Министерство образования и науки Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого
—
Институт кибербезопасности и защиты информации

# ЛАБОРАТОРНАЯ РАБОТА № 1

по дисциплине «Формальные грамматики и теории компиляторов»

Выполнил
студент гр. 4851003/80802 Сошнев М.Д.
*<подпись>*

Преподаватель Мясников А.В.
*<подпись>*

Санкт-Петербург
2021

## Оглавление

# 1 ЦЕЛЬ РАБОТЫ

Познакомиться с парсером, лексером выражений.

# 2 ЗАДАЧА

Описать грамматику, которая будет на вход получать файл с расширением *.z, а на выходе ответ, является ли это корректным исходным файлом без синтаксических ошибок. Если да, то сообщение "OK", например, иначе номер строки с местом ошибки и сообщение "не ок", например.

# 3 ХОД РАБОТЫ

С помощью утилиты pacman были скачены две программы — lex и bison, которые позволяют задавать лексику и грамматику для нашего языка. Создадим два файла — lexic.l и grammar.y в которых будем их задавать. Далее, с помощью скачанных программ из этих файлов будут сгенерирован код на языке Си, который и будет нашим исходным кодом для компилятора. Собрав его с помощью cc мы получим компилятор на выходе. Соответственно, Makefile для сборки нашего компилятора выглядит следующим образом:

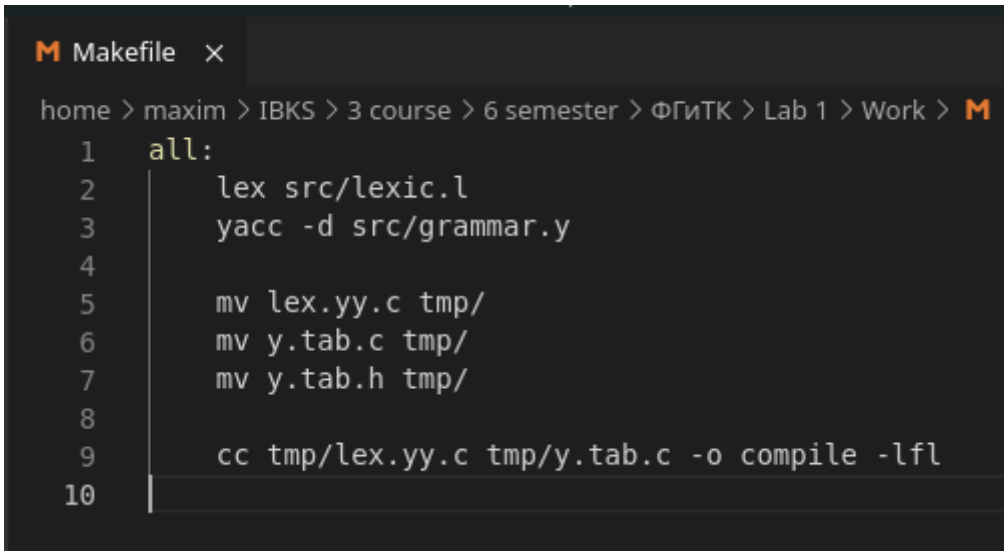


*Рисунок 1 — Makefile для сборки компилятора*

В строчках 2-3 мы генерируем Си-код, далее в строчках 5-7 переносим все исходные тексты будущего компилятора в отдельный каталог и в строчке 9 происходит сборка из исходных текстов.

В файле lexic.l происходит разбиение языка на токены, для каждого токена в коде задаётся отдельный макрос и «передаётся» файлу grammar.y в которых задаются всевозможные цепочки из токенов, которые позволены в данном языке.

```
12
13  %token NUMBER VAR SEMICOLON
14  %token OPEN CLOSE OBRACE EBRACE
15  %token IF ELSE_IF ELSE
16  %token WHILE
17  %token PRINT RETURN
18  %token ASSIGN COMPARE CHANGE
19  %token ADD SUB MUL DIV
20  %token DEC INC
21  %token AND OR BIT_AND BIT_OR
22  %token BIT_XOR BIT_LEFT_SHIFT BIT_RIGHT_SHIFT MOD
23
```

*Рисунок 2 — токены языка.*

В случае, когда в тексте встретится цепочка, которая не соответствует ни одному из правил, будет вызвана функция yyerror(), которую необходимо прописать:

```
100
101  void yyerror(char *s)
102  {
103      ++numError;
104      fprintf(stderr, "%s\n", s);
105  }
106
```

*Рисунок 3 — функция об ошибке*

В конце работы компилятора будет вызвана функция yywrap():

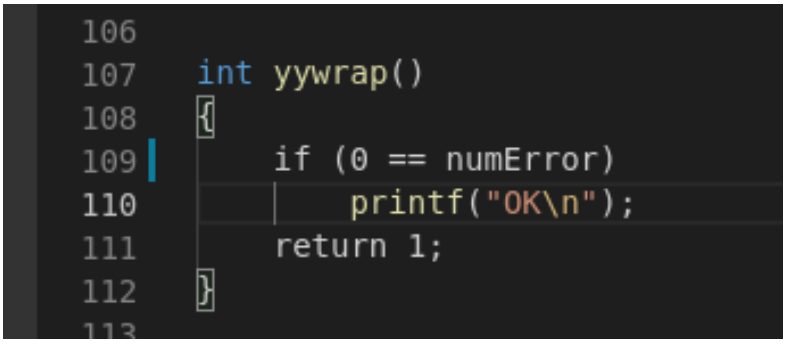


*Рисунок 4 — callback функция конца работы программы*

Входной поток программа определяет с помощью переменной FILE* yyin, в нашем случае это будет файл, название которого задаётся через аргумент командной строки. Результат будет писаться в стандартный поток вывода — либо сообщение «Ok» - текст соответствует всем правилам языка, либо «Syntax error» - обнаружена синтаксическая ошибка.

# 4 ТЕСТИРОВАНИЕ

```
[maxim@maxim-81ag Work]$ make
lex src/lexic.l
yacc -d src/grammar.y
src/grammar.y: предупреждение: 305 конфликтов сдвига/вывода [-Wconflicts-sr]
src/grammar.y: note: rerun with option '-Wcounterexamples' to generate conflict
counterexamples
mv lex.yy.c tmp/
mv y.tab.c tmp/
mv y.tab.h tmp/
cc tmp/lex.yy.c tmp/y.tab.c -o compile -lfl
```

*Рисунок 5 — успешная сборка компилятор*

tests > code.z

```
1   a=1;
2   q = b + 2 + a;
3   i = j = k = l = 2;
4   v = (u = 2);
5   h = (2 > 1);
6
7   p = q;
8
9   if (a == 2) {
10      print 1;
11  } else if (3 == a) {
12      print 2;
13  } else {
14      print 3;
15  }
16
17  b2 = b & a | k ^ 5;
18  b3 = (1 << 5) | (1 << 2);
19
20  a = b + (+a) - (-c);
21
22  while ((a > 0) && (b != 12)) {
23      if (a != b) {
24          a -= b;
25      }
26      else if (a % 2 == 1) {
27          a--;
28      }
29      else {
30          ++a;
31      }
32  }
33
34  print a;
35
36
37  return 0;
38
```

*Рисунок 6 — тестовая программа*

Проверим код на наличие синтаксических ошибок:


*Рисунок 7 — отсутствие синтаксических ошибок*

Теперь  добавим синтаксическую ошибку, например удалим один символ ;

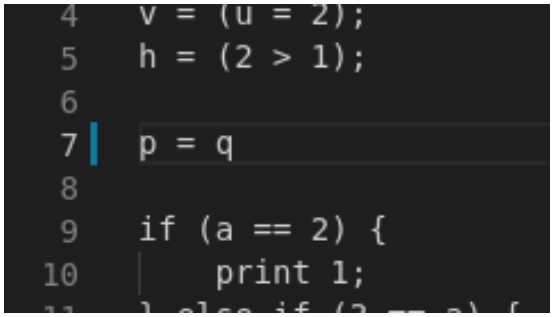


*Рисунок 8 — забытый знак ; вконце строки*

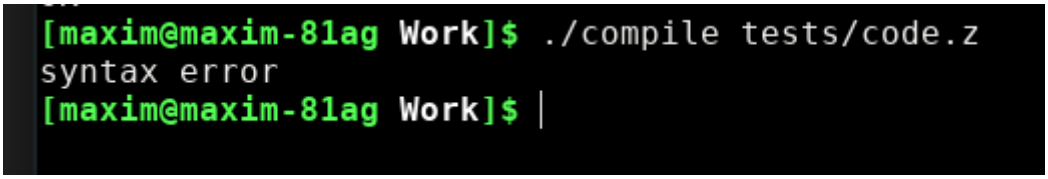


*Рисунок 9 — сообщение об ошибке*

# 5 ВЫВОД

Был получен опыт работы с парсером и лексером, был написан простейший компилятор, обнаруживающий синтаксические ошибки в текстах по заданным конструкциям.

# 6 ПРИЛОЖЕНИЕ

## Makefile

```
all:
	lex src/lexic.l
	yacc -d src/grammar.y

	mv lex.yy.c tmp/
	mv y.tab.c tmp/
	mv y.tab.h tmp/

	cc tmp/lex.yy.c tmp/y.tab.c -o compile -lfl
```

## lexic.l

```
%{
#include <stdlib.h>
#include "y.tab.h"
%}

%%
\{                      return OBRACE;
\}                      return EBRACE;
\(                      return OPEN;
\)                      return CLOSE;

print                     return PRINT;
return                     return RETURN;

if                      return IF;
else\ if                  return ELSE_IF;
else                      return ELSE;
```

```
while                   return WHILE;

=                       return ASSIGN;
;                       return SEMICOLON;

[ \t]+                          /* игнорируем пробелы и знаки
табуляции */
(\r\n)|\n                yylval++;

>|<|>=|<=|==|!=                 return COMPARE;
(\/=)|(\*=)|(\+=)|(-=)          return CHANGE;
\+\+                    return INC;
\-\-                    return DEC;

\+                      return ADD;
\-                      return SUB;
\*                      return MUL;
\/                      return DIV;

&&                      return AND;
\|\|                    return OR;

&                       return BIT_OR;
\|                      return BIT_AND;
\^                      return BIT_XOR;
\<\<                    return BIT_LEFT_SHIFT;
>>                      return BIT_RIGHT_SHIFT;
\%                      return MOD;
```

```
[0-9]*                      return NUMBER;
[a-zA-Z][a-zA-Z0-9]*        return VAR;

%%
```

**grammar.y**

```
%{
#include <stdio.h>

void yyerror(char *s) ;
int yylex();        // ????

extern FILE* yyin;
int numError = 0;
%}

%start commands

%token NUMBER VAR SEMICOLON
%token OPEN CLOSE OBRACE EBRACE
%token IF ELSE_IF ELSE
%token WHILE
%token PRINT RETURN
%token ASSIGN COMPARE CHANGE
%token ADD SUB MUL DIV
%token DEC INC
%token AND OR BIT_AND BIT_OR
%token BIT_XOR BIT_LEFT_SHIFT BIT_RIGHT_SHIFT MOD

%%
```

```
commands:
    /* empty */ |
    commands command;

command:
    PRINT expr SEMICOLON /*{ printf("print\n"); }*/ |
    RETURN expr SEMICOLON /*{ printf("return"); }*/ |
    assigment SEMICOLON /*{ printf("assigment\n"); }*/ |
    expr SEMICOLON { /*printf("expr");*/ } |

    condition |

    cycle_while
    ;

body:
    OBRACE commands EBRACE;

condition:
    IF OPEN expr CLOSE
    body
    else_case;

else_case:
    /* empty */ |
    ELSE_IF OPEN expr CLOSE
    body
    else_case   |
    ELSE
    body;
```

```
cycle_while:
    WHILE OPEN expr CLOSE
    body;

assigment:
    VAR ASSIGN expr { /*printf("assigment");*/ };

expr:
    OPEN expr CLOSE |
    binary_operation |
    unary_operation |
    assigment |
    VAR |
    NUMBER;

unary_operation:
    INC expr | expr INC |
    DEC expr | expr DEC |
    ADD expr | SUB expr;

binary_operation:
    comparation |

    expr ADD expr |
    expr SUB expr |
    expr MUL expr |
    expr DIV expr |

    expr CHANGE expr |
```

```
    expr AND expr |
    expr OR expr |

    expr BIT_AND expr |
    expr BIT_OR expr |
    expr BIT_XOR expr |
    expr BIT_LEFT_SHIFT expr |
    expr BIT_RIGHT_SHIFT expr |
    expr MOD expr;

comparation:
    expr COMPARE expr;

%%

void yyerror(char *s)
{
    ++numError;
    fprintf(stderr, "%s\n", s);
}

int yywrap()
{
    if (0 == numError)
        printf("OK\n");
    return 1;
}

int main(int argc, void *argv[])
```

```
{
    yylval = 0;

    if (2 != argc) {
        printf("Incorrect arguments\n");
        printf("Usage: ./compile [path]\n");

        return -1;
    }

    char* pathFile = argv[1];
    yyin = fopen(pathFile,"r");
    if (NULL == yyin) {
        printf("No such file: %s\n", pathFile);
          return -1;
    }

    yyparse();
    fclose(yyin);

    return 0;
}
```