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

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 зависимости между 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).

```
using System;

class Program
{
    static void Main()
    {
        int val = 79;
        string str = "some string";
        int[] arr = { 1, 2, 3 };
        int len = arr.Length;

         for (int i = 0; i < n; ++i)
        {
            for (int j = 0; j < n - i - 1; ++j)
            {
                if (arr[j] > arr[j + 1])
                {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }

        Console.WriteLine("Sorted array:", arr);


    }
    public void PrintMessage(string mes)
    {
        Console.WriteLine(mes);
    }
}
```

Рисунок 1 – Файл с входными данными

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

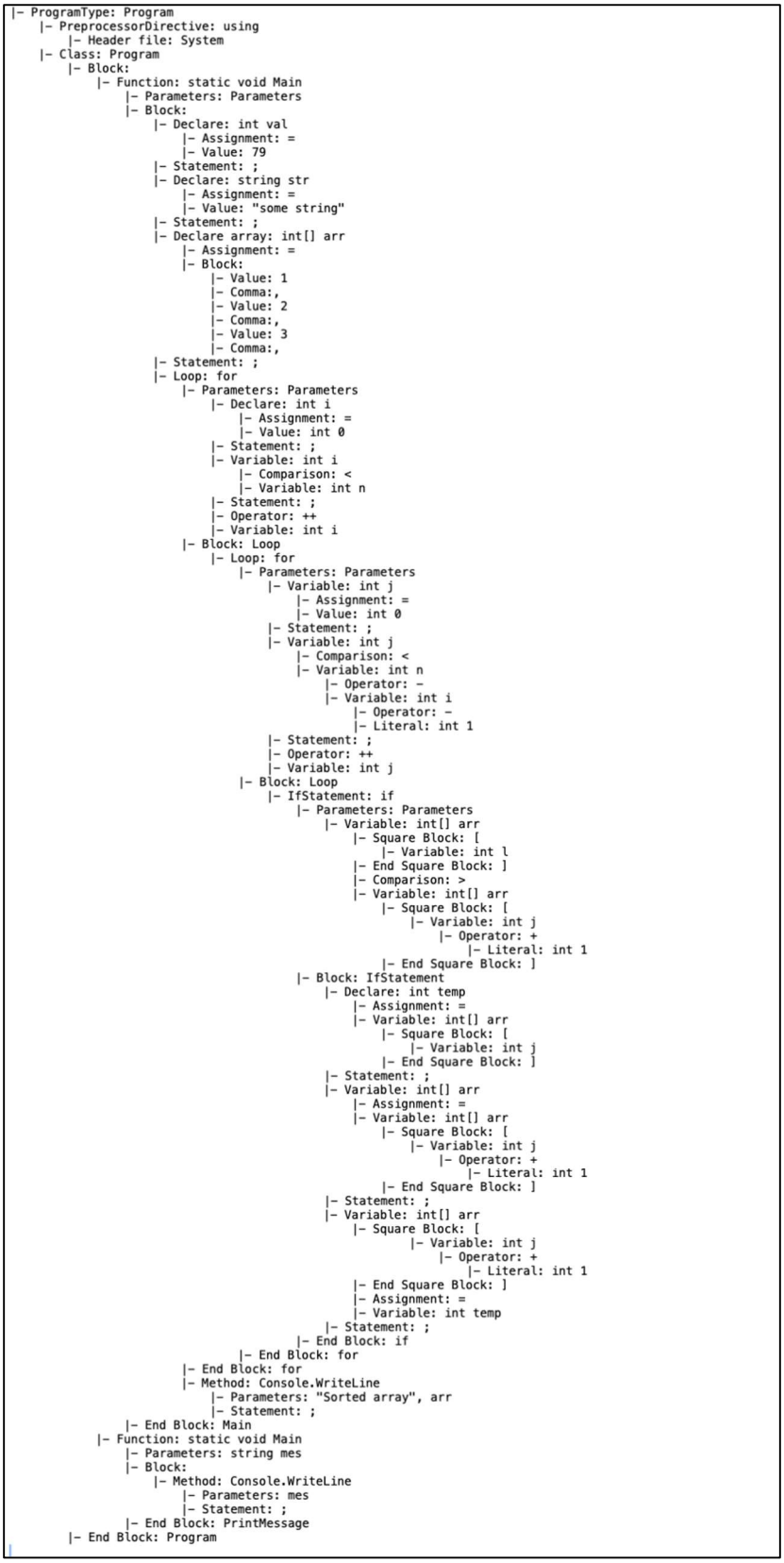


Рисунок 2 – Файл с выходными данными

# 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|p
rotected|static|foreach|do|if|else|for|while|switch|case|return|void|bool|tru
e|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
```