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 №2

на тему

**ЛЕКСИЧЕСКИЙ АНАЛИЗ**

Студент Д. С. Кончик

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc160624420)

[2 Теоретические сведения 4](#_Toc160624421)

[3 Результат выполнения 5](#_Toc160624422)

[Заключение 8](#_Toc160624423)

[Список использованных источников 9](#_Toc160624424)

[Приложение А (обязательное) Листинг кода 10](#_Toc160624425)

1. **ЦЕЛЬ РАБОТЫ**

Разработать лексический анализатор для определенного подмножества языка программирования, который был указан в первой лабораторной работе. Задать лексические правила и создать механизм преобразования последовательности символов в поток лексем (токенов). В случае обнаружения некорректной последовательности символов, предусмотреть механизм обнаружения ошибки и выдачи соответствующего сообщения.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Лексический анализатор выполняет основную функцию – он анализирует символы, вводимые из исходного текста программы, и группирует их в лексемы, которые затем преобразуются в последовательность токенов. Эта последовательность токенов передается на вход синтаксическому анализатору. Токен представляет собой структуру, содержащую имя токена и набор дополнительных атрибутов. Имя токена определяет тип лексической единицы, такой как ключевое слово или идентификатор переменной [1].

Лексема представляет собой последовательность символов из исходного кода программы, которая идентифицируется лексическим анализатором как экземпляр токена. Шаблон описывает формат, который может принимать лексема токена, и на его основе лексический анализатор определяет, соответствует ли данная последовательность символов какому-либо токену.

Лексемы могут представлять различные элементы программного кода, такие как идентификаторы, ключевые слова, операторы, константы и разделители. Каждая лексема имеет свое семантическое значение и правила использования в контексте конкретного языка программирования.

Идентификаторы используются для уникальной идентификации элементов программы, их объявления и использования в коде. Они могут содержать буквы, цифры и специальные символы, в зависимости от требований языка программирования [2].

Ключевые слова – это зарезервированные слова с особым значением в языке программирования, используемые для определения специальных действий и операций в программе.

Операторы представляют собой символы или их комбинации, выполняющие определенные операции или выражающие отношения между значениями.

Константы – это фиксированные значения, которые не могут изменяться во время выполнения программы.

Разделители используются для разделения элементов программы и обозначения границ различных конструкций.

Лексемы играют ключевую роль в создании структуры и синтаксической правильности кода в контексте языка программирования. Компиляторы и интерпретаторы используют лексемы для анализа и выполнения программного кода [3].

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В ходе лабораторной работы был создан лексический анализатор, который осуществляет анализ текстовых файлов на языке *C#*. В процессе работы были определены лексические правила и выявлены различные типы ошибок. На вход программы подается текстовый файл *(input.txt)*, содержащий строки символов анализируемой программы (рисунок 1).

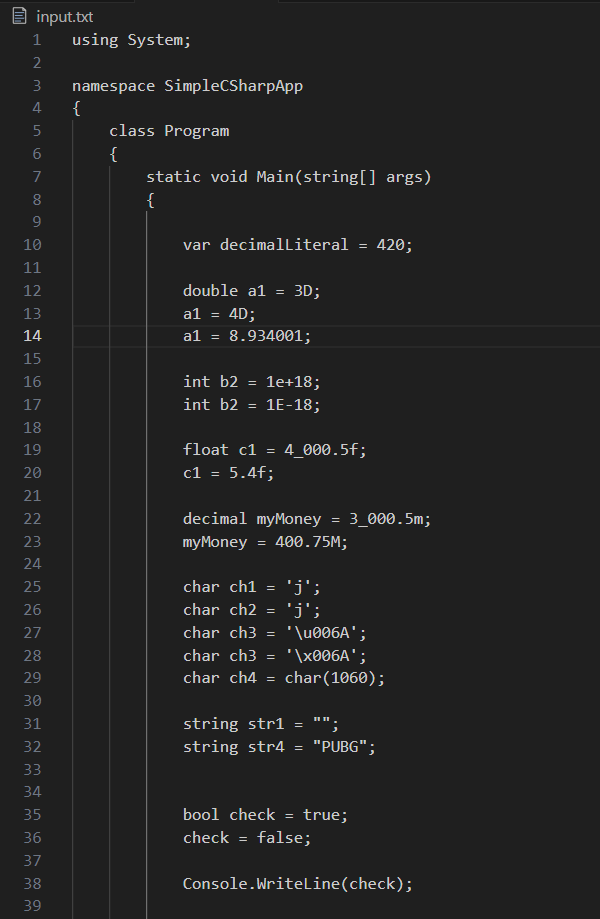


Рисунок 1 – Файл *input.txt*

Результатом является файл *categorized\_tokens.json*, который содержит список лексем, разделенные на группы (рисунок 2).

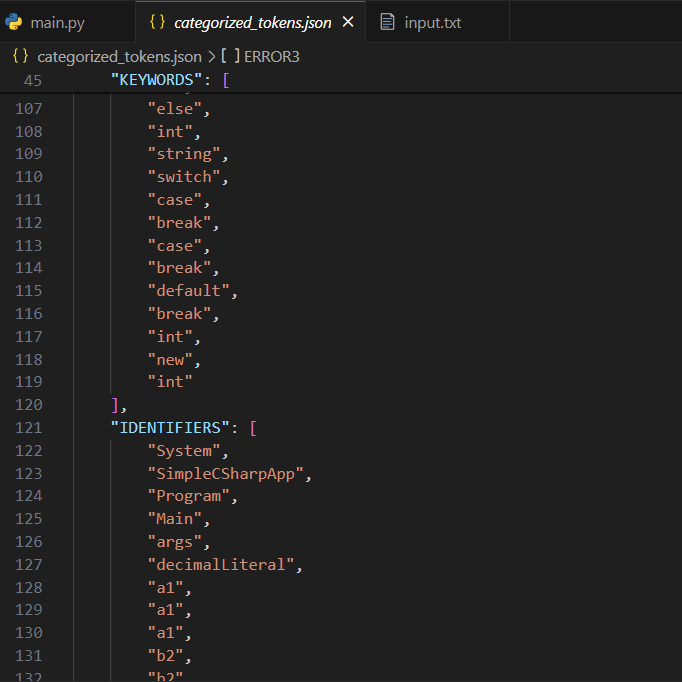


Рисунок 2 – Файл с лексемами

Также в файле *ids.txt* находится список соответствия идентификаторов лексемам.

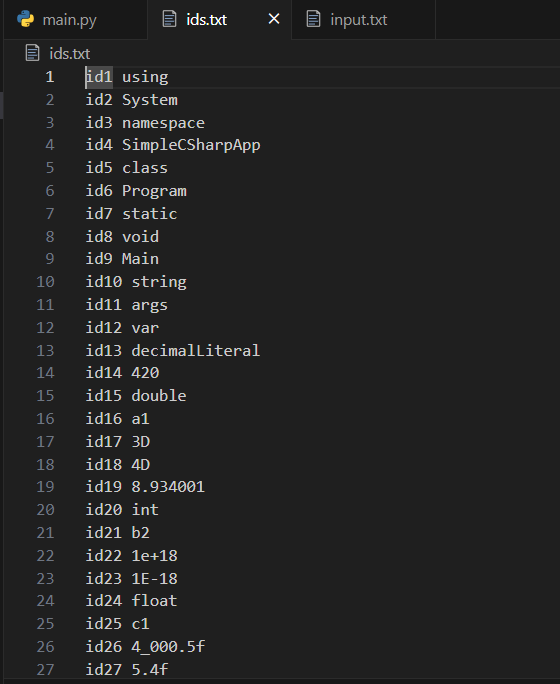


Рисунок 3 – Файл с идентификаторами

Файл *output.txt* представляет собой исходный файл с заменой лексем на их идентификаторы (рисунок 4).

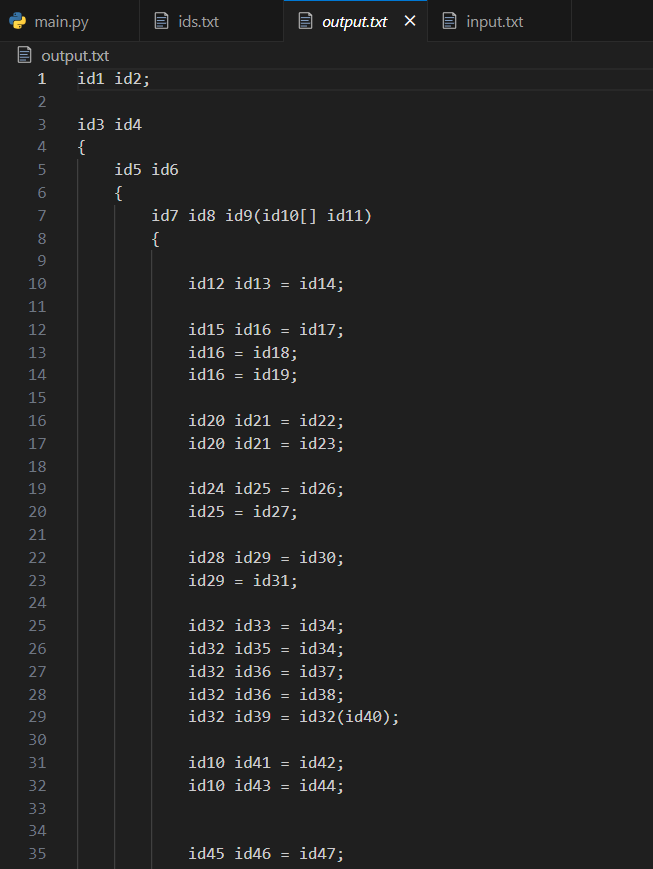


Рисунок 4 – Файл *output.txt*

# ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы был создан инструмент лексического анализа для определенного подмножества языка программирования *(C#)*, который был описан в первой лабораторной работе. Были определены правила для разбора лексем из потока символов, и осуществлен перевод этого потока в поток лексем (токенов). В случае обнаружения недопустимой последовательности символов программа сообщает об ошибке.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лексический анализ [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Лексический\_анализ.
2. Лексический анализ. Токены, шаблоны, лексемы [Электронный ресурс]. – Режим доступа: https://wiki.livid.pp.ru/students/sp/lectures/3.html.
3. Что такое лексема языка программирования [Электронный ресурс]. – Режим доступа: https://medium.com/@Omg\_wolfcii37/что-такое-лексема-языка-программирования-8ab35e951762.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – *main.py*

import re

import json

sample\_code = ""

try:

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

        sample\_code = file.read()

except Exception:

    print("Error reading from file")

token\_specs = [

    ('COMMENT',   r'//.\*'),                      # Single-line comment

    ('MULTICOMMENT', r'/\\*[\s\S]\*?\\*/'),         # Multi-line comment

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

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

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

    ('ERROR3', r'\b(int|float|double|string|bool|char|object)\b\s+\b(int|float|double|string|bool|char|object)\b\s\*='), # data types as vars

    ('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])\*["\']'),         # String literal

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

    ('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'), # Keywords

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

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

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

]

# Compile the regular expressions

token\_regex = '|'.join('(?P<%s>%s)' % pair for pair in token\_specs)

# print(token\_regex)

token\_re = re.compile(token\_regex)

token\_dict = {}

# Tokenization function

def tokenize\_enhanced\_csharp\_code(code):

    tokens = []

    i = 1

    for m in token\_re.finditer(code):

        token\_type = m.lastgroup

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

        if token\_type in ['STRING', 'NUMBER', 'KEYWORD', 'IDENTIFIER']:

            values = token\_dict.values()

            if token\_value not in values:

                key = f"id{i}"

                token\_dict[key] = token\_value

                i += 1

        if token\_type not in ['WHITESPACE', 'COMMENT', 'MULTICOMMENT']:

            tokens.append((token\_type, token\_value))

    return tokens

enhanced\_tokens = tokenize\_enhanced\_csharp\_code(sample\_code)

def categorize\_tokens(tokens):

    categories = {

        'ERROR1': [],

        'ERROR2': [],

        'ERROR3': [],

        'ERROR4': [],

        'ERROR5': [],

        'ERROR6': [],

        'KEYWORDS': [],

        'IDENTIFIERS': [],

        'OPERATORS': [],

        'NUMBERS': [],

        'STRINGS': [],

        'DELIMITERS': []

    }

    for token\_type, token\_value in tokens:

        if token\_type == 'ERROR1':

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

        elif token\_type == 'ERROR2':

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

        elif token\_type == 'ERROR3':

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

        elif token\_type == 'ERROR4':

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

        elif token\_type == 'ERROR5':

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

        elif token\_type == 'ERROR6':

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

        elif token\_type == 'KEYWORD':

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

        elif token\_type == 'IDENTIFIER':

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

        elif token\_type == 'OPERATOR':

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

        elif token\_type == 'NUMBER':

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

        elif token\_type == 'STRING':

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

        elif token\_type == 'DELIMITER':

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

    return categories

# Use the categorize\_tokens function on the enhanced tokens

categorized\_tokens = categorize\_tokens(enhanced\_tokens)

with open('categorized\_tokens.json', 'w') as json\_file:

    json.dump(categorized\_tokens, json\_file, indent=4)

for key, value in token\_dict.items():

    sample\_code = sample\_code.replace(value, key)

with open('output.txt', 'w') as output\_file:

    output\_file.write(sample\_code)

with open('ids.txt', 'w') as ids\_file:

    for key, value in token\_dict.items():

        ids\_file.write(f"{key} {value}\n")