Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей Кафедра информатики

Дисциплина: Методы трансляции

ОТЧЕТ

к лабораторной работе № 3

на тему

**СИНТАКСИЧЕСКИЙ АНАЛИЗ**

Выполнил:

студент гр. 153503

Кончик Д.С.

Проверил:

Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146836467)

[2 Теоретические сведения 4](#_Toc146836468)

[3 Полученные результаты 5](#_Toc146836469)

[Выводы 8](#_Toc146836470)

[Список использованных источников 9](#_Toc146836471)

[Приложение А (обязательное) листинг кода 10](#_Toc146836472)

# **1 ЦЕЛЬ РАБОТЫ**

Разработать синтаксический анализатор для языка программирования, выбранного в лабораторной работе 1. Необходимо вывести результат синтаксического анализа в виде дерева составляющих, а также обработать возможные синтаксические ошибки.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

На этапе генерации компилятор создает код, который представляет собой набор инструкций, понятных для целевой аппаратной платформы, итоговый файл компилируется в исполняемый файл, который может быть запущен на целевой платформе без необходимости наличия кода.

Фаза эмуляции интерпретатора происходит во время выполнения программы. В отличие от компилятора, интерпретатор работает с кодом напрямую, без предварительной генерации машинного кода.

Лексический анализатор – первый этап трансляции. Лексический анализатор читает поток символов, составляющих исходную программу, и группирует эти символы в лексемы или значащие последовательности. Лексема – это элементарная единица, которая может являться ключевым словом, идентификатором, константным значением. Для каждой лексемы анализатор строит токен, который по сути является кортежем, содержащим имя и значение.[1]

Синтаксический анализатор выясняет, удовлетворяют ли предложения, из которых состоит исходная программа, правилам грамматики языка программирования. Синтаксический анализатор получает на вход результат лексического анализатора и разбирает его в соответствии с грамматикой. Результат синтаксического анализа обычно представляется в виде синтаксического дерева разбора.[2]

Существует несколько видов деревьев разбора, к которым относятся:

– дерево зависимостей;

– дерево составляющих.

Дерево составляющих и дерево синтаксического разбора – это два термина, которые обозначают одно и тоже. Дерево составляющих описывает структура программы на уровне ее синтаксиса, разбивая ее на отдельные синтаксические единицы, например функции, циклы.

Дерево зависимостей в свою очередь помогает понять, какие части программы зависят от других. Дерево зависимостей описывает зависимостимежду компонентами программы и сфокусировано на отношениях между этими компонентами.

Грамматика – набор правил, описывающих, как необходимо формировать из алфавита языка строки, соответствующие синтаксису языка.

# **3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

В результате выполнения лабораторной работы был разработан синтаксический анализатор при помощи метода рекурсивного спуска, который на вход принимает результат лексического анализатора, а в результате отображает синтаксическое дерево разбора.

В качестве входных данных использовался текстовый файл с языком C#(рисунок 1)*.*

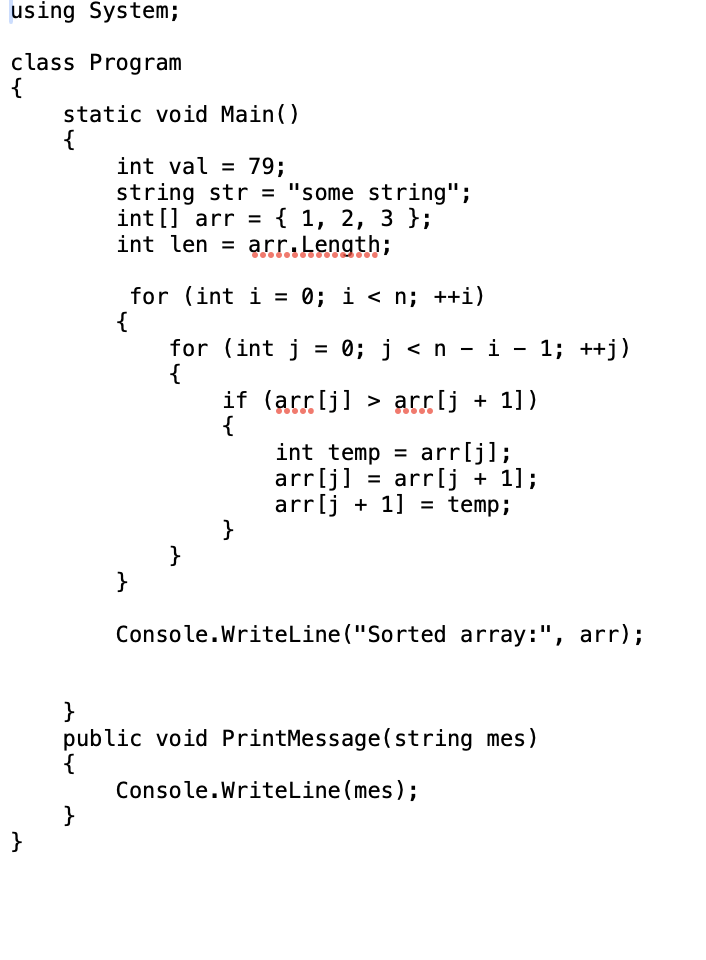
****

Рисунок 1 – Файл с входными данными

В качестве выходных данных было получено синтаксическое дерево разбора (рисунок 2).



Рисунок 2 – Файл с выходными данными

# **ВЫВОДЫ**

В результате выполнения лабораторной работы был разработан синтаксический анализатор подмножества языка программирования, определенного в лабораторной работе 1. Результат был выведен в виде дерева составляющих, а также обработана возможные синтаксические ошибки.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Лексический анализатор [Электронный ресурс]. – Режим доступа: https://csc.sibsutis.ru/sites/csc.sibsutis.ru/files/courses/trans/. – Дата доступа: 18.03.2024.
2. Синтаксический анализатор [Электронный ресурс]. – Режим доступа: https://csc.sibsutis.ru/sites/csc.sibsutis.ru/files/courses/trans/. – Дата доступа: 18.03.2024.

# **ПРИЛОЖЕНИЕ А (обязательное) Листинг кода**

Листинг 1 **–** Файл *main.py*:

import re

from regex import regex\_tokens

from token\_operations import assign\_token\_id, assign\_token\_type, group\_tokens

source = ""

with open('source.txt', 'r') as file:

source = file.read()

ultimate\_regex = re.compile('|'.join('(?P<%s>%s)' % pair for pair in regex\_tokens))

id\_tokens = assign\_token\_id(source, ultimate\_regex)

type\_tokens = assign\_token\_type(source, ultimate\_regex)

grouped\_tokens = group\_tokens(type\_tokens)

for i in grouped\_tokens:

print('\n\n', i, grouped\_tokens[i])

id\_tokens = {value: key for key, value in id\_tokens.items()}

for comment in grouped\_tokens['COMMENTS']:

source = source.replace(comment, id\_tokens[comment])

for multicomment in grouped\_tokens['MULTICOMMENTS']:

source = source.replace(multicomment, id\_tokens[multicomment])

for error\_1 in grouped\_tokens['ERROR1']:

source = source.replace(error\_1, id\_tokens[error\_1])

grouped\_tokens['ERROR2'] = sorted(grouped\_tokens['ERROR2'], reverse=True)

for error\_2 in grouped\_tokens['ERROR2']:

source = source.replace(error\_2, id\_tokens[error\_2])

for error\_3 in grouped\_tokens['ERROR3']:

source = source.replace(error\_3, id\_tokens[error\_3])

for error\_4 in grouped\_tokens['ERROR4']:

source = source.replace(error\_4, id\_tokens[error\_4])

for error\_5 in grouped\_tokens['ERROR5']:

source = source.replace(error\_5, id\_tokens[error\_5])

for error\_6 in grouped\_tokens['ERROR6']:

source = source.replace(error\_6, id\_tokens[error\_6])

id\_tokens = {value: key for key, value in id\_tokens.items()}

for key, value in id\_tokens.items():

source = source.replace(value, key)

print(source)

for key, value in id\_tokens.items():

print(key, value)

Листинг 2 **–** Файл *regex.py*:

regex\_tokens = [

('COMMENT', r'//.\*'),

('MULTICOMMENT', r'/\\*[\s\S]\*?\\*/'),

('WHITESPACE',r'\s+'),

('ERROR1', r'\b(?:class|int|float|double|string|char|bool|object)\b\s\*[\d!@#=%^&\*~\_$][a-zA-Z\_]\*\w\*\s\*='),

('ERROR2', r'(\w+\s\*(\+|-){3,})|((\+|-){3,}\s\*\w+)|;\s\*([+-]\s\*\w+)|(\w+\s\*[+-]);'),

('ERROR3', r'\b(int|float|double|string|bool|char|object)\b\s+\b(int|float|double|string|bool|char|object)\b\s\*=.\*;'),

('ERROR4', r'\b(string|char)\s+\w+\s\*=\s\*[\d]+\s\*([+\-\*/><~&|^]\s\*[\d]+\s\*)+;'),

('ERROR5', r'\b(bool)\s+\w+\s\*=\s\*[\d]+\s\*([&^|%+\-\*/]\s\*[\d]+\s\*)+;'),

('ERROR6', r'\b(int|float|double|decimal)\s+\w+\s\*=\s\*[\d]+\s\*(((&&)|(\|\|)|([><])|(!=))\s\*[\d]+\s\*)+;'),

('STRING', r'["\'](?:\\.|[^"\\\'\n])\*["\']'),

('NUMBER', r'\b(?:\d{1,3}(?:\_\d{3})\*(?:\.\d+)?|\d+(?:\.\d+)?(?:[eE][-+]?\d+)?)(?:[dfmDFM])?\b'),

('KEYWORD', r'\b(?:class|struct|var|int|float|double|decimal|char|string|public|private|protected|static|foreach|do|if|else|for|while|switch|case|return|void|bool|true|false|null|new|object|using|namespace|default|break|not|in)\b'),

('IDENTIFIER',r'\b[A-Za-z\_]\w\*\b'),

('OPERATOR', r'[\+\-\\*/=%<>^~!&:.|?]+'),

('DELIMITER', r'[{}()\[\],;]')

]

Листинг 2 **–** Файл *token\_operations.py*:

def assign\_token\_id(code, ultimate\_regex):

tokens = {}

i = 1

for matche in ultimate\_regex.finditer(code):

token\_type = matche.lastgroup

token\_value = matche.group(token\_type)

if token\_type not in ['WHITESPACE', 'DELIMITER', 'OPERATOR']:

values = tokens.values()

if token\_value not in values:

key = f"id{i}"

tokens[key] = token\_value

i += 1

return tokens

def assign\_token\_type(code, ultimate\_regex):

type\_tokens = []

for matche in ultimate\_regex.finditer(code):

token\_type = matche.lastgroup

token\_value = matche.group(token\_type)

if token\_type not in ['WHITESPACE']:

type\_tokens.append((token\_type, token\_value))

return type\_tokens

def group\_tokens(tokens):

groups = {

'COMMENTS': [],

'MULTICOMMENTS': [],

'ERROR1': [],

'ERROR2': [],

'ERROR3': [],

'ERROR4': [],

'ERROR5': [],

'ERROR6': [],

'KEYWORDS': [],

'IDENTIFIERS': [],

'OPERATORS': [],

'NUMBERS': [],

'STRINGS': [],

'DELIMITERS': []

}

for token\_type, token\_value in tokens:

if token\_type == 'COMMENT':

groups['COMMENTS'].append(token\_value)

elif token\_type == 'MULTICOMMENT':

groups['MULTICOMMENTS'].append(token\_value)

elif token\_type == 'ERROR1':

groups['ERROR1'].append(token\_value)

elif token\_type == 'ERROR2':

groups['ERROR2'].append(token\_value)

elif token\_type == 'ERROR3':

groups['ERROR3'].append(token\_value)

elif token\_type == 'ERROR4':

groups['ERROR4'].append(token\_value)

elif token\_type == 'ERROR5':

groups['ERROR5'].append(token\_value)

elif token\_type == 'ERROR6':

groups['ERROR6'].append(token\_value)

elif token\_type == 'KEYWORD':

groups['KEYWORDS'].append(token\_value)

elif token\_type == 'IDENTIFIER':

groups['IDENTIFIERS'].append(token\_value)

elif token\_type == 'OPERATOR':

groups['OPERATORS'].append(token\_value)

elif token\_type == 'NUMBER':

groups['NUMBERS'].append(token\_value)

elif token\_type == 'STRING':

groups['STRINGS'].append(token\_value)

elif token\_type == 'DELIMITER':

groups['DELIMITERS'].append(token\_value)

return groups