fo denouncing pleasure
```

Программа отработала корректно.

2.11 Двусмысленный набор правил

Приведенная ниже программа демонстрирует выбор правила при двусмысленном наборе.

```
1 %%
2 read    { printf( "operation: " ); ECHO; }
3 [a-z]+  { printf( "identifier: " ); ECHO; }
```

```
4 %%
5
6 #include "yy.c"
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
1 reader
2 ready
3 read
4 forread
5 red
```

В результате получили:

```
1 identifier: reader
2 identifier: ready
3 operation: read
4 identifier: forread
5 identifier: red
```

Программа отработала корректно.

2.12 Неправильный шаблон для распознавания строки в кавычках

Ниже приведена программа, которая демонстрирует неверное определение шаблона, в задаче выявления строк, не заключенных в кавычки.

```
1 %%
2 '.*'    ;
3 %%
4
5 #include "yy.c"
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
1 At 'vero' eos et 'accusamus' et
2 iusto 'odio' dignissimos
3 earum ‘rerum hic tenetur
4 maxime placeat facere‘ possimus
```

В результате получили:

```
1 At  et
2 iusto  dignissimos
3 earum ‘rerum hic tenetur
4 maxime placeat facere‘ possimus
```

Программа отработала некорректно, так как вместо того, чтобы исключить слова выделенные в кавычках она исключила и всё, что в пределах строки между выделенными словами находилось.

2.13 Правильный шаблон для распознавания строки в кавычках

Приведенная ниже программа демонстрирует работу правильного шаблона для распознования строки в кавычках.

```
%%
'[^'\n]*'    ;
%%

#include "yy.c"
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
At 'vero' eos et 'accusamus' et
iusto 'odio' dignissimos
earum `rerum hic tenetur
maxime placeat facere` possimus
```

В результате получили:

```
At  eos et  et
iusto  dignissimos
earum `rerum hic tenetur
maxime placeat facere` possimus
```

Программа отработала корректно.

2.14 Использование переменной состояния

Приведенная ниже программа заменяет слово magic на first, second, third, в зависимости от того, какая цифра стоит в начале строки.

```
    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"
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
magic Cras pulvinar nisi libero, eu pharetra sapien elementum et
1 Fusce id justo tempor, sollicitudin diam non
3 vestibulum lorem Fusce pellentesque magicante at
2 leo mattis magic condimentum.
magic 1
2 magic magic condimentum
```

В результате получили:

```
magic Cras pulvinar nisi libero, eu pharetra sapien elementum et
1 Fusce id justo tempor, sollicitudin diam non
3 vestibulum lorem Fusce pellentesque <third>ante at
2 leo mattis <second> condimentum.
magic 1
2 <second> <second> condimentum
```

Программа отработала корректно.

2.15 Решение предыдущей задачи при помощи стартовых условий

Программа была модифицирована для решения той же самой задачи, но уже с использованием 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"
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
magic Cras pulvinar nisi libero, eu pharetra sapien elementum et
1 Fusce id justo tempor, sollicitudin diam non
3 vestibulum lorem Fusce pellentesque magicante at
2 leo mattis magic condimentum.
magic 1
2 magic magic condimentum
```

В результате получили:

```
magic Cras pulvinar nisi libero, eu pharetra sapien elementum et
1 Fusce id justo tempor, sollicitudin diam non
3 vestibulum lorem Fusce pellentesque <third>ante at
2 leo mattis <second> condimentum.
magic 1
2 <second> <second> condimentum
```

Программа отработала корректно. Результат был получен тот же самый.

2.16 Трассировка стартовых условий

Опять решается таже задача с модифицированной программой. Добавлена 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"
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
magic Cras pulvinar nisi libero, eu pharetra sapien elementum et
1 Fusce id justo tempor, sollicitudin diam non
3 vestibulum lorem Fusce pellentesque magicante at
2 leo mattis magic condimentum.
magic 1
2 magic magic condimentum
```

В результате получили:

```
magic Cras pulvinar nisi libero, eu pharetra sapien elementum et
1 Fusce id justo tempor, sollicitudin diam non
3 vestibulum lorem Fusce pellentesque <third>ante at
2 leo mattis <second> condimentum.
magic 1
2 <second> <second> condimentum
```

Программа отработала корректно. Результат был получен тот же самый. Кроме того, программа вывела в консоль следующее сообщение:

<0><0><0><0><0><0><0><0><0><0><0><0>
<0><0><0><0><0><0><0><0><0><0><0><0>
<0><0><0><0><0><0><0><0><0><0><0><0>
<0><0><0><0><0><0><0><0><0><0><0><0>
<0><0><0><0><0><0><0><0><0><0><0><0>
<0><0><0><0><0><0><1><1><1><1><1><1>
<1><1><1><1><1><1><1><1><1><1><1><1>
<1><1><1><1><1><1><1><1><1><1><1><1>
<1><1><1><1><1><1><1><1><1><1><1><1>
<1><1><1><1><1><1><1><1><1><1><1><1>
<1><1><1><1><0><3><3><3><3><3><3><3>
<3><3><3><3><3><3><3><3><3><3><3><3>
<3><3><3><3><3><3><3><3><3><3><3><3>
<3><3><3><3><3><3><3><3><3><3><3><3>
<3><3><3><0><2><2><2><2><2><2><2><2>
<2><2><2><2><2><2><2><2><2><2><2><2>

<2><2><2><2><2><2><2><0><0><0><0><0>
<0><0><0><0><2><2><2><2><2><2><2><2>
<2><2><2><2><2><2><2><2>

2.17 Подсчет количества she и he без учета he внутри she

Приведенная ниже программа вычисляет количество he и she в тексте.

```
    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 );
}
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
Praesent arcu esthe congue sed she she heehe posuere sit amet
dictum ac dui he Utshe augue purus consectetur ut
porttitor sed mollis eget she turpis Pellentesque elit felis
posuere vitae he libero id sollicitudin she porta sapien
```

В результате получили:

```
she: 5 times, he: 5 times
```

Программа отработала корректно.

2.18 Подсчет всех экземпляров she и he с использованием REJECT

Приведенная выше программа была модифицированна с использованием команды REJECT.

```
    int s = 0, h = 0;

%%
she     { s++; REJECT; }
he      { h++; REJECT; }
.|\n    ;
%%

#ifndef yywrap
int yywrap() { return( 1 ); }
#endif
```

```
12
13 main()
14 {
15     while( yylex() );
16     printf( "she: %d times, he: %d times\n", s, h );
17 }
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
1 Praesent arcu esthe congue sed she she heehe posuere sit amet
2 dictum ac dui he Utshe augue purus consectetur ut
3 porttitor sed mollis eget she turpis Pellentesque elit felis
4 posuere vitae he libero id sollicitudin she porta sapien
```

В результате получили:

```
1 she: 5 times, he: 10 times
```

Программа отработала корректно. Результат изменился относительно предыдущего пункта. Это связано с тем, что теперь в слове *she* мы находим *he*.

2.19 Подсчет she и he с использованием yyless

Приведенная выше программа была модифицированна с заменой REJECT на yyless.

```
1     int s = 0, h = 0;
2 %%
3 she    { s++; yyless(1); }
4 he     { h++; }
5 .|\n   ;
6 %%
7
8 #ifndef yywrap
9 int yywrap() { return 1; }
10 #endif
11
12 main ()
13 {
14     while (yylex());
15     printf("she: %d times, he: %d times\n", s, h);
16 }
```

Тестовые данные, подаваемые на вход, приведены ниже:

```
1 Praesent arcu esthe congue sed she she heehe posuere sit amet
2 dictum ac dui he Utshe augue purus consectetur ut
3 porttitor sed mollis eget she turpis Pellentesque elit felis
4 posuere vitae he libero id sollicitudin she porta sapien
```

В результате получили:

```
1 she: 5 times, he: 10 times
```

Программа отработала корректно. Результат соответствует тому, что мы получили ранее.

3 Вывод

В результате выполнения работы были получены навыки работы с генератором программ лексической обработки текстов Lex. Было изучено и запущено четырнадцать программ, тексты которых были распространенны преподавателем. Также была разработана собственная программа, в соответствии с индивидуальным заданием. Программы были транслированы в текста на языке C, скомпилированы и запущены с различными входными данными. Эксперименты, которые наиболее полно отражают суть работы программы, приведены в отчёте. Проведенная работа позволила изучить проблемы, возникающие при разработке и шаблоны, которые могу служить решением для них.