

Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы трансляции

ОТЧЁТ  
по лабораторной работе  
на тему

Лексический анализ

Выполнил  
Студент гр. 053502  
Шаргородский И.С.

Проверил  
Ассистент кафедры информатики  
Гриценко Н.Ю.

Минск 2023

## СОДЕРЖАНИЕ

1 Цель работы .....	3
2 Краткие теоретические сведения.....	4
3 Виды токенов лексического анализатора .....	5
4 Демонстрация работы лексического анализатора .....	6
4.1 Результаты работы .....	6
4.2 Лексические ошибки.....	8
5 Выводы .....	10
Приложение А (информационное)_Код программ.....	11

## **1 ЦЕЛЬ РАБОТЫ**

Освоение работы с существующими лексическими анализаторами (по желанию). Разработка лексического анализатора подмножества языка программирования, определенного в лабораторной работе 1. Определяются лексические правила. Выполняется перевод потока символов в поток лексем (токенов).

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Лексический анализ – это первый этап в теории трансляции, на котором исходный код программы преобразуется в последовательность токенов. Токены – это независимые элементы программы, такие как идентификаторы, ключевые слова, символы операций и т.д.

Цель лексического анализа – разбить исходный код на единицы информации, которые можно использовать для дальнейшей обработки. Для этого используется лексер, который сканирует исходный код и разбивает его на токены.

Результатом работы лексического анализа является последовательность токенов, которая подается на вход для дальнейшей обработки, такой как синтаксический анализ. Корректность лексического анализа определяет, насколько хорошо программа может быть обработана дальше. Если лексер обнаруживает ошибку, такую как недопустимый символ или неизвестный идентификатор, он генерирует ошибку лексического анализа.

Одним из важных аспектов лексического анализа является его эффективность. Лексер должен работать быстро, так как этот этап является одним из самых длительных в процессе компиляции. Поэтому разработчики обычно используют алгоритмы, такие как автоматы или грамматические анализаторы, чтобы улучшить эффективность лексического анализа.

В целом, лексический анализ играет ключевую роль в теории трансляции, поскольку он позволяет преобразовать исходный код в формат, который может быть легко обработан дальше. Корректный и эффективный лексический анализ также позволяет сохранять сведения о токенах и их атрибутах, что может быть полезно для дальнейшей обработки и отладки кода.

### 3 ВИДЫ ТОКЕНОВ ЛЕКСИЧЕСКОГО АНАЛИЗТОРА

Для выбранного подмножества языка можно выделить следующие виды токенов:

- Токены, соответствующие переменным, например, `variable1`.
- Токены, соответствующие целым числам, например, `123`.
- Токены, соответствующие числам с плавающей запятой, например, `5.5`.
- Токены, соответствующий строковым литералам, например, `“Hello world!”`.
- Токены, соответствующие ключевым словам языка python: `and, as, None, assert, pass, break, return, class, def, or, continue, not, lambda, del, raise, elif, except, else, False, finally, for, from, global, if, import, in, is, nonlocal, True, yield, while, try, with`.
- Токены, соответствующие built-in типам данных python: `float, bool, bytes, bytearray, complex, frozenset, dict, int, list, set, str, tuple`.
- Токены соответствующие built-in функциям python: `abs, all, bytes, setattr, bytearray, divmod, exec, bool, any, ascii, bin, int, setattr, dict, max, callable, dir, chr, classmethod, compile, min, filter, complex, float, enumerate, eval, range, format, locals, frozenset, str, getattr, globals, hasattr, tuple, map, hash, staticmethod, object, help, hex, ord, id, round, input, isinstance, issubclass, iter, len, list, memoryview, sum, sorted, print, next, open, oct, pow, property, repr, reversed, set, slice, super, type, vars, zip`.
- Токены, соответствующие операторам: `+, *=, %, -, **, *, /, //, ==, !=, |, <, |=, <=, ~, >, >=, /=, &, ^, <<, >>, <<=, =, +=, -=, //=, %=, **=, &=, ^=, >>=`.

## 4 ДЕМОНСТРАЦИЯ РАБОТЫ

### 4.1 Результаты работы

Рассмотрим результат лексического анализа тестовой программы (см. приложение А) программой-анализатором:

- Демонстрация используемых ключевых слов представлена на рисунке 1.

```
-----  
Key words:  
-----  
    if  
    elif  
    else  
    for  
    in  
-----
```

Рисунок 1 – Таблица ключевых слов

- Демонстрация используемых констант представлена на рисунке 2.

```
-----  
Constants:  
-----  
    2  
    0  
    1  
-----
```

Рисунок 2 – Таблица констант

- Демонстрация используемых строковых констант представлена на рисунке 3.

```
-----  
Literals:  
-----  
Enter a number:  
Sorry, factorial does not exist for negative numbers  
The factorial of 0 is 1  
The factorial of  
is  
-----
```

Рисунок 3 – Таблица литералов

- Демонстрация используемых операторов представлена на рисунке 4.

Operators:
=
+
*
<
==

Рисунок 4 – Таблица операторов

- Демонстрация используемых переменных представлена на рисунке 5.

Variables:
num
factorial
i

Рисунок 5 – Таблица переменных

- Демонстрация используемых встроенных функций представлена на рисунке 6.

Built-in
int
input
print
range

Рисунок 6 – Таблица встроенных функций

- Демонстрация используемых разделителей представлена на рисунке 7.

Delimiters
(
)
:
,

Рисунок 7 – Таблица разделителей

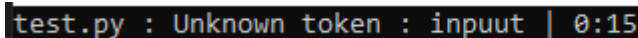
Видно, что каждый токен имеет ряд свойств:

- Имя токена.
- Категория токена.
- Позиция в исходном файле – [строка: колонка].

## 4.2 Лексические ошибки

Ошибка незнакомого токена – производится, когда лексер встречает слово, которое не является переменной и не определено в языке программирования. Результат анализа ошибки представлен на рисунке 6. Входная программа:

```
num = int(inpuut('Enter a number : '))
```

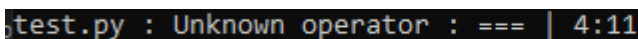


```
test.py : Unknown token : inpuut | 0:15
```

Рисунок 6 – Пример ошибки незнакомого токена

Ошибка незнакомого оператора – производится, когда лексер встречает слово, состоящее из операторных символов, но которой не зарегистрировано. Результат анализа ошибки представлен на рисунке 7. Входная программа:

```
elif num === 0:
```

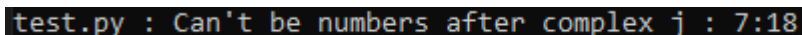


```
test.py : Unknown operator : === | 4:11
```

Рисунок 7 – Пример ошибки незнакомого оператора

Ошибка неправильного комплексного числа – производится, когда лексер встречает число, которое содержит цифры после мнимой единицы. Результат анализа ошибки представлен на рисунке 8. Входная программа:

```
for i in range(1j1,number + 1.1.2j):
```



```
test.py : Can't be numbers after complex j : 7:18
```

Рисунок 8 – Пример ошибки незнакомого токена

Ошибка неправильного дробного числа – производится, когда лексер встречает число, которое содержит несколько точек. Результат анализа ошибки представлен на рисунке 9. Входная программа:

```
for i in range(1j1,number + 1.1.2j):
```



```
test.py : Can't ave more than one dot in number : 7:32
```

Рисунок 9 – Пример ошибки незнакомого токена

Ошибка незакрытых кавычек в литерале – производится, когда лексер встречает литерал, который открывается, но не закрывается. Результат анализа ошибки представлен на рисунке 10. Входная программа: “

```
print("The factorial of", num, "is", factorial)
```

```
test.py : Braces are opend, but never closed : 10:61
```

Рисунок 10 – Пример ошибки незнакомого токена

## **5 ВЫВОДЫ**

Таким образом, в ходе лабораторной работы было изучено понятие лексического анализа в теории трансляции. Было создано приложение-лексер для выбранного подмножества языка. В процессе работы были исследованы способы определения лексических единиц, особенности их хранения и обработки. Было замечено, что для лексического анализа эффективно использовать сканирование исходного текста посимвольно.

В результате был получен ценный практический и теоретический опыт работы с лексическим анализом и созданием лексеров. Это знание может быть полезным для дальнейшей работы с компиляторами и интерпретаторами языков программирования, а также для разработки собственных лексических анализаторов для других языков программирования.

## **ПРИЛОЖЕНИЕ А** **(информационное)** **Код программ**

### **1. Тестовая программа с ошибками**

```
num = int(input("Enter a number: "))
factorial = 1
if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    for i in range(1,num + 1.1.2j):
        factorial = factorial*ibragim
    print("The factorial of", num, "is", factorial)
```

### **2. Тестовая программа**

```
num = int(input("Enter a number: "))
factorial = 1
if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    for i in range(1,number + 1):
        factorial = factorial*ibragim
    print("The factorial of", num, "is", factorial)
```