ГУАП

КАФЕДРА 44

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент, канд. техн. наук |  |  |  | Н. В. Кучин |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| ПОСТРОЕНИЕ РАСПОЗНОВАТЕЛЯ ДЛЯ РЕГУЛЯРНОЙ ГРАММАТИКИ И ПОСТРОЕНИЕ ЛЕКСИЧЕСКОГО АНАЛИЗАТОРА |
| по курсу: СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4941 |  |  |  | Н.С. Горбунов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

1. **Цель работы:** изучение основных понятий теории регулярных языков и грамматик, ознакомление с назначением и принципами работы конечных автоматов (КА), получение практических навыков построения КА на основе заданной регулярной грамматики; ознакомление с назначением и принципами работы лексических анализаторов (сканеров), получение практических навыков построения сканера на примере заданного простейшего входного языка.
2. **Вариант 7**

Входной язык содержит арифметические выражения, разделенные символом ; (точка с запятой). Арифметические выражения состоят из идентификаторов, римских чисел, знака присваивания (:=), знаков операций +, -, \*, / и круглых скобок.

1. **Возможные входные данные языка:**

* Буквы латинского алфавита составляющие идентификаторы и римские цифры
* Специальные символы для обозначения присваивания и конца

строки - «:=» и «;» соответственно

1. **Таблицы**

Таблица 1. Крайние левые и правые символы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Символ (U) | Шаг 1 (начало построения) | | Последний шаг (результат) | |
| L(U) | R(U) | L(U) | R(U) |
| E | ( F - a | F ) a | ( F T E - a | F ) T E a |
| T | T E | E | T E ( F - a | E ) T a |
| F | F T | T | F T E ( - a | T E ) a |
| S | a | F | a | F T E ) a |

Таблица 2. Крайние левые и правые терминальные символы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Символ (U) | Шаг 1 (начало построения) | | Последний шаг (результат) | |
| Lt (U) | Rt (U) | Lt (U) | Rt (U) |
| E | ( - a | ) a | ( - a \* / + | ) a \* / + |
| T | \* / | \* / | \* / ( - a + | \* / ) a + |
| F | + | + | \* / ( - a + | ) a \* / + |
| S | a | ; | a | ; + \* / ) a |

Таблица 3. Матрица предшествований

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ( | ) | - | + | \* | / | a | ; | ⊥к |
| ( | <. |  | <. | <. | <. | <. | <. |  |  |
| ) |  | .> | .> | .> | .> | .> | .> |  | .> |
| - | <. |  | <. | <. | <. | <. | <. |  |  |
| + | <. | .> | <. | <. | <. | <. | <. |  | .> |
| \* | <. | .> | <. | <. | .> | <. | <. |  | .> |
| / | <. | .> | <. | <. | <. | <. | <. |  | .> |
| a |  |  |  |  |  |  |  |  | .> |
| ; |  | .> |  | .> | .> | .> | .> |  | .> |
| ⊥н |  |  |  |  |  |  | <. |  |  |

1. **Программа**

Программа синтаксического анализатора написана с использование автоматизированного средства построения «YACC». Использованные языки: GUI – C#, Анализатора – Python

Листинг 1. Использование YACC через Python

# ---------------------------------------------------------------------------#

#

#

# Грамматика: S: a:=F;

#

# F: F + T

# | T

#

# T: T \* E

# | T/E

# | E

#

# E: (F)

# | -(F)

# | a

#

# ---------------------------------------------------------------------------

import re

from sys import argv

from ply.lex import lex

from ply.yacc import yacc

# Наименование символов

tokens = ['PLUS', 'MINUS', 'DIVIDER', 'MULT', 'DIVIDE',

'ID', 'ROMAN', 'ASSIGN', 'LEFTBRECK', 'RIGHTBRECK']

# игнорируемые символы

t\_ignore = ' \t'

# регулярные выражения

t\_PLUS = r'\+'

t\_MINUS = r'-'

t\_MULT = r'\\*'

t\_DIVIDE = r'/'

t\_DIVIDER = r';'

t\_ASSIGN = r':='

t\_LEFTBRECK= r'\('

t\_RIGHTBRECK= r'\)'

# thousand = 'M{0,3}'

# hundred = '(C[MD]|D?C{0,3})'

# ten = '(X[CL]|L?X{0,3})'

# digit = '(I[VX]|V?I{1,3})'

# t\_ROMAN = r"%s%s%s%s$" % (thousand, hundred, ten, digit)

t\_ROMAN = r'^[IVX]+$'

# Методы для лексем

def t\_ID(t): # тип ID

r'[a-zA-Z\_][a-zA-Z\_]\*' # регулярное выражения для идентификатора

t.type = reserved.get(t.value, 'ID')

if bool(re.match(t\_ROMAN, t.value)):

t.type = 'ROMAN' # проверка на римские числа

return t

def t\_ignore\_newline(t): # игнорируем символ новой строки

r'\n+'

t.lexer.lineno += t.value.count('\n') # увеличиваем номер строки, если был этот символ

# комментарии

def t\_COMMENT(t):

r'\{.\*\}'

pass

# пропуск токена комментария

# при ошибке

def t\_error(t):

print(f'Illegal character {t.value[0]!r}') # ошибка, неизвестный символ

t.lexer.skip(1) # пропуск неизвестного символа

# лексер:

lexer = lex()

# --- Парсер

# Грамматика

def p\_empty(p): # правила грамматики

'''program :''' # Программ

p[0] = []

def p\_program(p):

'''program : program S ''' # выражение для работы нескольки строк

p[0] = p[1] # добавление токена для вывода древа

p[0].append(p[2]) # добавление токена для вывода древа

def p\_S(p):

'''

S : term ASSIGN F DIVIDER

''' # правило грамматики, S: a:=F;

p[0] = ('S', p[0], p[1], p[2], p[3]) # добавление токенов выражения для вывода древа

def p\_F(p):

'''

F : F PLUS T

| T

''' # F: F + T| T

try:

p[0] = ('F', p[1], p[2], p[3])

except:

p[0] = ('F', p[1]) # добавление токенов выражения для вывода древа

def p\_T(p):

'''

T : T MULT expression

| T DIVIDE expression

| expression

''' # T -> T \* E| T/E | E

try:

p[0] = ('T', p[1], p[2], p[3]) # добавление токена для вывода древа

except:

p[0] = ('T', p[1])

def p\_T\_expression(p):

'''

expression : LEFTBRECK F RIGHTBRECK

| MINUS LEFTBRECK F RIGHTBRECK

''' # (F)| -(F) | a

try:

p[0] = ('E', p[1], p[2], p[3], p[4])

except:

p[0] = ('E', p[1], p[2], p[3]) # добавление токена для вывода древа

def p\_t\_expression\_term(p):

'''

expression : term

''' # a

p[0] = ('E', p[1]) # добавление токена для вывода древа

def p\_term\_factor(p):

'''

term : factor

''' # a -> ROMAN|ID

p[0] = p[1] # добавление токена для вывода древа

def p\_factor\_roman(p):

'''

factor : ROMAN

''' # римское число

p[0] = ('roman', p[1]) # добавление токена для вывода древа

def p\_factor\_ID(p):

'''

factor : ID

''' # ID

p[0] = ('id', p[1]) # добавление токена для вывода древа

def p\_error(p):

print("Syntax error in input!")

# Парсер

parser = yacc()

data = str(argv[1])

print(data) # вывод введенной строки

ast = parser.parse(data) # парсинг строки

#print(ast)

if len(ast)!=0:

print('Success') # вывод если парсер завершил работу без ошибок

print(ast) # вывод дерева

# данные из строки в лексер

lexer.input(data)

# Вывод токенов (работа лексера)

while True:

tok = lexer.token()

if not tok:

break # No more input

print(tok)

1. **Скриншоты работы программы**

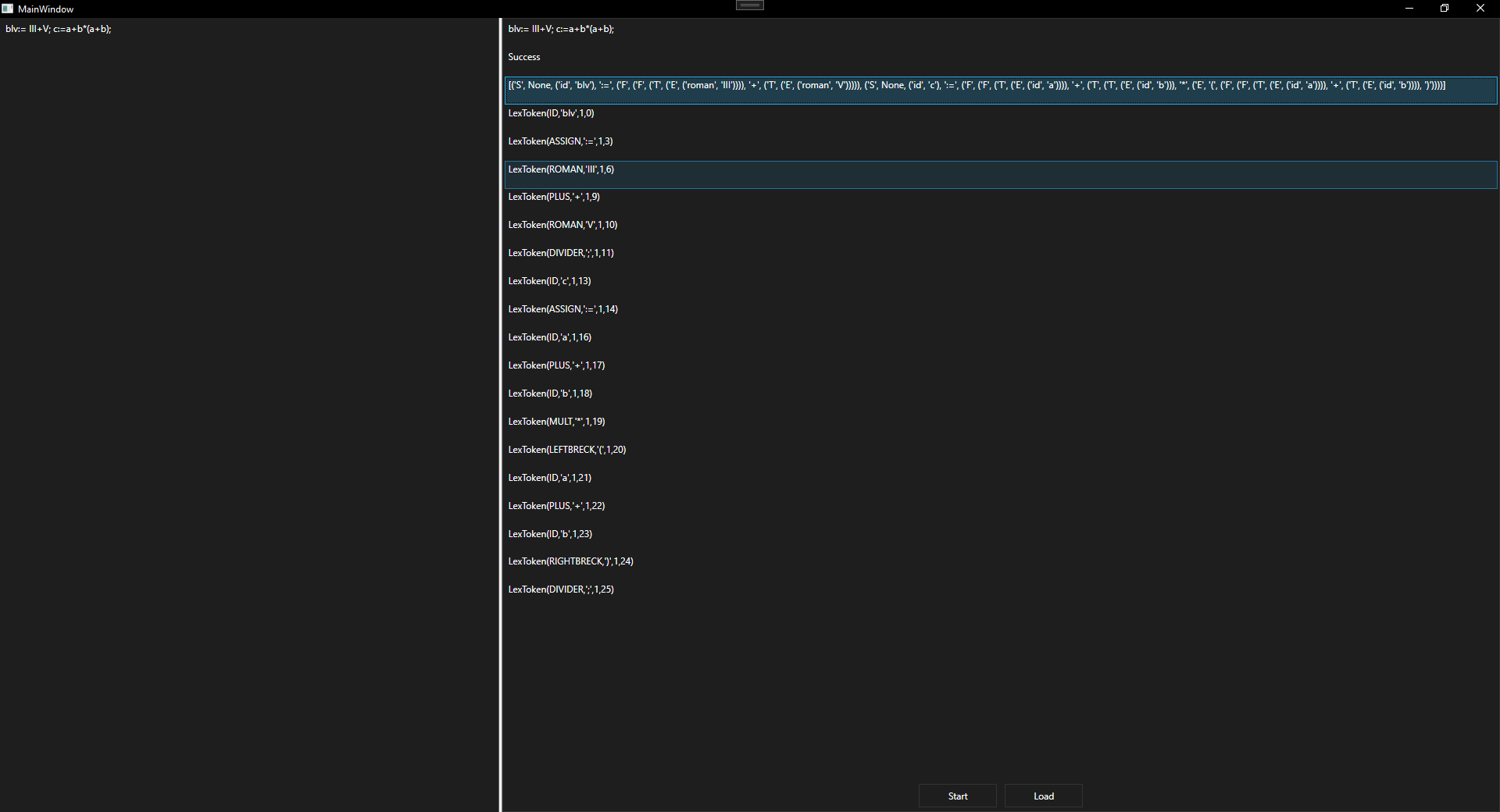


Рисунок 1. Общий вид окна

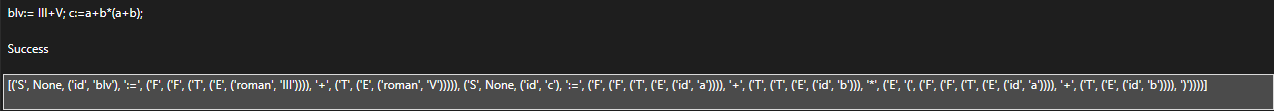


Рисунок 2. Вывод дерева

1. **Граф дерева**

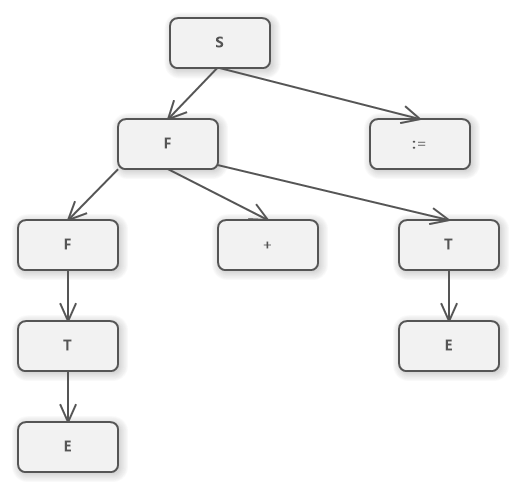


Рисунок 3. Граф дерева вывода от «bIv:= III+V;»