Федеральное государственное автономное
образовательное учреждение
высшего профессионального образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и Информационных Технологий

Кафедра Информатики

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
Разработка блока синтаксического анализа транслятора

Преподаватель _____ А.С. Кузнецов
подпись, дата

Студент гр. КИ10-11 031010132 _____ К.О. Васильев
подпись, дата

Красноярск 2014

1 Цель работы

Изучение методов синтаксического анализа с их программной реализацией.

2 Постановка задачи

Разработать программу, осуществляющую синтаксический, а также лексический анализ простого языка программирования. Входной язык содержит операторы цикла с параметром, разделенные символом точки с запятой (';'). Операторы цикла состоят из идентификаторов, целочисленных констант в шестнадцатеричной системе, битовых строк (последовательностей из 0 и 1, начинающихся с обязательной пары знаков 0b или 0B), знаков присваивания (':='), знаков операций сравнения (<, >, =, ==, <>, !=, >=, <=) и круглых скобок.

3 Исходный текст распознавателя

```
import ply.yacc as yacc

from lexer import tokens, lxr
from node import Node, Leaf, LeafError

precedence = (
    ('nonassoc', 'LESS', 'GREATER', 'GREQUALS', 'LSEQUALS'),
    ('right', 'ASSIGN'),
    ('left', 'NOTEQUALS', 'EQUALS')
)


def p_program_start(p):
    'program : statement_list'
    p[0] = Node(p, [p[1]])


def p_statement_list(p):
    '''statement_list : statement SEMICOLON statement_list
                      | empty'''
     p[0] = Node(p, [p[1]] if len(p) == 2 else [p[1], Leaf(p, 2),
```

```
p[3]])


def p_nonterminated_statement(p):
    '''statement_list : statement error statement_list'''
        p[0]  =  Node(p,  [p[1],  LeafError('Missing  semicolon',
p.lineno(2)), p[3]])


def p_statement(p):
    '''statement : expression
                 | for_statement'''
    p[0] = Node(p, [p[1]])


def p_expression_constants(p):
    '''expression : HCONST
                  | BCONST
                  | ID'''
    p[0] = Node(p, leaf=Leaf(p, 1))


def p_expression_operations(p):
    '''expression : expression LESS expression
                  | expression GREATER expression
                  | expression GREQUALS expression
                  | expression LSEQUALS expression
                  | expression NOTEQUALS expression
                  | expression EQUALS expression
                  | expression ASSIGN expression'''
    p[0] = Node(p, [p[1], Leaf(p, 2), p[3]])


def p_expression_group(p):
    'expression : LPAR expression RPAR'
    p[0] = Node(p, [Leaf(p, 1), p[2], Leaf(p, 3)])


def p_missing_rpar_error(p):
    '''expression : LPAR expression error'''
     p[0] = Node(p, [Leaf(p, 1), p[2], LeafError('Missing closing
parenthesis', p.lineno(3))])


def p_missing_lpar_error(p):
    'expression : error expression RPAR'
       p[0]  =  Node(p,  [LeafError('Missing  opening  parenthesis',
p.lineno(1)), p[2], Leaf(p, 3)])


def p_for_statement(p):
```

```
        'for_statement : FOR LPAR for_expr SEMICOLON for_expr
SEMICOLON for_expr RPAR statement'
    p[0] = Node(p,
                  [Leaf(p, 1), Leaf(p, 2), p[3], Leaf(p, 4), p[5],
Leaf(p, 6), p[7],
                   Leaf(p, 8), p[9]])


def p_for_expr(p):
    '''for_expr : expression
                | empty'''
    p[0] = p[1]


def p_empty(p):
    'empty :'
    p[0] = Node(p)


def p_error(p):
    if not p:
        print "Syntax error at EOF"
    else:
                    print('Unexpected symbol \'{0}\' at line
{1}'.format(p.value, p.lineno))
        yacc.errok()
        return yacc.token()




yacc_parser = yacc.yacc(debug=True)


def yparse(data, debug=0):
    yacc_parser.error = 0
    lxr.lineno = 1
    p = yacc_parser.parse(data, debug=debug, lexer=lxr)
    if yacc_parser.error:
        return None
    return p
```

4 Тестовые примеры работы программы

Выводиться одновременно будут дерево разбора и полученные лексемы.

Передадим программе пустой файл, тогда результат будет таким:

Parsed tree:

```
program/1
  1 statement_list/1
    1 empty/0
```

Lexemes:

Теперь передадим программе файл, содержащий код:

```
for(a := 0b11; a != 0xf;)
        b < a;
```

Результат работы программы будет следующий:

```
Parsed tree:
program/1
  1 statement_list/3
    1 statement/1
      1 for_statement/9
        1 FOR [name:'for']
        2 LPAR [text:'(']
        3 expression/3
          1 expression/1
            1 ID [name:'a']
          2 ASSIGN [text:':=']
          3 expression/1
            1 BCONST [text:'0b11']
        4 SEMICOLON [text:';']
        5 expression/3
          1 expression/1
            1 ID [name:'a']
          2 NOTEQUALS [text:'!=']
          3 expression/1
            1 HCONST [text:'0xf']
        6 SEMICOLON [text:';']
        7 empty/0
        8 RPAR [text:')']
        9 statement/1
          1 expression/3
            1 expression/1
              1 ID [name:'b']
            2 LESS [text:'<']
```

```
        3 expression/1
          1 ID [name:'a']
    2 SEMICOLON [text:';']
    3 statement_list/1
      1 empty/0
```

```
Lexemes:
KEYWORD:        for             line:1
PUNCTUATOR:     (               line:1
ID:             a               line:1
OPERATOR:       :=              line:1
CONSTANT:       0b11            line:1
PUNCTUATOR:     ;               line:1
ID:             a               line:1
OPERATOR:       !=              line:1
CONSTANT:       0xf             line:1
PUNCTUATOR:     ;               line:1
PUNCTUATOR:     )               line:1
ID:             b               line:2
OPERATOR:       <               line:2
ID:             a               line:2
PUNCTUATOR:     ;               line:2
```

Если передать программе файл, содержащий следующий код:

```
0x5 := a

a < b;
```

То на выходе будет получена следующая таблица:

```
Unexpected symbol 'a' at line 2
Parsed tree:
program/1
  1 statement_list/3
    1 statement/1
      1 expression/3
        1 expression/3
          1 expression/1
            1 HCONST [text:'0x5']
          2 ASSIGN [text:':=']
```

```
      3 expression/1
        1 ID [name:'a']
     2 LESS [text:'<']
     3 expression/1
       1 ID [name:'b']
  2 SEMICOLON [text:';']
  3 statement_list/1
    1 empty/0
```

```
Lexemes:
CONSTANT:       0x5             line:1
OPERATOR:       :=              line:1
ID:             a               line:1
ID:             a               line:2
OPERATOR:       <               line:2
ID:             b               line:2
PUNCTUATOR:     ;               line:2
```