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

Синтаксический анализ

Выполнил

Студент гр. 053502

Шаргородский И.С.

Проверил

Ассистент кафедры информатики

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc129978528)

[2 Краткие теоретические сведения 4](#_Toc129978529)

[3 Виды токенов лексического анализатора 5](#_Toc129978530)

[4 Демонстрация работы лексического анализатора 6](#_Toc129978531)

[4.1 Результаты работы 6](#_Toc129978532)

[4.2 Лексические ошибки 7](#_Toc129978533)

[5 Выводы 9](#_Toc129978534)

Приложение А (информационное) [Код программ 10](#_Toc129978535)

**1 ЦЕЛЬ РАБОТЫ**

Освоение работы с существующими синтаксическими анализаторами. Разработать свой собственный синтаксический анализатор, выбранного подмножества языка программирования.

Построить синтаксическое дерево.

Определить минимум 4 возможных синтаксических ошибки и показать их корректное выявление.

Основной целью работы является написание сценариев, которые задают синтаксические правила для выбранного подмножества языка.

1. **КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Синтаксический анализ — это процесс сопоставления линейной последовательности лексем (слов, токенов) естественного или формального языка с его формальной грамматикой. Результатом обычно является дерево разбора (синтаксическое дерево). Обычно применяется совместно с лексическим анализом.

Синтаксический анализатор — это программа или часть программы, выполняющая синтаксический анализ.

Как правило, результатом синтаксического анализа является синтаксическое строение предложения, представленное либо в виде дерева зависимостей, либо в виде дерева составляющих, либо в виде некоторого сочетания первого и второго способов представления.

Таким образом на основе анализа выражений, состоящих из литералов, операторов и круглых скобок выполняется группирование токенов исходной программы в грамматические фразы, используемые для синтеза вывода.

1. **ВИДЫ ТОКЕНОВ ЛЕКСИЧЕСКОГО АНАЛИЗВТОРА**

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 запяятой, 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 buit-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, delattr, 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: +, \*=, %, -, \*\*, \*, /, //, ==, !=, |, <, |=, <=, ~, >, >=, /=, &, ^, <<, >>, <<=, =, +=, -=, //=, %=, \*\*=, &=, ^=, >>=.

1. **ДЕМОНСТРАЦИЯ РАБОТЫ**
   1. **Результаты работы**

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

* Демонстрация используемых ключевых слов представлена на рисунке 1.

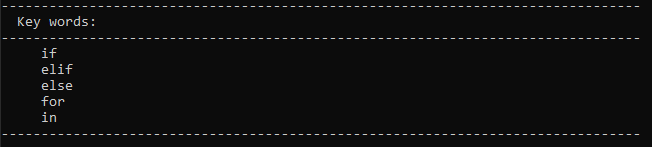


Рисунок 1 – Таблица ключевых слов

* Демонстрация используемых констант представлена на рисунке 2.

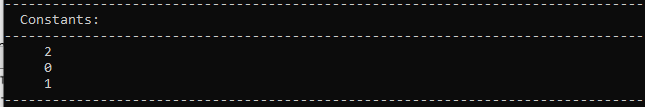


Рисунок 2 – Таблица констант

* Демонстрация используемых строковых констант представлена на рисунке 3.

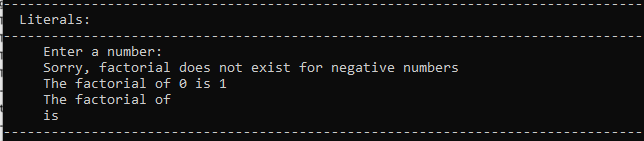


Рисунок 3 – Таблица литералов

* Демонстрация используемых операторов представлена на рисунке 4.

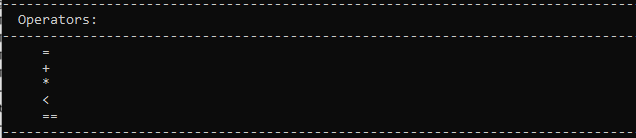


Рисунок 4 – Таблица операторов

* Демонстрация используемых переменных представлена на рисунке 5.

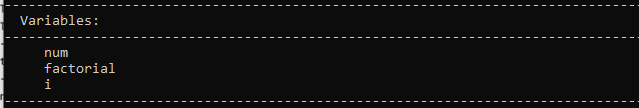


Рисунок 5 – Таблица переменных

Видно, что каждый токен имеет ряд свойств:

* Имя токена.
* Категория токена.
* Позиция в исходном файле – [строка: колонка].
  1. **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 : ‘))

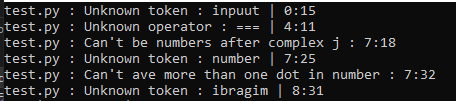


Рисунок 6 – Пример ошибки незнакомого токена

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

elif num === 0:

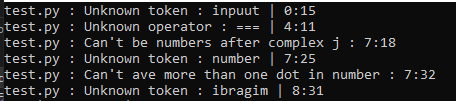


Рисунок 7 – Пример ошибки незнакомого оператора

Ошибка неправильного комплексного числа – производится, когда лекскр 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):

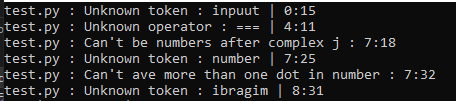


Рисунок 8 – Пример ошибки незнакомого токена

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):

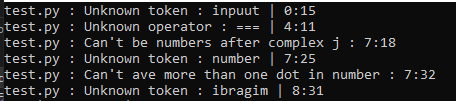


Рисунок 9 – Пример ошибки незнакомого токена

Ошибка незакрытых ковычек 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)



Рисунок 10 – Пример ошибки незнакомого токена

1. **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

## **Тестовая программа с ошибками**

num = int(inpuut("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(1j1,number + 1.1.2j):

factorial = factorial\*ibragim

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

## **Тестовая программа**

num = int(inpuut("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)