**Цель работы:** «Построение синтаксического анализатора

Основная задача синтаксического анализа - проверка исходной программы на соответствие грамматике языка программирования. Следует еще раз напомнить, что синтаксический анализ производится над кодом программы, который получен на выходе лексического анализа. Результат синтаксического анализа, который часто называется разбором, представляется в виде дерева разбора. Данное дерево должно демонстрировать вывод исходной программы как цепочки символов из начального символа грамматики.

**Код программы:**

from Lab1 import \*

def char\_init(arr, in\_arr, iterator):

flag = 0

iter = iterator

while iter < len(arr) and flag != 1:

for el in in\_arr:

if el == arr[iter]:

flag = 1

if(flag == 0):

iter += 1

return iter

def multp(multip):

iter = 0

if len(multip)>1:

left\_rart = multip[0:len(multip)]

if len(left\_rart) > 1:

if (["R7"] in left\_rart or ["R8"] in left\_rart):

if "I" in multip[0][0]:

open\_bracket = char\_init(multip, ["R7"], 0)

close\_bracket = char\_init(multip, ["R7"], 0)

if open\_bracket == len(multip) or close\_bracket == len(multip):

return False

if(open\_bracket - close\_bracket == 1):

return False

perem = open\_bracket+1

while close\_bracket - perem>=1:

end\_perem = char\_init(multip, ["R11"], perem)

if(multip[end\_perem] != ["R11"]):

if not Expression(left\_rart[perem:close\_bracket]):

return False

if not Expression(left\_rart[perem:end\_perem]):

return False

perem = end\_perem+1

return True

elif (["R3"] in left\_rart and ["R2"] in left\_rart):

if "I" in multip[0][0]:

open\_bracket = char\_init(multip, ["R7"], 0)

close\_bracket = char\_init(multip, ["R7"], 0)

if (open\_bracket - close\_bracket == 1):

return False

perem = open\_bracket + 1

while close\_bracket - perem >= 1:

end\_perem = char\_init(multip, ["R11"], perem)

if (multip[end\_perem] != ["R11"]):

if not Expression(left\_rart[perem:close\_bracket]):

return False

if not Expression(left\_rart[perem:end\_perem]):

return False

perem = end\_perem + 1

return True

if "N" in multip[iter][0] or "C" in multip[iter][0]:

return True

elif "I" in multip[iter][0]:

return True

elif ['R3'] == multip[iter]:

new\_iter = char\_init(multip, ["R2"], iter)

if multip[new\_iter] == ['R2']:

if Expression(multip[iter + 1:new\_iter - 1]):

return True

return False

return False

def sums(sums):

if (len(sums) != 0):

iter = 0

new\_iter = char\_init(sums, [['O3'], ['O4']], iter)

if (multp(sums[iter:new\_iter])):

iter = new\_iter+1

while (iter < len(sums)):

new\_iter = char\_init(sums, [['O3'], ['O4']], iter)

if (multp(sums[iter:new\_iter])):

iter = new\_iter + 1

else:

return False

else:

return False

return True

def Expression(in\_expr):

if(len(in\_expr)!=0):

iter = 0

new\_iter = char\_init(in\_expr, [['O1'], ['O2']], iter)

if (sums(in\_expr[iter:new\_iter])):

iter = new\_iter+1

else:

return False

while(iter<len(in\_expr)):

new\_iter = char\_init(in\_expr, [['O1'], ['O2']], iter)

if(sums(in\_expr[iter:new\_iter])):

iter = new\_iter+1

else:

return False

return True

def equals(in\_assig):

iter = char\_init(in\_assig, [["O12"]], 0)

if iter == len(in\_assig):

return False

else:

left\_rart = in\_assig[0:iter]

if len(left\_rart) > 1:

if(["R7"] in left\_rart and ["R8"] in left\_rart):

if("I" in in\_assig[0][0] and Expression(left\_rart[2:len(left\_rart)-1]) and Expression(in\_assig[iter+1:len(in\_assig)])):

return True

return False

param = char\_init(in\_assig, [["R4"]], 0)

if param - iter == 1:

return False

if("I" in in\_assig[0][0] and Expression(in\_assig[iter+1:len(in\_assig)])):

return True

return False

#Условие

def condition(in\_cond):

iter = char\_init(in\_cond, [['06'], ['07'], ['O8'], ['O9'], ['O10'], ["O11"]], 0)

if(iter == len(in\_cond)):

return False

else:

if Expression(in\_cond[0:iter]) and Expression(in\_cond[iter+1:len(in\_cond)]):

return True

else:

return False

def cond\_operator(in\_if):

if(["W2"] in in\_if):

open\_cond = char\_init(in\_if, [["R3"]], 1)

close\_cond = char\_init(in\_if, [["R2"]], 1)

if(open\_cond == close\_cond-1):

print("Условие не задано")

return False

elif close\_cond == len(in\_if):

return False

else:

if(condition(in\_if[open\_cond + 1:close\_cond])):

open\_oper = char\_init(in\_if, [["R12"]], 1)

close\_oper = char\_init(in\_if, [["R13"]], 1)

if (close\_oper == len(in\_if)):

return False

if (open\_oper == close\_oper - 1):

return False

else:

open\_oper = char\_init(in\_if, [["R4"]], open\_oper) + 1

if(syntax\_analizer(in\_if[open\_oper:close\_oper])):

if(close\_oper != len(in\_if) and ["W3"] in in\_if):

open\_oper = char\_init(in\_if, [["R12"]], close\_oper)

close\_oper = char\_init(in\_if, [["R13"]], close\_oper + 1)

if(close\_oper == len(in\_if)):

return False

if (open\_oper == close\_oper - 1):

return False

else:

return True

return True

else:

return False

else:

return False

else:

return False

def function(in\_f):

open\_cond = char\_init(in\_f, [["R3"]], 1)

close\_cond = char\_init(in\_f, [["R2"]], 1)

if open\_cond == len(in\_f) or close\_cond == len(in\_f):

return False

param = open\_cond + 1

while close\_cond - param >= 1:

if "I" in in\_f[param][0]:

if(close\_cond - param == 1):

param +=1

else:

if ["R11"] == in\_f[param+1]:

param +=2

else:

return False

else:

return False

open\_cond = char\_init(in\_f, [["R12"]], 1)

close\_cond = char\_init(in\_f, [["W9"]], 1)

if close\_cond == len(in\_f):

return False

if not syntax\_analizer(in\_f[open\_cond + 2:close\_cond]):

return False

open\_cond = char\_init(in\_f, [["R3"]], close\_cond)

close\_cond = char\_init(in\_f, [["R2"]], open\_cond)

if open\_cond == len(in\_f) or close\_cond == len(in\_f):

return False

if(close\_cond - open\_cond == 1):

return False

if(Expression(in\_f[open\_cond+1:close\_cond])):

return True

return False

def cicle(in\_w):

if (["W4"] in in\_w):

open\_cond = char\_init(in\_w, [["R3"]], 1)

close\_cond = char\_init(in\_w, [["R2"]], 1)

if (open\_cond == close\_cond - 1):

return False

else:

if (condition(in\_w[open\_cond + 1:close\_cond])):

open\_oper = char\_init(in\_w, [["R12"]], 1)

close\_oper = char\_init(in\_w, [["R13"]], 1)

if (close\_oper == len(in\_w)):

return False

if (open\_oper == close\_oper - 1):

return False

else:

open\_oper = char\_init(in\_w, [["R4"]], open\_oper)

if (syntax\_analizer(in\_w[open\_oper + 1:close\_oper])):

return True

else:

return False

else:

return False

else:

return False

def oper\_in(s, n):

if n == 1:

return cicle(s)

if n == 2:

return function(s)

if n == 3:

return cond\_operator(s)

if n == 4:

return equals(s)

return False

def syntax\_analizer(text):

iter = 0

if text == []:

return False

else:

while(iter < len(text)):

new\_iter = char\_init(text, [["R4"]], iter)

if ["W2"] in text[iter:new\_iter]:

new\_iter = char\_init(text, [["R13"]], new\_iter)

#Если мы её не нашли и упёрлись в конец программы

if(new\_iter == len(text)):

return False

prov = char\_init(text, [["R12"]], iter)

if ['W2']in text[prov + 1:new\_iter] or ['W3'] in text[prov + 1:new\_iter] or ['W4'] in text[prov + 1:new\_iter]:

return False

start\_next\_str = char\_init(text, [["R4"]], new\_iter) + 1

end\_next\_str = char\_init(text, [["R4"]], start\_next\_str)

if(['W3'] in text[start\_next\_str:end\_next\_str]):

new\_iter = char\_init(text, [["R13"]], start\_next\_str + 1)

prov = char\_init(text, [["R12"]], end\_next\_str)

if ['W2'] in text[prov + 1:new\_iter] or ['W3'] in text[prov + 1:new\_iter] or [

'W4'] in text[

prov + 1:new\_iter]:

return False

if oper\_in(text[iter:new\_iter + 1], 3):

iter = char\_init(text, [["R4"]], new\_iter) + 1

else:

return False

else:

if oper\_in(text[iter:new\_iter + 1], 3):

iter = char\_init(text, [["R4"]], new\_iter) + 1

else:

return False

elif ["W4"] in text[iter:new\_iter]:

new\_iter = char\_init(text, [["R13"]], new\_iter)

if (new\_iter == len(text)):

return False

# }

prov = char\_init(text, [["R12"]], iter)

if ['W2'] in text[prov + 1:new\_iter] or ['W3'] in text[prov + 1:new\_iter] or ['W4'] in text[prov + 1:new\_iter]:

return False

if oper\_in(text[iter:new\_iter + 1], 1):

iter = char\_init(text, [["R4"]], new\_iter) + 1

else:

return False

elif ["W8"] in text[iter:new\_iter]:

new\_iter = char\_init(text, [["R13"]], new\_iter)

if (new\_iter == len(text)):

return False

prov = char\_init(text, [["R12"]], iter)

if ['W2'] in text[prov + 1:new\_iter] or ['W3'] in text[prov + 1:new\_iter] or ['W4'] in text[prov + 1:new\_iter]:

return False

if oper\_in(text[iter:new\_iter + 1], 2):

iter = char\_init(text, [["R4"]], new\_iter) + 1

else:

return False

elif ["O12"] in text[iter:new\_iter]:

if oper\_in(text[iter:new\_iter], 4):

iter = char\_init(text, [["R4"]], new\_iter) + 1

else:

return False

else:

return False

return True

def get\_array(arr):

new\_arr = []

for i in arr:

for j in i:

new\_arr.append(j)

return new\_arr

syntax = get\_array(lexical\_analize)

if syntax\_analizer(syntax):

print('Программа прошла синтаксический анализ!')

else:

print('Программа не прошла синтаксический анализ!')

**Примеры работы программы:**

Пример №1:

if(x==y){

a=x-y;

}

else{

a=y-x;

}

Результат: Программа прошла синтаксический анализ!

Пример №2:

while(x!=y){

a=a+x;

y=y+1;

}

Результат: Программа прошла синтаксический анализ!

Пример №3:

while(x===y){

a=a+x;

y=y+1;

}

Результат: Программа не прошла синтаксический анализ!

Пример №4:

function func(x,y){

x=a+5;

y=x-b;

return(y+x)

}

Результат: Программа прошла синтаксический анализ!

**Вывод:** в данной работе был разработан и проведен синтаксический анализ кода языка javascript.