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 анализатор»

Выполнил             Я. Ю. Прескурел

Проверил                          Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc158040802)

[2 Краткие теоретические сведения 4](#_Toc158040803)

[3 Результаты выполнения лабораторной работы 5](#_Toc158040804)

[Выводы 7](#_Toc158040805)

[Список использованных источников 8](#_Toc158040806)

[Приложение А (обязательное) Листинг исходного кода 9](#_Toc158040807)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является разработка синтаксического анализатора подмножества языка программирования в Python. Также необходимо определить синтаксическое строение предложения, представленное в виде дерева зависимостей. Таким образом на основе анализа выражений выполнится группирование токенов исходной программы в грамматические фразы.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Синтаксический анализатор – это вторая фаза компиляции, следующая за лексическим анализом. Его задачей является проверка исходного текста программы на соответствие синтаксическим нормам языка программирования, а также построение структуры, отражающей синтаксическую структуру входной последовательности. Эта структура обычно представляется в виде синтаксического дерева, которое удобно использовать для дальнейшего анализа и синтеза вывода.[1]

Процесс синтаксического анализа осуществляется на основе грамматики языка программирования, которая определяет правила построения правильных программных конструкций. Результатом синтаксического анализа является структурное представление программы в виде синтаксического дерева или другой подобной структуры.

Для реализации синтаксического анализатора могут использоваться различные табличные методы, такие как LL-метод, LR-метод, метод предшествования и другие. Эти методы определяют правила, по которым происходит построение синтаксического дерева, и обеспечивают обработку правильных программных конструкций, а также выдачу сообщений об ошибках в случае несоответствия синтаксическим нормам.

Для написания синтаксического анализатора подмножества языка программирования в Python были применены следующие теоретические сведения и концепции:

1 Грамматика языка программирования: определение правил синтаксиса языка, которые используются для построения синтаксического анализатора.

2 Табличные методы синтаксического анализа: выбор и реализация конкретного табличного метода для построения синтаксического анализатора.

3 Синтаксическое дерево: создание структуры данных для представления синтаксического дерева, которое отражает иерархию конструкций в программе.

4 Обработка ошибок: введение механизмов для выдачи сообщений об ошибках при обнаружении несоответствия синтаксическим нормам языка.

5 Использование результатов лексического анализа: взаимодействие с выходными данными лексического анализатора для более эффективной обработки программного кода.

Эти концепции позволили разработать синтаксический анализатор, который группирует токены исходной программы в грамматические фразы и строит синтаксическое дерево для дальнейшего использования в синтезе вывода.

# 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе лабораторной работы был разработан синтаксический анализатор для языка программирования Python. Для работы с кодом используются файлы. При запуске программы код анализируется и разбивается на токены. Далее идет группировка токенов исходной программы в грамматические фразы, используемые для синтеза вывода. Результат работы синтаксического анализатора представлен на рисунке 3.1.

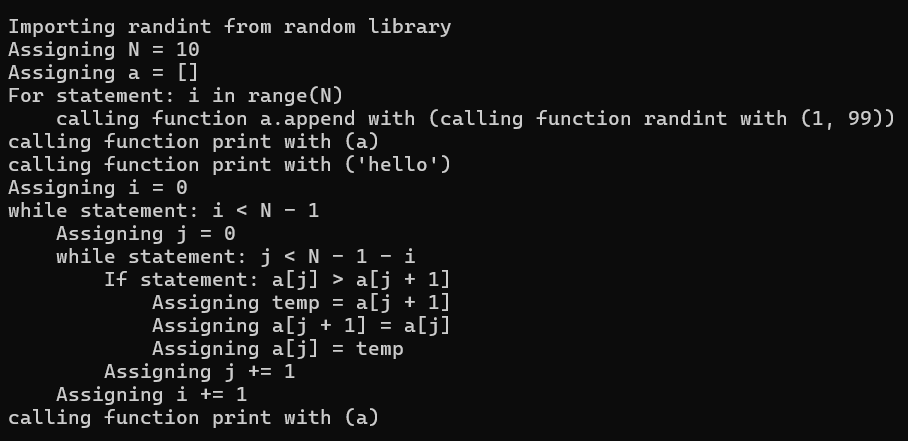


Рисунок 3.1 – Результат работы синтаксического анализатора

Кроме вывода лексем и представления грамматических фраз программа обрабатывает некоторые ошибки в коде. Если совершить попытку использования оператора сравнения вместо оператора присваивания, то выведется ошибка. Результат нахождения данной ошибки представлен на рисунке 3.2.

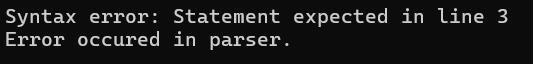


Рисунок 3.2 – Результат нахождения ошибки при неправильном использовании операторов

Если неправильно передавать значения параметров функции, также выведется ошибка с ее наименованием и нумерацией строки, в которой данная ошибка возникла. Нахождение данной ошибки представлено на рисунке 3.3.

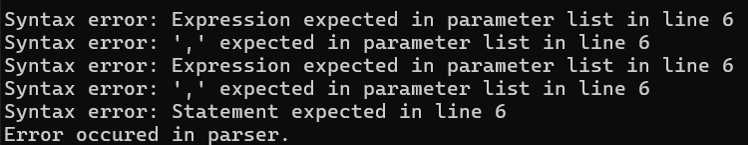


Рисунок 3.3 – Результат нахождения ошибки при неправильной передаче параметров функции

Если будет допущено использование массивов или других структур данных с пропущенной закрывающей или открывающей скобкой, то результат нахождения данной ошибки представлен на рисунке 3.4.

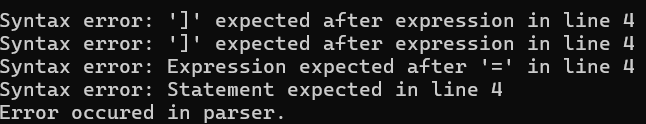


Рисунок 3.4 – Результат нахождения ошибки при пропущенной лексеме

Если допустить ошибку при написании циклов и пропустить двоеточие после условия, то выведется результат нахождения данной ошибки представленный на рисунке 3.5.

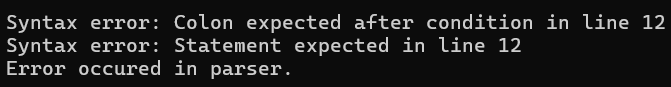


Рисунок 3.5 – Результат нахождения ошибки пропуска оператора двоеточие

Таким образом, по итогу лабораторной работы был разработан синтаксический анализатор кода, написанный на языке программирования С#, а также было реализовано нахождение разного рода синтаксических ошибок.

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы был разработан синтаксический анализатор подмножества языка программирования Python. Также были определены синтаксические правила и выполнено синтаксическое строение в виде дерева зависимостей. При определении была реализована возможность обнаружение ошибок и демонстрация сообщений о данных ошибках.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Синтаксический анализатор [Электронный ресурс]. – Режим доступа: https://nauchniestati.ru/spravka/razrabotka-sintaksicheskogo-analizatora/ – Дата доступа: 03.03.2024.
2. Введение в Python [Электронный ресурс]. – Режим доступа: https://metanit.com/python/tutorial/1.1.php. – Дата доступа: 03.03.2024.
3. Типы данных [Электронный ресурс]. – Режим доступа: https://metanit.com/python/tutorial/2.2.php. – Дата доступа: 03.03.2024.
4. Операторы в Python [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/cpp/c-operators>. – Дата доступа: 03.03.2024.
5. Функции Python [Электронный ресурс]. – Режим доступа: https://metanit.com/python/tutorial/2.8.php. – Дата доступа: 03.03.2024.
6. Классы Python [Электронный ресурс]. – Режим доступа: https://metanit.com/python/tutorial/7.1.php. – Дата доступа: 03.03.2024.

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг исходного кода

Листинг 1 – Программный код Ast.cs

using System.Collections.Generic;

namespace Lab3

{

internal enum StatementType

{

STATEMENT\_TYPE\_IMPORT,

STATEMENT\_TYPE\_ASSIGNMENT,

STATEMENT\_TYPE\_IF,

STATEMENT\_TYPE\_ELSE,

STATEMENT\_TYPE\_FOR,

STATEMENT\_TYPE\_WHILE,

STATEMENT\_TYPE\_EXPRESSION,

STATEMENT\_TYPE\_FUNCTION\_CALL,

}

internal enum AssignmentType

{

ASSIGNMENT\_TYPE\_ASSIGNMENT,

ASSIGNMENT\_TYPE\_ADD,

ASSIGNMENT\_TYPE\_SUB,

ASSIGNMENT\_TYPE\_MUL,

ASSIGNMENT\_TYPE\_DIV,

}

internal enum Expressiontype

{

EXPRESSION\_TYPE\_NONE,

EXPRESSION\_TYPE\_NAME,

EXPRESSION\_TYPE\_NUMBER,

EXPRESSION\_TYPE\_STR,

EXPRESSION\_TYPE\_RANGE,

EXPRESSION\_TYPE\_ARR,

EXPRESSION\_TYPE\_FUNC,

EXPRESSION\_TYPE\_SUB,

EXPRESSION\_TYPE\_ADD,

EXPRESSION\_TYPE\_MUL,

EXPRESSION\_TYPE\_DIV,

EXPRESSION\_TYPE\_DOT,

EXPRESSION\_TYPE\_INDEX,

EXPRESSION\_TYPE\_LESS,

EXPRESSION\_TYPE\_LESS\_OR\_EQUALS,

EXPRESSION\_TYPE\_GREATER,

EXPRESSION\_TYPE\_GREATER\_OR\_EQUALS,

EXPRESSION\_TYPE\_EQUALS,

EXPRESSION\_TYPE\_OR,

EXPRESSION\_TYPE\_AND,

EXPRESSION\_TYPE\_XOR,

EXPRESSION\_TYPE\_NOT,

}

internal class Ast

{

internal List<Statement> statements;

public override string ToString()

{

string result = "";

foreach (var stat in statements)

{

for (int i = 0; i < stat.indentation; i++)

{

result += " ";

}

result += stat.ToString() + '\n';

}

return result;

}

}

internal class Statement

{

internal int line;

internal int row;

internal int indentation;

internal StatementType statementType;

public Statement(int line, int row, StatementType statementType)

{

this.line = line;

this.row = row;

this.statementType = statementType;

}

public override string ToString()

{

return $"{statementType.ToString()}({line}:{row})";

}

}

internal class Import : Statement

{

internal string library;

internal string name;

public Import(int line, int row) : base(line, row, StatementType.STATEMENT\_TYPE\_IMPORT) { }

public override string ToString()

{

return $"Importing {name} from {library} library";

}

}

internal class Assignment : Statement

{

internal Expression left;

internal Expression right;

internal AssignmentType type;

public Assignment(int line, int row) : base(line, row, StatementType.STATEMENT\_TYPE\_ASSIGNMENT) { }

public override string ToString()

{

switch (type)

{

case AssignmentType.ASSIGNMENT\_TYPE\_ASSIGNMENT:

return $"Assigning {left} = {right}";

case AssignmentType.ASSIGNMENT\_TYPE\_ADD:

return $"Assigning {left} += {right}";

case AssignmentType.ASSIGNMENT\_TYPE\_SUB:

return $"Assigning {left} -= {right}";

case AssignmentType.ASSIGNMENT\_TYPE\_MUL:

return $"Assigning {left} \*= {right}";

case AssignmentType.ASSIGNMENT\_TYPE\_DIV:

return $"Assigning {left} /= {right}";

default:

return "???error???";

}

}

}

internal class If : Statement

{

internal Expression condition;

internal bool isElif = false;

public If(int line, int row) : base(line, row, StatementType.STATEMENT\_TYPE\_IF) { }

public override string ToString()

{

return $"If statement: {condition}";

}

}

internal class Else : Statement

{

public Else(int line, int row) : base(line, row, StatementType.STATEMENT\_TYPE\_ELSE) { }

public override string ToString()

{

return $"Else branch:";

}

}

internal class For : Statement

{

internal Expression variable;

internal Expression range;

public For(int line, int row) : base(line, row, StatementType.STATEMENT\_TYPE\_FOR) { }

public override string ToString()

{

return $"For statement: {variable} in {range}";

}

}

internal class While : Statement

{

internal Expression condition;

public While(int line, int row) : base(line, row, StatementType.STATEMENT\_TYPE\_WHILE) { }

public override string ToString()

{

return $"while statement: {condition}";

}

}

internal class Expression : Statement

{

internal Expressiontype expressionType;

internal Expression left;

internal Expression right;

internal string value;

public Expression(int line, int row, StatementType statementType = StatementType.STATEMENT\_TYPE\_EXPRESSION) : base(line, row, statementType) { }

public override string ToString()

{

if (expressionType == Expressiontype.EXPRESSION\_TYPE\_SUB && right == null)

{

return $"-{left}";

}

switch (expressionType)

{

case Expressiontype.EXPRESSION\_TYPE\_NAME:

return $"{value}";

case Expressiontype.EXPRESSION\_TYPE\_NUMBER:

return $"{value}";

case Expressiontype.EXPRESSION\_TYPE\_STR:

return $"\'{value}\'";

case Expressiontype.EXPRESSION\_TYPE\_OR:

return $"{left} | {right}";

case Expressiontype.EXPRESSION\_TYPE\_AND:

return $"{left} & {right}";

case Expressiontype.EXPRESSION\_TYPE\_XOR:

return $"{left} ^ {right}";

default:

return "???error???";

}

}

}

internal class FunctionCall : Expression

{

internal List<Expression> parameters;

public FunctionCall(int line, int row) : base(line, row, StatementType.STATEMENT\_TYPE\_FUNCTION\_CALL) { }

public override string ToString()

{

string args = "";

for (int i = 0; i < parameters.Count; i++)

{

if (i > 0)

args += ", ";

var a = parameters[i];

args += a.ToString();

}

return $"calling function {left} with ({args})";

}

}

}