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

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.

```
Importing randint from random library
Assigning N = 10
Assigning a = []
For statement: i in range(N)
    calling function a.append with (calling function randint with (1, 99))
calling function print with (a)
calling function print with ('hello')
Assigning i = 0
while statement: i < N - 1
    Assigning j = 0
    while statement: j < N - 1 - i
        If statement: a[j] > a[j + 1]
            Assigning temp = a[j + 1]
            Assigning a[j + 1] = a[j]
            Assigning a[j] = temp
        Assigning j += 1
    Assigning i += 1
calling function print with (a)
```

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.

```
Syntax error: Statement expected in line 3
Error occured in parser.
```

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.

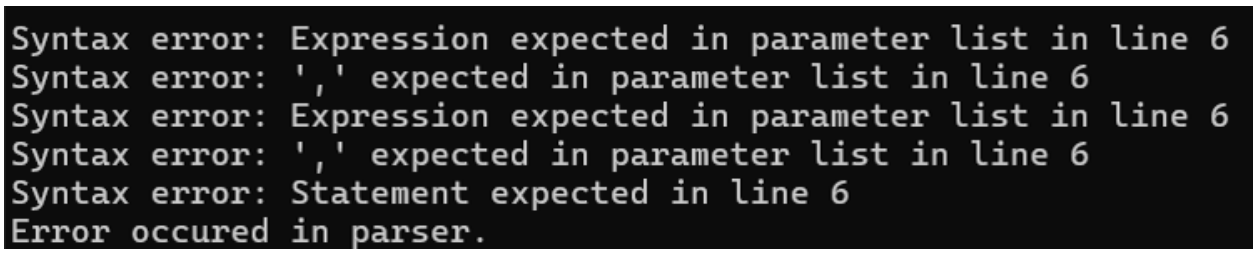

Рисунок 3.3 – Результат нахождения ошибки при неправильной передаче параметров функции

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.

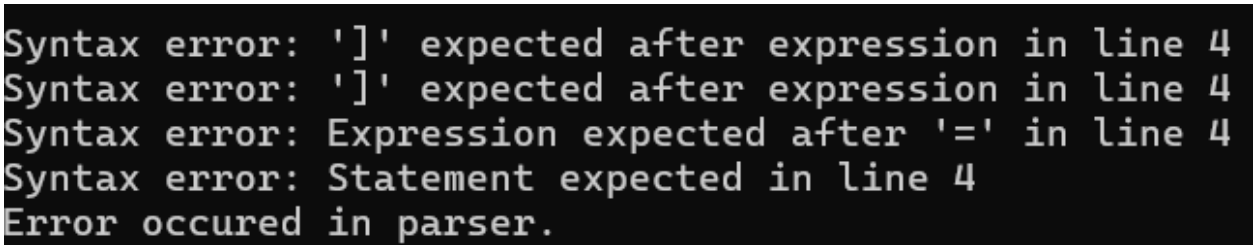

Рисунок 3.4 – Результат нахождения ошибки при пропущенной лексеме

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

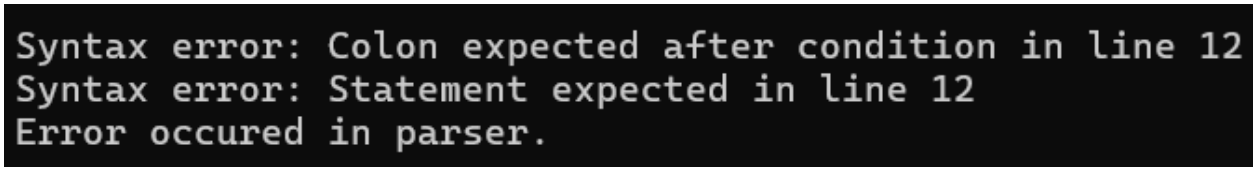

Рисунок 3.5 – Результат нахождения ошибки пропуска оператора двоеточие

Таким образом, по итогу лабораторной работы был разработан синтаксический анализатор кода, написанный на языке программирования C#, 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})";
            }
      }
}
```