Университет ИТМО Кафедра ИПМ

Отчет по лабораторной работе № 1

«Разработка лексического анализатора»

Выполнил: студент группы P3317

Плюхин Д.А.

Преподаватель: Лаздин Артур Вячеславович

Санкт-Петербург
2017 год

Цель работы

Разработать и отладить лексический анализатор для заданной грамматики.

БНФ реализуемого языка

```
<Программа> ::= <Объявление переменных> <Описание вычислений>
<Описание вычислений> ::= Begin < Список операторов > End.
<Объявление переменных> ::= Var <Список переменных>
<Список переменных> ::= <Идент>; | <Идент> , <Список переменных> | <Идент> ; <Список
переменных>
<Список операторов> ::= <Оператор> | <Оператор> <Список операторов>
<Оператор>::=<Присваивание> |<Сложный оператор>|<Составной оператор>
<Составной оператор>::= Begin < Список операторов > End;
<Присваивание> ::= <Идент> = <Выражение> ;
<Выражение> ::= <Ун.оп.><Подвыражение> | <Подвыражение>
<Подвыражение> :: = ( <Выражение> ) | <Операнд> | < Подвыражение > <Бин.оп.>
<Подвыражение>
<Ун.оп.> ::= "-" | not
<Бин.оп.> ::= "-" | "+" | "*" | "/" | "**" | ">" | "<" | "=="
<Операнд> ::= <Идент> | <Const>
<Сложный оператор>::= If (< Выражение> ) <Оператор>;
<Идент> ::= <Буква> <Идент> | <Буква>
<Const> ::= <Цифра> <Const> | <Цифра>
```

Список классов лексем реализуемого языка

class name	possible values
KEYWORD	Begin End If Var
SEPARATOR	, ; .
IDENTIFIER	[a-zA-Z]
CONSTANT	[0-9]
EQUALITY OPERATOR	=
BINARY SIMPLE OPERATOR	- +
BINARY COMPLEX OPERATOR	** / *
BRACKET	( )
UNARY OPERATOR	- not
BINARY COMPARASION OPERATOR	> < ==

Листинг программы с описанием входящих в ее состав процедур

```
#(название класса лексемы; следует ли искать точное соответствие в наборе значений или же там
#приведено регулярное выражение; набор значений; набор названий классов лексем, которые могут предшествовать
#лексеме данного класса)
lex_classes = (("KEYWORD",True,("Begin","End","If","Var"),[]),
               ("SEPARATOR",True,(",",";","."),[]),
               ("IDENTIFIER",False,["[a-zA-Z]"],[]),
               ("CONSTANT",False,["[0-9]"],[]),
               ("EQUALITY OPERATOR",True,["="],["IDENTIFIER"]),
               ("BINARY SIMPLE OPERATOR",True,("-","+"),["IDENTIFIER"]),
               ("BINARY COMPLEX OPERATOR",True,("**","/","*"),["IDENTIFIER"]),
               ("BRACKET",True,("(",")"),[]),
               ("UNARY OPERATOR",True,("-","not"),["EQUALITY OPERATOR", "BRACKET", "KEYWORD"]),
               ("BINARY COMPARASION OPERATOR",True,(">","<","=="),["IDENTIFIER"]))

#Добавление в результирующую таблицу токена с номером строки, где он был обнаружен.
#Запись в таблице идентифицируется именем класса и значением токена.
#Если такой токен в таблице уже есть, то записывается только дополнительный номер строки.
def append_token_to_table(token_class, token_name, row_number):
    try:
        table.get((token_class, token_name)).append(row_number)
    except:
        table[(token_class, token_name)] = [row_number]

#Чтение программы из внешнего файла
def get_program_from_file(filename):
    program = ""
    with open(filename) as file_with_program:
        program += file_with_program.read()
    return program
```

```
#Попытка выделения лексемы из "урезанной" программы, то есть из программы, n начальных символов которой
#отброшены - при этом предполагается, что лексема должна начинаться с первого символа переданной
#программы и имеет максимально возможный размер. Функция возвращает длину выделенного токена в случае
#его идентификации и -1 в случае ошибки. Помимо всего прочего функция записывает строку в протокол
#разбора программы, а также фиксирует результат в таблице токенов
def match(program, lex_classes):
    global row_number
    global previous_class
    global comment
    global stop_after_fail
    if (ord(program[0]) == 10):
        row_number += 1
        if (comment):
            comment = False
        return 1
    if (comment):
        return 1
    if (program[:2] == "//"):
        comment = True
        return 1
    for lex_class in lex_classes:
        if ((len(lex_class[3]) != 0) and (previous_class not in (lex_class[3]))):
            continue
        if (lex_class[1]):
            for lex_template in lex_class[2]:
                if (program[:len(lex_template)] == lex_template):
                    report.append("%2i %s %s" % (row_number, lex_class[0], lex_template))
                    #print("%2i %s %s" % (row_number, lex_class[0], lex_template))
                    append_token_to_table(lex_class[0], lex_template, row_number)
                    previous_class = lex_class[0]
                    return len(lex_template)
        else:
            for lex_template in lex_class[2]:
                length = 1
                matching_result = re.search("%s{%i}" % (lex_template, length),program[:length])
                if (matching_result == None):
                    continue
                while (matching_result != None):
                    length += 1
                    matching_result = re.search("%s{%i}" % (lex_template, length),program[:length])
                length -= 1
                matching_result = re.search("%s{%i}" % (lex_template, length),program[:length]).group(0)
                if (lex_class[0] == "CONSTANT"):
                    matching_result = hex(int(matching_result))
                report.append("%2i %s %s" % (row_number, lex_class[0], matching_result))
                append_token_to_table(lex_class[0], matching_result, row_number)
                previous_class = lex_class[0]
                return length
    if (stop_after_fail):
        return -1
    return 1

#Удаление пробелов и символов табуляции из строки
def delete_spaces(string):
    return string.replace(chr(9),"").replace(chr(32),"")

#Основная функция, которая производит последовательный анализ переданной программы, выводит протокол
#анализа программы в случае успешного завершения и результирующую таблицу, и сообщение об ошибке,
#если был встречен неверный токен или неверная пара токенов
def analyze(program):
    global report
    start_index = 0

    while (start_index < len(program)):
        matching_result = match(program[start_index:], lex_classes)
        if (matching_result < 0):
            print("Program contains invalid token. Analyzing aborted.")
            return
        start_index += matching_result
    print("Program analyzed successfully")
    print("Program :")
    for row in report:
        print(row)
    print("Table : ")
    print("%-30s %-20s %-30s" % ("token class","token","row numbers"))
    for key in table:
        print("%-30s %-20s %-30s" % (key[0],key[1],",".join([str(row_num) for row_num in set(table[key])])))
```

Контрольные примеры и результаты их выполнения

Пример 1

```
Var a, b, c;
Begin
```

```
      //Program one
      a = 5;
      b = -(a + 8)
      If (a + b < 12)
      Begin
            a = a ** 2
      End
End.
```

Результат

token class	token	row numbers
KEYWORD	Var	1
IDENTIFIER	a	1,4,5,6,8
SEPARATOR	,	1
IDENTIFIER	b	1,5,6
IDENTIFIER	c	1
SEPARATOR	;	1,4
KEYWORD	Begin	2,7
EQUALITY OPERATOR	=	8,4,5
CONSTANT	0x5	4
UNARY OPERATOR	-	5
BRACKET	(	5,6
BINARY SIMPLE OPERATOR	+	5,6
CONSTANT	0x8	5
BRACKET	)	5,6
KEYWORD	If	6
BINARY COMPARASION OPERATOR	<	6
CONSTANT	0xc	6
BINARY COMPLEX OPERATOR	**	8
CONSTANT	0x2	8
KEYWORD	End	9,10
SEPARATOR	.	10

Пример 2

```
Var foo, bar, acc;
Begin
      //Program two
      foo = 1;
      bar = 2;
      acc = 3;
      If (bar > acc)
      Begin
            If (acc == foo)
            Begin
                  acc = foo * ( - (bar - acc))
            End;
            bar = acc + foo
      End;
End.
```

Результат

token class	token	row numbers
KEYWORD	Var	1
IDENTIFIER	foo	1,4,9,11,13
SEPARATOR	,	1
IDENTIFIER	bar	1,5,7,11,13
IDENTIFIER	acc	1,6,7,9,11,13
SEPARATOR	;	1,4,5,6,12,14
KEYWORD	Begin	8,2,10
EQUALITY OPERATOR	=	4,5,6,11,13
CONSTANT	0x1	4
CONSTANT	0x2	5
CONSTANT	0x3	6
KEYWORD	If	9,7
BRACKET	(	9,11,7

```
BINARY COMPARASION OPERATOR    >                   7
BRACKET                        )                   9,11,7
BINARY COMPARASION OPERATOR    ==                  9
BINARY COMPLEX OPERATOR        *                   11
UNARY OPERATOR                 -                   11
BINARY SIMPLE OPERATOR         -                   11
KEYWORD                        End                 12,14,15
BINARY SIMPLE OPERATOR         +                   13
SEPARATOR                      .                   15
```

Пример 3

```
Var one, two, three, value;
Begin
      value = 20;
      if (one && two && three)
      Begin
            If (value == 20) value = -50;
      End;
End.
```

Результат

```
token class                    token               row numbers
KEYWORD                        Var                 1
IDENTIFIER                     one                 1,4
SEPARATOR                      ,                   1
IDENTIFIER                     two                 1,4
IDENTIFIER                     three               1,4
IDENTIFIER                     value               1,3,6
SEPARATOR                      ;                   1,3,6,7
KEYWORD                        Begin               2,5
EQUALITY OPERATOR              =                   3,6
CONSTANT                       0x14                3,6
IDENTIFIER                     if                  4
BRACKET                        (                   4,6
BRACKET                        )                   4,6
KEYWORD                        If                  6
BINARY COMPARASION OPERATOR    ==                  6
UNARY OPERATOR                 -                   6
CONSTANT                       0x32                6
KEYWORD                        End                 8,7
SEPARATOR                      .                   8
```

Вывод

Таким образом, был разработан и отлажен лексический анализатор для относительно простого языка программирования.