

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
Преподаватель

Д. С. Кончик
Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

1 Цель работы	3
2 Теоретические сведения	4
3 Результат выполнения	5
Заключение	8
Список использованных источников	9
Приложение А (обязательное) Листинг кода	10

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.

2 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.

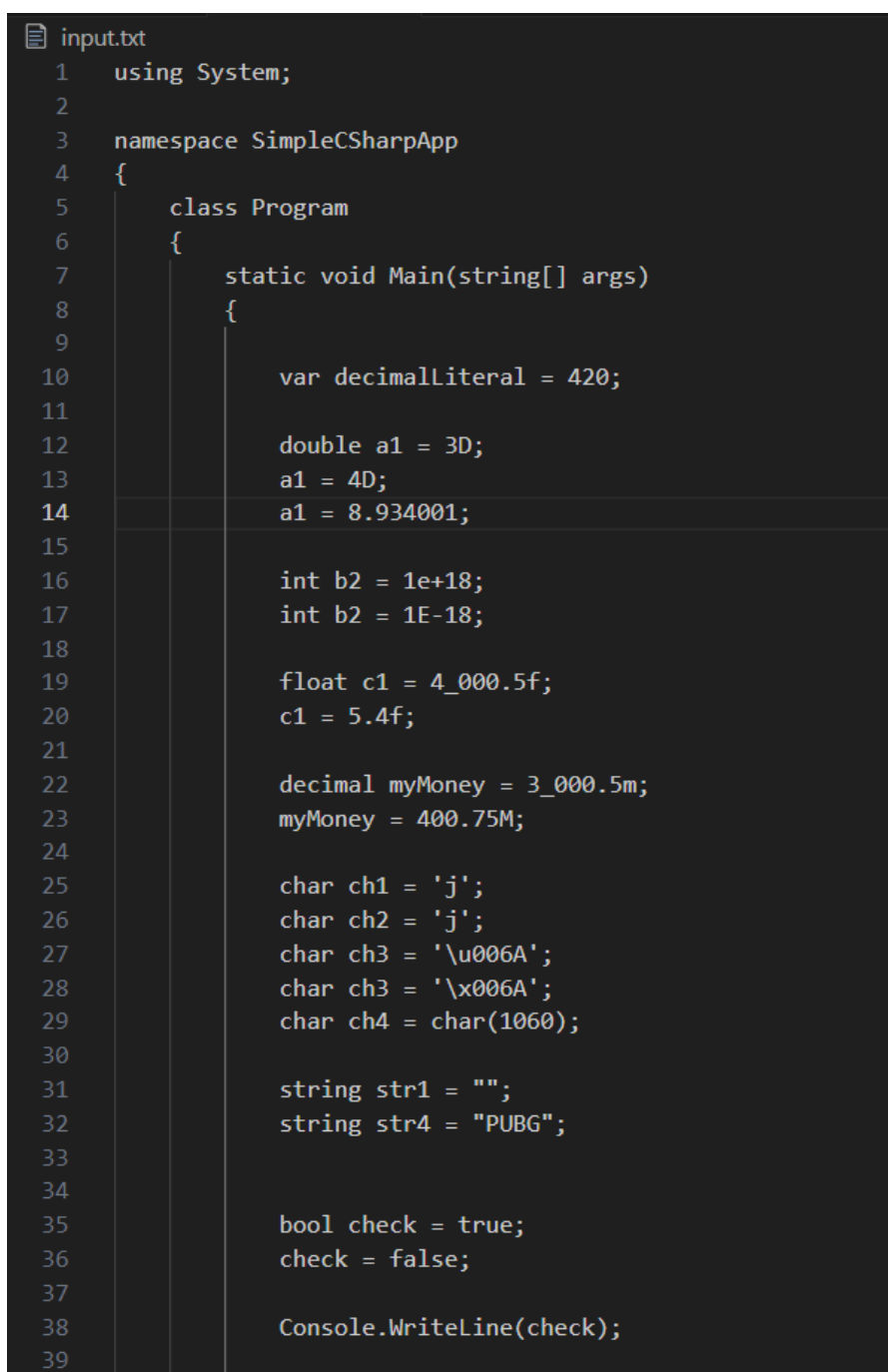
Константы – это фиксированные значения, которые не могут изменяться во время выполнения программы.

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].

3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

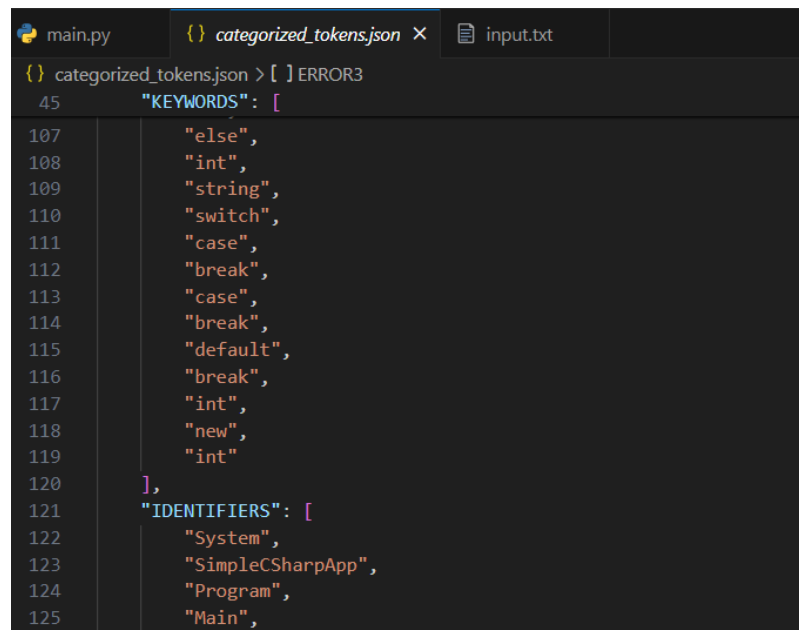
В ходе лабораторной работы был создан лексический анализатор, который осуществляет анализ текстовых файлов на языке C#. В процессе работы были определены лексические правила и выявлены различные типы ошибок. На вход программы подается текстовый файл (*input.txt*), содержащий строки символов анализируемой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



```
input.txt
1  using System;
2
3  namespace SimpleCSharpApp
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9
10             var decimalLiteral = 420;
11
12             double a1 = 3D;
13             a1 = 4D;
14             a1 = 8.934001;
15
16             int b2 = 1e+18;
17             int b2 = 1E-18;
18
19             float c1 = 4_000.5f;
20             c1 = 5.4f;
21
22             decimal myMoney = 3_000.5m;
23             myMoney = 400.75M;
24
25             char ch1 = 'j';
26             char ch2 = 'j';
27             char ch3 = '\u006A';
28             char ch3 = '\x006A';
29             char ch4 = char(1060);
30
31             string str1 = "";
32             string str4 = "PUBG";
33
34
35             bool check = true;
36             check = false;
37
38             Console.WriteLine(check);
39
```

Рисунок 1 – Файл *input.txt*

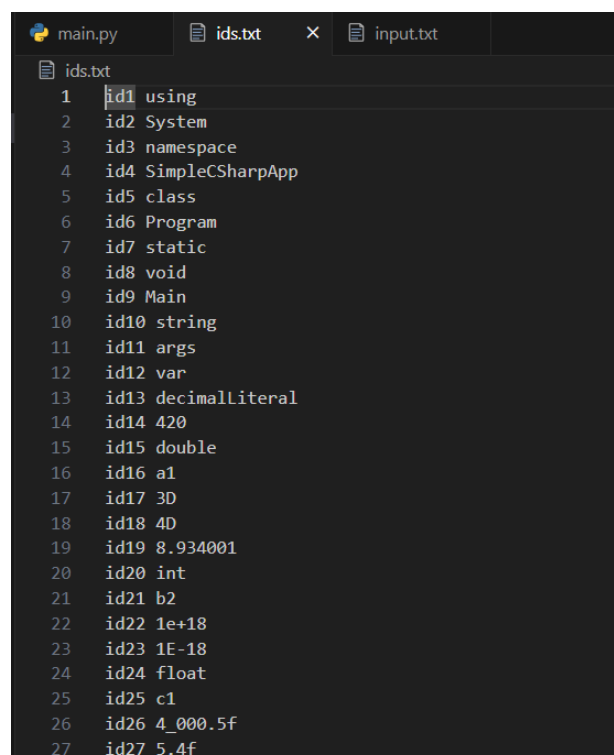
Результатом является файл *categorized_tokens.json*, который содержит список лексем, разделенные на группы (рисунок 2).



```
{ } categorized_tokens.json > [ ] ERROR3
45     "KEYWORDS": [
107         "else",
108         "int",
109         "string",
110         "switch",
111         "case",
112         "break",
113         "case",
114         "break",
115         "default",
116         "break",
117         "int",
118         "new",
119         "int"
120     ],
121     "IDENTIFIERS": [
122         "System",
123         "SimpleCSharpApp",
124         "Program",
125         "Main",
```

Рисунок 2 – Файл с лексемами

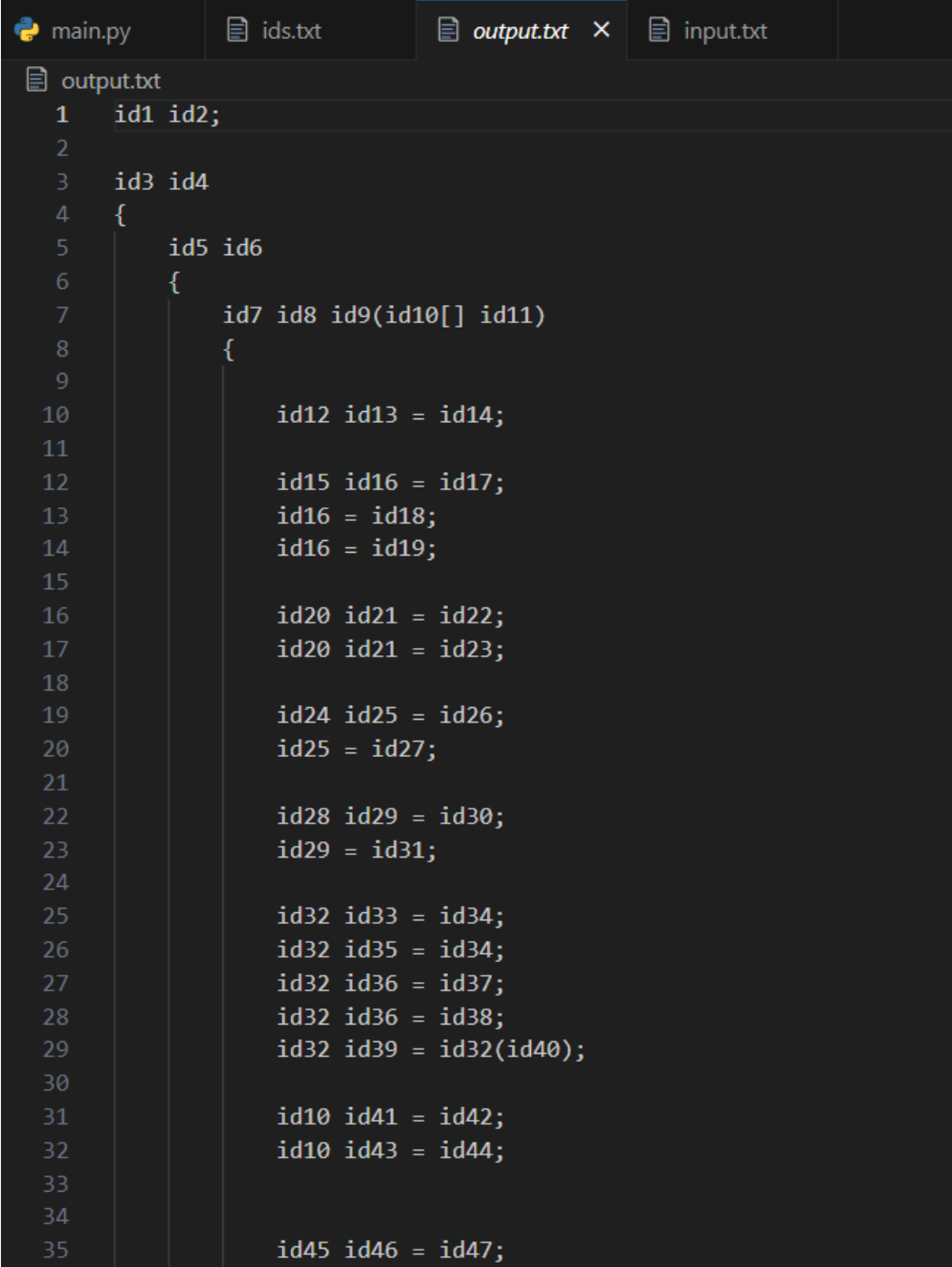
Также в файле *ids.txt* находится список соответствия идентификаторов лексемам.



```
ids.txt
1 id1 using
2 id2 System
3 id3 namespace
4 id4 SimpleCSharpApp
5 id5 class
6 id6 Program
7 id7 static
8 id8 void
9 id9 Main
10 id10 string
11 id11 args
12 id12 var
13 id13 decimalLiteral
14 id14 420
15 id15 double
16 id16 a1
17 id17 30
18 id18 40
19 id19 8.934001
20 id20 int
21 id21 b2
22 id22 1e+18
23 id23 1E-18
24 id24 float
25 id25 c1
26 id26 4_000.5f
27 id27 5.4f
```

Рисунок 3 – Файл с идентификаторами

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



```
1  id1 id2;  
2  
3  id3 id4  
4  {  
5      id5 id6  
6      {  
7          id7 id8 id9(id10[] id11)  
8          {  
9  
10             id12 id13 = id14;  
11  
12             id15 id16 = id17;  
13             id16 = id18;  
14             id16 = id19;  
15  
16             id20 id21 = id22;  
17             id20 id21 = id23;  
18  
19             id24 id25 = id26;  
20             id25 = id27;  
21  
22             id28 id29 = id30;  
23             id29 = id31;  
24  
25             id32 id33 = id34;  
26             id32 id35 = id34;  
27             id32 id36 = id37;  
28             id32 id36 = id38;  
29             id32 id39 = id32(id40);  
30  
31             id10 id41 = id42;  
32             id10 id43 = id44;  
33  
34  
35             id45 id46 = id47;
```

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|re
turn|void|bool|true|false|null|new|object|using|namespace|default|break|not|i
n)\\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")

```