Лабораторная работа №3

“Исследование алгоритма нисходящего распознавателя

языка LL(k) - грамматики”

1 ПОСТАНОВКА ЗАДАЧИ

Таблица 1 – Фрагмент программы для анализа



2 ХОД РАБОТЫ

1. Построение формальной грамматики:

<фрагмент> ::= <идентификатор> : FUNC ( <аргумент> ) \ <тело программы> FINAL

<идентификатор> ::= <iden>

<аргумент> ::= <data>

<тело программы> ::= { <операторы>

<операторы> ::= <оператор> <операторы> | }

<оператор> ::= <идентификатор>:=<инициализатор> \

<инициализатор> ::= <знак> <выражение> | <выражение>

<выражение> ::= ( <операнд> <конец выражения>

<операнд> ::= <iden> | <data>

<конец выражения> ::= <знак> <операнд> <конец выражения> | )

<знак> ::= ! | && | !!

Элементами разработанной грамматики являются:

1) словарь нетерминалов Vn ={фрагмент, идентификатор, аргумент, тело программы, операторы, оператор, инициализатор, выражение, операнд, конец выражения, знак };

2) словарь терминалов Vt = { FUNC, FINAL, !, !!, &&, :=, <iden>, <data>};

3) аксиома грамматики – нетерминальный символ <фрагмент>;

2. Код программы:

import re

index = -1

consts = {}

# декоратор

def decorator\_function(func):

def wrapper(\*args, \*\*kwargs):

try:

global index

index += 1

return func(\*args, \*\*kwargs)

except Exception as e:

return 0

wrapper.\_\_name\_\_ = func.\_\_name\_\_

return wrapper

def read\_consts(name\_file):

consts = {}

with open(name\_file) as file:

for line in file:

key, value, msg = line.split('\t')

msg = msg.replace('\n', '')

consts[key] = {

'code': int(value),

'msg': msg.encode('cp1251').decode('utf-8')

}

return consts

@decorator\_function

def check(data, const):

if data[index] == const['code']:

return 1

else:

print("\t{}".format(const['msg']))

return 0

@decorator\_function

def S\_fragment (data):

global index

index = -1

return S\_identif(data) \*\

check(data, consts['Semicolon']) \* \

check(data, consts['FUNC']) \* \

check(data, consts['Bracket\_open']) \* \

S\_arg(data) \* \

check(data, consts['Bracket\_close']) \* \

check(data, consts['Division']) \* \

S\_telo\_prog(data) \* \

check(data, consts['FINAL'])

@decorator\_function

def S\_identif(data):

if data[index] == consts['Iden']['code']:

return 1

else:

print("\tожидался идентификатор")

return 0

@decorator\_function

def S\_arg(data):

if data[index] == consts['Data']['code']:

return 1

else:

print("\tожидался аргумент")

return 0

@decorator\_function

def S\_telo\_prog(data):

if data[index] == consts['Brace\_open']['code']:

return S\_operatori(data)

else:

print("\tожидалось тело программы")

return 0

@decorator\_function

def S\_operatori(data):

global index

if data[index] == consts['Brace\_close']['code']:

return 1

else:

index -= 1

return S\_operator(data) \* S\_operatori(data)

@decorator\_function

def S\_operator(data):

if data[index] == consts['Iden']['code']:

return check(data, consts['Prisv']) \* \

S\_initcializator(data) \* \

check(data, consts['Division'])

else:

print("\tожидался oператор")

return 0

@decorator\_function

def S\_initcializator(data):

global index

if data[index] == consts['Exclamation\_mark']['code']:

return S\_expression(data)

else:

index -= 1

return S\_expression(data)

@decorator\_function

def S\_expression(data):

global index

if data[index] == consts['Bracket\_open']['code']:

return S\_operand(data) \* S\_end\_expression(data)

else:

print("\tожидалось выражение")

return 0

@decorator\_function

def S\_operand (data):

if data[index] == consts['Iden']['code']:

return 1

elif data[index] == consts['Data']['code']:

return 1

else:

print("\tожидался операнд")

return 0

@decorator\_function

def S\_end\_expression(data):

if data[index] == consts['Exclamation\_mark']['code'] or \

data[index] == consts['Dounbel\_exclamation\_mark']['code'] or \

data[index] == consts['Double\_and']['code']:

return S\_operand(data) \* S\_end\_expression(data)

elif data[index] == consts['Bracket\_close']['code']:

return 1

else:

print("\tожидался конец выражения")

return 0

def main():

global consts, index

consts = read\_consts('D:\data.txt')

test\_data = []

with open('D:/test\_data.txt') as file\_handle:

for line in file\_handle:

test\_data.append([int(x) for x in line.split()])

i = 0

for test in test\_data:

i += 1

print("\nЦепочка №{}:".format(i))

index = -1

try:

if S\_fragment(test) == 0:

print("\tСинтаксическая ошибка")

else: print("\tЦепочка верна")

except Exception as e:

print(str(e))

print("\tСинтаксическая ошибка")

main()

3. Результаты выполнения

|  |  |
| --- | --- |
| **Входная последовательность** | **Результат** |
| 300 501 100 506 400 507 502 504 300 508 506 400 509 400 | Верно |
| 507 502 300 508 503 506 400 510 400 507 502 505 200 | Верно |
| 300 501 100 506 507 | Неверно |
| 100 506 400 507 | Неверно |

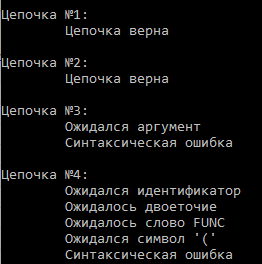


Рисунок 1 – Результат выполнения программы