

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

Фигура

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**Департамент программной инженерии и искусственного интеллекта**

РАЗРАБОТКА ТРАНСЛЯТОРА, ПЕРЕВОДЯЩЕГО ПОДМОЖЕСТВО ЯЗЫКА TURBO PASCAL В ЭКВИВАЛЕНТНОЕ ПОДМНОЖЕСТВО ЯЗЫКА PYTHON

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Теория языков программирования и компиляторы»

по образовательной программе подготовки бакалавров по направлению

09.03.04 – Программная инженерия

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |  |  | Студенты гр. Б9121-09.03.04 | | |
|  |  |  | | Д.О. Верховцов |
|  | | | | | | |  |  | (подпись) | |  |
|  | | | | | | |  |  |  | | Ю.В. Коринецкая |
|  | | | | | | |  |  | (подпись) | |  |
|  | | | | | | |  |  |  | | М.А. Омельченко |
|  | | | | | | |  |  | (подпись) | |  |
|  | | | | | | | | | | | |
| Защищен с оценкой | | | | | | |  |  | Руководитель | | |
|  | | | | | | |  |  | ассистент ДПИиИИ | | |
|  | | | |  |  | |  |  | ученая степень, должность |  | В.В. Самойлов |
| (подпись) | | | |  | (И.О. Фамилия) | |  |  | (подпись) |  | (И.О. Фамилия) |
| « |  | » |  | | | 2024г |  |  |  | | |
|  | | | | | | | | | | | |

г. Владивосток

2024 г.

**Оглавление**

[Введение 2](#_Toc1322947428)

[1 Неформальная постановка задачи 2](#_Toc1751066931)

[2 Синтаксис входного языка 4](#_Toc915362381)

[3 Контекстные условия языка Python 10](#_Toc1279632542)

[Таблица соответствия 11](#_Toc1117704123)

[5 Проект лексического анализатора 16](#_Toc961249952)

[5.1 Таблица ключевых слов Pascal 16](#_Toc375539931)

[5.2 Типы лексем 18](#_Toc2050952523)

[5.3 Ошибки лексического анализатора 20](#_Toc153633567)

[5.4 Конечный автомат лексического анализатора 20](#_Toc588227547)

[6 Проект синтаксического анализатора 21](#_Toc2111299355)

[Модель данных 22](#_Toc1120807922)

[Синтаксический анализатор 29](#_Toc1127066423)

[Пример дерева разбора для следующей входной цепочки 29](#_Toc573405482)

[Ошибки синтаксического анализатора 32](#_Toc1023671706)

[Заключение 33](#_Toc1524303002)

[Список литературы 34](#_Toc596846946)

[Исполнители работы 35](#_Toc636654305)

# **Введение**

Транслятор — программа, выполняющая трансляцию преобразование программы, представленной на одном из языков программирования, в программу, написанную на другом языке.

Транслятор обычно выполняет также диагностику ошибок, формирует словари идентификаторов, выдаёт для печати текст программы и т. д.

Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель под её управлением.

Язык, на котором представлена исходная программа, называется входным языком – Turbo Pascal. Выходным языком является Python.

**Целью курсового проекта** является разработка проекта транслятора, который преобразует программу, содержащую подмножество языка Turbo Pascal, в программу, содержащую подмножество языка Python генерируя эквивалентный исходный код.

**Задачи курсового проекта:**

1. Описать грамматику подмножества входного языка.
2. Описать контекстные условия входного и выходного языков.
3. Описать соответствие конструкций входного и выходного языков.
4. Разработать проект лексического анализатора.
5. Разработать проект синтаксического анализатора.

# **1 Неформальная постановка задачи**

Неформальной постановкой задачи курсового проекта является разработка проекта транслятора из подмножества языка **Turbo Pascal** в эквивалентное подмножество языка **Python**.

Подмножество языка Turbo Pascal включает в себя:

1. Типы данных:

1.1. Простые типы:

1.1.1. Целочисленные типы: ShortInt, Integer, LongInt, Byte, Word.

1.1.2. Булевы типы: Boolean, WordBool, LongBool и ByteBool.

1.1.3. Символьный тип: Char.

1.1.4. Вещественные типы: Real, Single, Double, Extended, Comp.

1.2. Строковые типы: String.

1.3. Структурные типы:

1.3.1. Типизированные массивы: array.

1.3.2. Файловые типы: file.

1.3.3. Объектные типы: object.

1.3.4. Типы записей: record.

1.3.5. Типы наборов: set.

1. Операторы:

2.1. Арифметические: +, -, \*, /, mod, div

2.2. Сравнения: =, >, >=, <, <=, <>

2.3. Логические: xor, or, and, not

2.4. Присваивания: :=

1. Операторы циклов:

3.1. Арифметические циклы: for <переменная> := <значение\_1> to <значение\_2> do <оператор>

3.2. Циклы с предусловием: while <выражение> do <оператор>;

3.3. Циклы с постусловием: repeat <оператор>; ... <оператор>; until <выражение>;

3.4. Операторы завершения циклов: Break, Continue.

1. Операторы ветвления:

4.1. Оператор условного перехода: if <условие> then <оператор 1> else <оператор 2>;

4.2. Оператор выбора:

case <выражение> of  
 <вариант> : <оператор>;  
 ...  
 <вариант> : <оператор>;  
 else <оператор>  
end;

4.3. Оператор безусловного перехода: goto <метка>;

1. Стандартные функции:

5.1. Вывод:

5.1.1. print

5.1.2. println

5.2. Чтение:

5.2.1. read

5.2.2. readln

1. Переменные:

6.1. Описание переменной: Var <идентификатор>, ... <идентификатор> : <тип>;

6.2. Объявление переменной: <идентификатор> := <значение>

1. Ключевые слова:

If, and, repeat, array, in, set, begin, case, const, string, mod, then, nil, to, div, not, type, do, object, unit, downto, of, until, else, end, or, var, file, while, for, procedure, function, xor, goto, record.

# **2 Синтаксис входного языка**

На вход в компилятор поступает цепочка, принадлежащая языку Turbo Pascal. Для корректной обработки входных данных необходимо разобрать синтаксис входного языка Turbo Pascal.

G = (N, ∑, P, <программа>);

N = {<программа>, <разделы программы>,<заголовок программы>, <подключение библиотек>, <библиотеки>, <имя библиотеки>, <объявление меток>, <метки>, <объявление констант>,<константы>, <объявление типов>,<типы> , <определение типа>, <перечисляемое определение>, <допустимые значение типа>, <значение типа>, <интервальное определение>, <объявление переменных>,<переменные>, <имена переменных>, <значение>, <основной блок программы>, <тело подпрограммы>, <объявление процедуры> , <объявление функции>, <вывод в консоль>, <содержимое вывода>, <параметры процедуры>, <тип данных>, <множество>, <тип данных функции>, <символ>, <цифра>, <целое число>, <вещественное число>, <число>, <строка>, <идентификатор>, <символ идентификатора>, <ид>, <тело функции>, <возврат значения>, <блок кода>, <цикл>, <ветвление>, <арифметический оператор>, <унарный арифметический оператор> , <символьное значение> , <строковое значение>, <оператор сравнения>, <унарная арифметическая операция>, <арифметический операнд>, <арифметическое выражение> , <операнд сравнения>, <выражение сравнения>, <логический бинарный оператор>, <логическое значение>, <логический операнд>, <логическое выражение>, <побитовый бинарный оператор>, <побитовый операнд>, <побитовое выражение>, <инструкция>, <выражение>, <вызов функции>, <параметры вызова функции>, <оператор присваивания>, <присваивание>, <имя функции>, <имя переменной>, <имя константы>, <имя программы>, <тип данных переменной>, <начало программы>, <конец программы>, <прочие символы>, <любая строка>, <тело цикла>, <оператор цикла>, <знак числа>, <имя или инициализация>, <следующая константа>}

∑ = {a, b, c, …, x, y, z, A, B, C, … X, Y, Z, а … я, А … Я, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (, ), {, }, [, ], +, -, \*, /, %, >, <, =, !, &, |, ; , “ , ‘, , , \_, # , @, $, ^, ~, №, :, ?};

P: { <программа> → <разделы программы>

<разделы программы> → <заголовок программы> | <заголовок программы><разделы программы> | <подключение библиотек>| <подключение библиотек> <разделы программы>| <объявление меток>| <объявление меток> <разделы программы>| <объявление констант>| <объявление констант> <разделы программы>| <объявление типов>| <объявление типов> <разделы программы>| <объявление переменных>| <объявление переменных> <разделы программы>|<объявление процедуры> <разделы программы> | <объявление процедуры> | <объявление функции> <разделы программы> | <объявление функции>|<основной блок программы>

<заголовок программы> → **programm**<имя программы>

<имя программы > → <идентификатор>;

<подключение библиотек> → **uses**<библиотеки>

<библиотеки> → <имя библиотеки>, <библиотеки> | <имя библиотеки>;

<объявление меток> →**label**<метки>

<метки> → <имя метки>, <метки> | <имя метки>;

<объявление констант> →**const<**константы**>**

**<**константы**>** →<имя константы> = <значение>, **<**константы**>|** <имя константы> = <значение>;

<значение> → <число> | <строковое значение> | <символьное значение> | <логическое значение>|<множество>

<объявление типов> →**type** <типы>

<типы> →<имя типа> = <определение типа>, <типы>| <имя типа> = <определение типа>;

<определение типа> →<перечисляемое определение>|<интервальное определение>

<перечисляемое определение> →(<допустимые значения типа>);

<допустимые значения типа> →<значение типа>, <допустимые значения типа>| <значение типа>

<значение типа> →<строка>

<интервальное определение> →<целое число>.. <целое число>| <вещественное число>..< вещественное число>| <символьное значение>.. <символьное значение> | < логическое значение >..< логическое значение >

<объявление переменных> →**var** <переменные>

<переменные> → <имена переменных>:<тип данных>; <переменные>| <имена переменных>:<тип данных>;

<имена переменных > →<имя переменной>,<имена переменных > |<имя переменной>

<объявление процедуры>→ **procedure** <имя процедуры> (<параметры процедуры>);<описание подпрограммы>

<описание подпрограммы> → <разделы подпрограммы>

<параметры процедуры> → <имена переменных>:<тип данных> < параметры процедуры >| <имена переменных>:<тип данных>

<объявление функции>→ **function** <имя функции > (<параметры процедуры>):<тип данных>;<описание подпрограммы>

<тип данных> → boolean | byte | string| integer | double | char

<тип данных переменной> → <тип данных>

<символ> → a |…| z | A |… | Z

<цифра>→ 0|1|2|3|4|5|6|7|8|9

< число> → <цифра> | <число>

<строка> → <символ><строка>|<цифра><строка>| <символ> | <цифра> | <строка>

<прочие символы> → а| … | я | А | … | Я | \_ | ! | @ | # | $ | % | ^ | & | \* | ( | ) | - | + | [ | ] | { | } | ~ | № | ; | : | ? | = | < | >| . | ,

<символ идентификатора> → <символ> <идентификатор> → <символ идентификатора> <ид> | <символ идентификатора>

<ид> → <символ идентификатора> <ид> | <цифра> <ид> | <символ идентификатора> | <цифра>

<основной блок программы> →**begin**<блок кода> <возврат значения>**end** | **begin**<блок кода>**end**

<возврат значения> → <имя функции>:=<выражение>;| <имя функции>:= <имя переменной>; | <имя функции>:= <имя константы>;

<блок кода> → <инструкция> <блок кода> | <инструкция>;

<цикл> → while (<логическое выражение>) <тело цикла>| repeat <тело цикла> until(<логическое выражение>); | for (<инструкция>; <логическое выражение>; <инструкция>) <тело цикла>

<тело цикла> → **begin**<блок кода> <оператор цикла> <блок кода> **end**| **begin** <оператор цикла> <блок кода> **end** | **begin** <блок кода> <оператор цикла> **end** | **begin** <блок кода> **end** | **begin** <оператор цикла> **end|** <оператор цикла>| <инструкция>

<оператор цикла> → **break** | **continue**

<ветвление> → **if** <логическое выражение>**then begin**<блок кода>**end** | if <логическое выражение> **then begin**<блок кода>**end else** {<блок кода>} | if <логическое выражение> **begin** <блок кода> **end** **else** <ветвление>

<арифметический оператор> → + | - | / | \* | %

<знак числа> → + | -

<символьное значение> → ’<символ>’ | ’<цифра>’ | ‘<арифметический оператор>’ | ‘<прочие символы>’

<строковое значение> → ‘<любая строка>'

<оператор сравнения> → == | **<>** | > | < | >= | <=

<арифметический операнд> → <число> | <имя переменной> | <вызов функции> | (<арифметическое выражение>) | <унарная арифметическая операция>

<арифметическое выражение> → <арифметический операнд> <арифметический оператор> <арифметический операнд>

<операнд сравнения> → <число> | <символьное значение> | <имя переменной> | <вызов функции>

<выражение сравнения> → <операнд сравнения> <оператор сравнения> <операнд сравнения>

<логический бинарный оператор> → **and** | **or**

<логическое значение> → **true** | **false**

<логический операнд> → <логическое значение> | <выражение сравнения> | (<логическое выражение>) | <имя переменной> | <вызов функции>

<логическое выражение> → **<>**<логический операнд> | <логический операнд> <логический бинарный оператор> <логический операнд> | <логический операнд>

<инструкция> → <присваивание>; | <выражение>; | <цикл> | <ветвление> | <вывод в консоль>

<вывод в консоль> → Writeln( <содержимое вывода> ) ;

<содержимое вывода> → <идентификатор>, <содержимое вывода> | <идентификатор>

<выражение> → <арифметическое выражение> | <логическое выражение> | <значение> | <побитовое выражение>

<вызов функции> → <имя функции> (<параметры вызова функции>)

<параметры вызова функции> → <выражение>| <выражение>, <параметры вызова функции>

<оператор присваивания> → :=

<присваивание> → <имя переменной> <оператор присваивания> <выражение>

<имя функции> → <идентификатор>

<имя переменной> → <идентификатор>

<имя константы> → <идентификатор>

**3 Контекстные условия языка Pascal**

**Объявление и использование идентификаторов:**

* Все переменные, константы, типы, процедуры и функции должны быть объявлены перед использованием.
* Идентификаторы должны начинаться с буквы и могут содержать буквы, цифры и символ подчеркивания.

**Типы данных:**

* Pascal является строго типизированным языком, поэтому тип переменной должен быть явно указан при её объявлении.
* Совместимость типов проверяется компилятором. Присваивание значений несовместимых типов не допускается без явного преобразования.

**Область видимости:**

* Переменные, объявленные в теле процедуры или функции, видны только в пределах этой процедуры или функции.
* Глобальные переменные объявляются в разделе var и доступны во всей программе после их объявления.

**Структуры управления:**

* Все управляющие структуры (такие как if, while, for, repeat) должны быть правильно вложены.
* Использование begin и end обязательно для группировки операторов в составные операторы.

**Процедуры и функции:**

* Процедуры и функции должны быть объявлены с указанием типов параметров и возвращаемых значений.
* Вызов функции должен быть частью выражения, а вызов процедуры — отдельным оператором.

**Модули и блоки:**

* Программа на Pascal состоит из главного блока program и, возможно, нескольких модулей, процедур и функций.
* Каждая программа должна начинаться с заголовка программы и может содержать разделы uses, const, type, var и begin...end.

# **Таблица соответствия**

При помощи таблицы соответствия отображаются элементы языка Pascal, эквивалентные элементам языка Python. В ней отображаются как представление отдельных лексем, так и структура программы, процедур и функций.

Таблица 1 – Таблица соответствия языков Pascal и Python

|  |  |
| --- | --- |
| **Pascal** | **Python** |
| 1. Структура программы | |
| program <заголовок программы>  <блок подключений>  <блок постоянных значений>  <блок описания переменных>  <объявление функций>  <главная функция> | <блок подключений>  <блок описания переменных>  <объявление функций>  <главная функция> |
| 1. Блок описания переменных | |
| <имя переменной 1>, <имя переменной 2>, …, <имя переменной n>: <тип данных>; | <имя переменной 1>: <тип данных>  <имя переменной 2>: <тип данных>  …  <имя переменной n>: <тип данных> |
| <имя переменной>: <тип данных>:= <значение переменной>; | <имя переменной>: <тип данных> = <значение переменной> |
| 1. Циклы | |
| for <имя переменной> := <нижняя граница> to <верхняя граница> do <блок операторов> end; | for <имя переменной> in range(<нижняя граница >; < верхняя граница >):  <блок операторов> |
| for <имя переменной> := <верхняя граница> downto <нижняя граница> do begin <блок операторов> end; | for <имя переменной> in range(<верхняя граница>, <нижняя граница>, -1):  <блок операторов> |
| while <условие> do begin <оператор> end; | while <условие\_1>:  <блок операторов> |
| repeat <операторы> until <условие\_2>; | Отсутствует |
| 1. Условные переходы | |
| if <условие> then begin <блок операторов> end; | if <условие>:  <блок операторов> |
| if <условие> then begin <блок операторов> else <блок операторов> end; | if <условие>:  <блок операторов>  else:  <блок операторов> |

Продолжение таблицы 1

|  |  |
| --- | --- |
| **Pascal** | **Python** |
| 1. Условные переходы | |
| if <условие> then begin <блок операторов> else if <условие> then <блок операторов> end; end; | if <условие>:  <блок операторов>  elif <условие>:  <блок операторов> |
| if <условие> then begin <блок операторов> else <ветвление > | if <условие>  <блок операторов>  else  <ветвление> |
| 1. Типы данных | |
| 5.1 Целочисленные типы | |
| ShortInt  Integer  LongInt  Byte  Word | class 'int'  class 'int'  class 'int'  class 'bytes'  class 'int' |
| 5.2 Булевы типы | |
| Boolean  WordBool  LongBool  ByteBool. | class 'bool'  list of 'bool'  list of 'bool'  list of 'bool' |
| 5.3 Символьные типы данных | |
| Сhar  String | class 'str'  class 'str' |

Продолжение таблицы 1

|  |  |
| --- | --- |
| **Pascal** | **Python** |
| 5.4 Вещественные типы данных | |
| Real  Single  Double  Extended  Comp | class 'float'  class 'float'  class 'float'  class 'float'  class 'complex' |
| 5.5 Структурные типы данных | |
| array  file  object  record  set | list  class '\_io.TextIOWrapper'  class of …  class of …  class 'set' |
| 1. Логические значения | |
| true | True |
| false | False |
| 1. Логические операции | |
| not | not |
| or | or |
| 1. Логические операции | |
| and | and |
| xor | ^ |
| 1. Модификаторы типов данных | |
| const | отсутствуют\* |
| 1. Математические операции | |
| + | + |
| - | - |
| / | / |
| \* | \* |
| - (унарный) | - (унарный) |
| 1. Математические функции | |
| Abs() | abs() |

Продолжение таблицы 1

|  |  |
| --- | --- |
| **Pascal** | **Python** |
| Sin()  Cos()  Arctan() | math.sin()  math.cos()  math.atan() |
| Sqrt()  Sqr()  Exp()  Ln() | math.sqrt()  \*\*2  math.exp()  math.log() |
| Frac()  Int() | math.modf()[0]  math.modf()[1] |
| Random()  Succ()  Pred() | random.randint(A, B)  отсутствует  отсутствует |
| 1. Операторы присваивания | |
| :=  +=  -=  \*=  /= | =  +=  -=  \*=  /= |
| 1. Точка входа в программу | |
| program  begin  end. | if \_\_name\_\_ == ‘\_\_main\_\_’:  pass |
| 1. Функции и процедуры | |
| procedure <имя > (<список формальных параметров>);  <раздел описаний>  begin  <блок операторов>  end; | def <имя > (<список формальных параметров>) -> None:  <раздел описаний>  <блок операторов> |

Продолжение таблицы 1

|  |  |
| --- | --- |
| **Pascal** | **Python** |
| 1. Функции и процедуры | |
| procedure <имя > (<список формальных параметров>);  <раздел описаний>  begin  <блок операторов>  end; | def <имя > (<список формальных параметров>) -> None:  <раздел описаний>  <блок операторов> |
| function <имя > (<список формальных параметров>): <тип возвращаемого значения>;  <раздел описаний>  begin  <блок операторов>  <имя>:= <значение>;  end; | def <имя > (<список формальных параметров>) -> <тип возвращаемого значения>:  <раздел описаний>  <блок операторов>  return <значение> |
| 1. Функции ввода-вывода | |
| readln(“Введите ваше число”) | input(‘Введите ваше число”) |
| writeln(“Ваше число:”, a) | print(“Ваше число:”, a) |

\* существует договорённость использовать именованные константы

# **5 Проект лексического анализатора**

# **5.1 Таблица ключевых слов Pascal**

Ключевые слова – это слова языка программирования, которые имеют специальное, раз и навсегда закрепленное за ними значение. В программе нельзя использовать идентификаторы, совпадающие по написанию с ключевыми словами.

Таблица 2 – Ключевые слова языка Turbo Pascal

|  |  |
| --- | --- |
| **Ключевые слова** | **Ключ** |
| And | K1(логический оператор) |
| Array | K2(объявление массива) |

Продолжение таблицы 2

|  |  |
| --- | --- |
| **Ключевые слова** | **Ключ** |
| Begin | K3(начало блока кода) |
| Case | K4(оператор выбора) |
| Const | K5(объявление константы) |
| Div | K6(оператор целочисленного деления) |
| Do | K7(ключевое слово для начала цикла или выполнения серии инструкций) |
| Downto | K8(указывает на уменьшение значения переменной в цикле for) |
| Else | K9(часть условного оператора, выполняемая, если условие не истинно) |
| end | K10(конец блока кода) |
| File | K11(тип данных, представляющий файл на диске) |
| For | K12(оператор цикла) |
| Function | K13(определение функции) |
| Goto | K14(оператор перехода к метке) |
| If | К15(условный оператор) |
| Label | К16(метка) |
| Mod | К17(оператор остатка на деление) |
| Nil | К18(пустая ссылка) |
| Not | К19(оператор логического отрицания) |
| Of | К20(Используется при работе с типами данных или записями) |
| Or | К21(логический оператор) |
| Record | К22(определение структуры данных) |
| Repeat | К23(цикл, который выполняется до тех пор, пока условие не станет истинным ) |
| Set | К24(тип данных, представляющий множество значений) |
| Then | К25(часть условного оператора, выполняемая, если условие истинно) |

Продолжение таблицы 2

|  |  |
| --- | --- |
| **Ключевые слова** | **Ключ** |
| To | К26(используется в цикле for для указания диапазона значений переменной) |
| Type | К27(определение нового типа данных) |
| Until | К28(условие цикла repeat-until) |
| Var | К29(объявление переменных) |
| While | К30(цикл, который выполняется до тех пор, пока условие истинно) |
| With | К31(используется для упрощения доступа к полям записи или объекта) |

Зарезервированные слова – это слова языка программирования, которые имеют специальное значение. Отличительным свойством от ключевых слов является возможность переопределить зарезервированные имена. Так в Pascal можно переопределить слова Integer, Writeln и т.д.

Таблица 3 *–* Зарезервированные слова языкаTurbo Pascal

|  |  |
| --- | --- |
| **Зарезервированные имена** | **Значения** |
| Byte | R1(тип данных) |
| Short | R2(тип данных) |
| Word | R3(тип данных) |
| Integer | R4(тип данных) |
| Logint | R5(тип данных) |
| String | R6(тип данных) | |
| Boolean | R7(тип данных) | |
| Writeln | R8(встроенный метод) | |
| Readln | R9(встроенный метод) | |
| Read | R10(встроенный метод) | |
| Write | R11(встроенный метод) | |
| Text | R12(тип данных) | |

# **5.2 Типы лексем**

Лексемы — это неделимые последовательности символов алфавита (элементарные конструкции), имеющие самостоятельный смысл. Они образуются из основных символов языка, так же как слова обычного языка строятся из букв. Возможны лексемы, состоящие из одного символа, например, знаки операций.

Таблица 4 – Таблица лексем

|  |  |
| --- | --- |
| **Идентификатор** | **1. Ключевое слово**  **2. Зарезервированное имя** |
| Унарные оператор | 1. @ (O1), not (O2) |
| Операторы умножения | 2. \* (O3), / (O4), div (O5), mod (O6), and (O7), shl (O8), shr (O9) |
| Оператор сложения | 3. +(O10), -(O11), or(O12), xor(O13) |
| Операции сравнения | 4. = (O14), <> (O15), > (O16), < (O17), >= (O18), <= (O19) |
| Операторы присваивания | 5. += (O19), -=(O20), \*=(O21), /=(O22), :=(O23) |
| Разделители | ( (D1)  ) (D2)  .(D3)  , (D4)  : (D5)  .. (D6) | |

# **5.3 Ошибки лексического анализатора**

1) Некорректно задана лексема численного типа (после последней цифры числа обнаружена лишняя точка, либо строковый разделитель, либо буква).

2) Некорректно задана лексема строкового типа (после строкового разделителя встречена цифра, буква или ещё один строковый разделитель).

3) Некорректно задана лексема символьного типа (после строкового разделителя встречена цифра, буква или ещё один строковый разделитель).

4) Не встречена закрывающая кавычка во время чтения лексемы строкового типа (символов между разделителями больше одного, либо отсутствует закрывающий строковый разделитель (двойная кавычка)).

5) Не встречена закрывающая кавычка во время чтения лексемы строкового типа (не обнаружен закрывающий строковый разделитель (одинарная кавычка)).

6) Некорректно заданная лексема (не нашлось такой точки входа в лексическом анализаторе, которая распознаёт текущий символ входного текста).

# **Конечный автомат лексического анализатора**

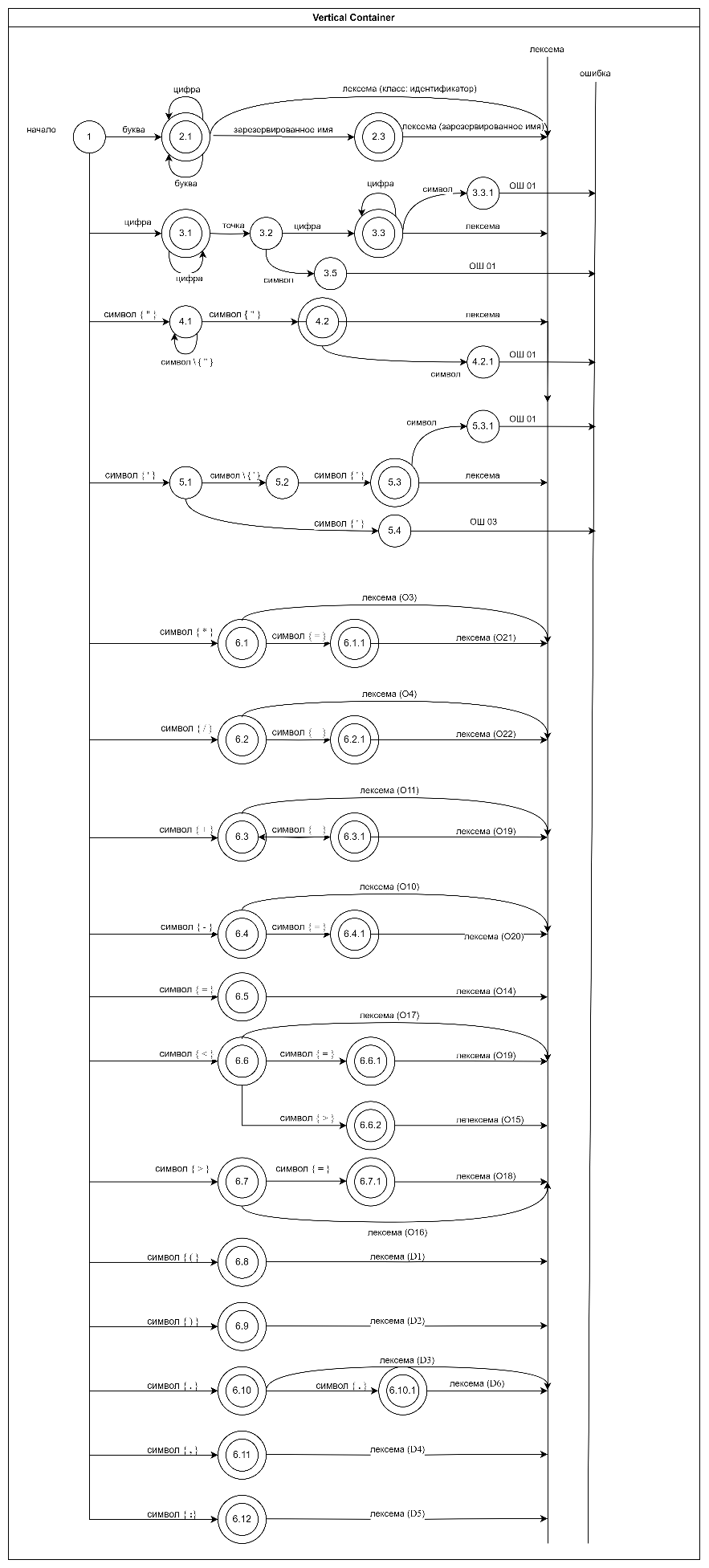


Рисунок 1 – Конечный автомат лексического анализатора

# **6 Проект синтаксического анализатора**

Синтаксический анализатор — это часть компилятора, которая отвечает за выявление и проверку синтаксических конструкций входного языка. Синтаксический анализатор получает строку токенов от лексического анализатора, и проверяет, может ли эта строка токенов порождаться грамматикой входного языка. Ещё одной функцией синтаксического анализатора является генерация сообщений обо всех выявленных ошибках, причём достаточно внятных и полных, а кроме того, синтаксический анализатор должен уметь обрабатывать обычные, часто встречающиеся ошибки и продолжать работу с оставшейся частью программы.

В качестве алгоритма синтаксического анализа был выбран нисходящий разбор с возвратами. Это вытекает из простоты в понимании алгоритма для его реализации.

# **Модель данных**

Язык реализации – Python. Алгоритм разбора реализован в классе Parser. Для работы необходимо проинициализировать объект этого класса: задать ему формальную грамматику для разбора класса Grammar и лексический анализатор класса Lexical\_Analyzer. Метод parse класса Parser получает в качестве параметра входную цепочку, содержащую текст исходной программы, написанный на Turbo pascal, и возвращает массив, содержащий правый разбор входной цепочки в случае успеха, либо код ошибки синтаксического анализатора в случае неудачи. Формальная грамматика будет представлена в виде класса Grammar. Алгоритм лексического анализа спроектирован в классе LexicalAnalyzer, перед лексическим анализом тест должен быть предобработан классом Preprocessing, так же для корректной работы лексического анализа спроектирован класс лексемы Lexem и класс ошибок лексического анализа LexemException(Exception). После лексического анализа список лексем передается в алгоритм синтаксического анализа, спроектировано в классе Parser. Для работы синтаксического анализа так же спроектированы: класс дерева разбора Node, класс с методами разбора по правилам грамматики Rules и класс с ошибками синтаксического анализатора.

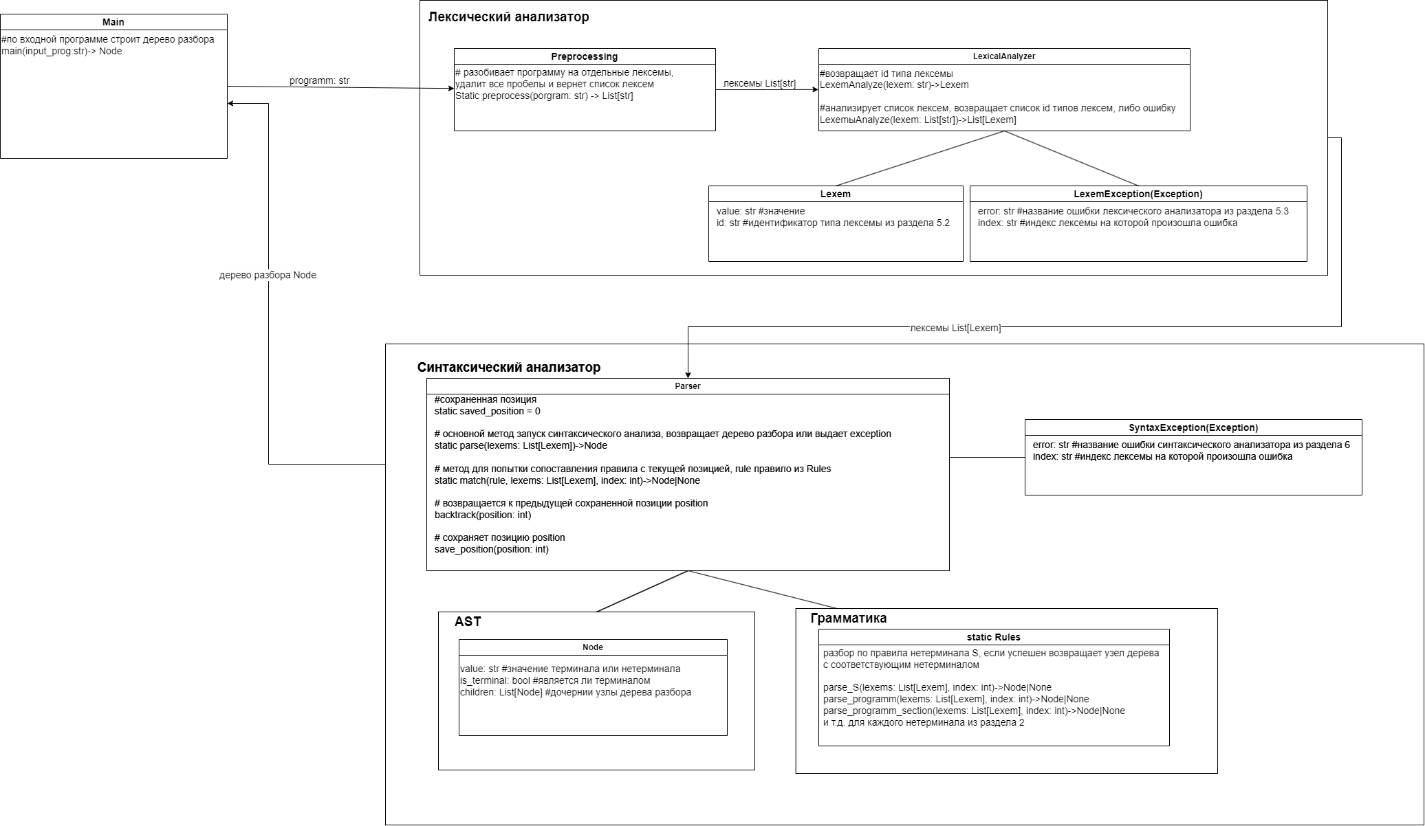
****

Рисунок 2 - диаграмма модели данных

**Алгоритм нисходящего разбора с возвратами**

1. Строится грамматика, которая задает синтаксис языка, с помощью контекстно-свободной грамматики. Грамматика состоит из набора правил, каждое из которых определяет, каким образом символы могут сочетаться друг с другом.
2. Создается стек разбора, который будет содержать терминалы (лексемы) и нетерминалы (символы грамматики). Начальным символом является стартовый символ грамматики.
3. Рекурсивная функция разбора начинает анализ с начала строки. В каждом шаге функция получает текущий символ из строки и проверяет его соответствие с символами на вершине стека разбора.
4. Если символ на вершине стека является терминалом (лексемой), то он должен совпадать с текущим символом из строки. Если совпадение происходит, то символ снимается со стека и анализ переходит к следующему символу в строке.
5. Если символ на вершине стека является нетерминалом, то происходит выбор соответствующего правила грамматики для этого нетерминала. Для каждого правила создается новая ветвь рекурсии, в которой в стек разбора добавляются символы этого правила в обратном порядке. Затем рекурсивно применяется функция разбора для каждой ветви.
6. Если в процессе разбора обнаруживается несоответствие между символами строки и стеком разбора, то происходит откат (возврат) к предыдущему состоянию, где была альтернатива выбора правила грамматики или ветвь рекурсии. Это позволяет исследовать другие возможности разбора.
7. Вся процедура продолжается до тех пор, пока не будет достигнуто правило, соответствующее всей строке, или пока все ветви рекурсии не будут исследованы.
8. Если успешно найдено правило грамматики, соответствующее строке, то может быть построено дерево разбора, которое отражает структуру строки в соответствии с правилами грамматики.

# **Пример дерева разбора для следующей входной цепочки**

Пусть входная цепочка имеет вид:

**program** prog**;**

**var**

n1: Integer;

**begin**

n1:=1;

WriteLn(n1);

**end.**

**Пример дерева разбора данной цепочки представлен на рисунке 3:**

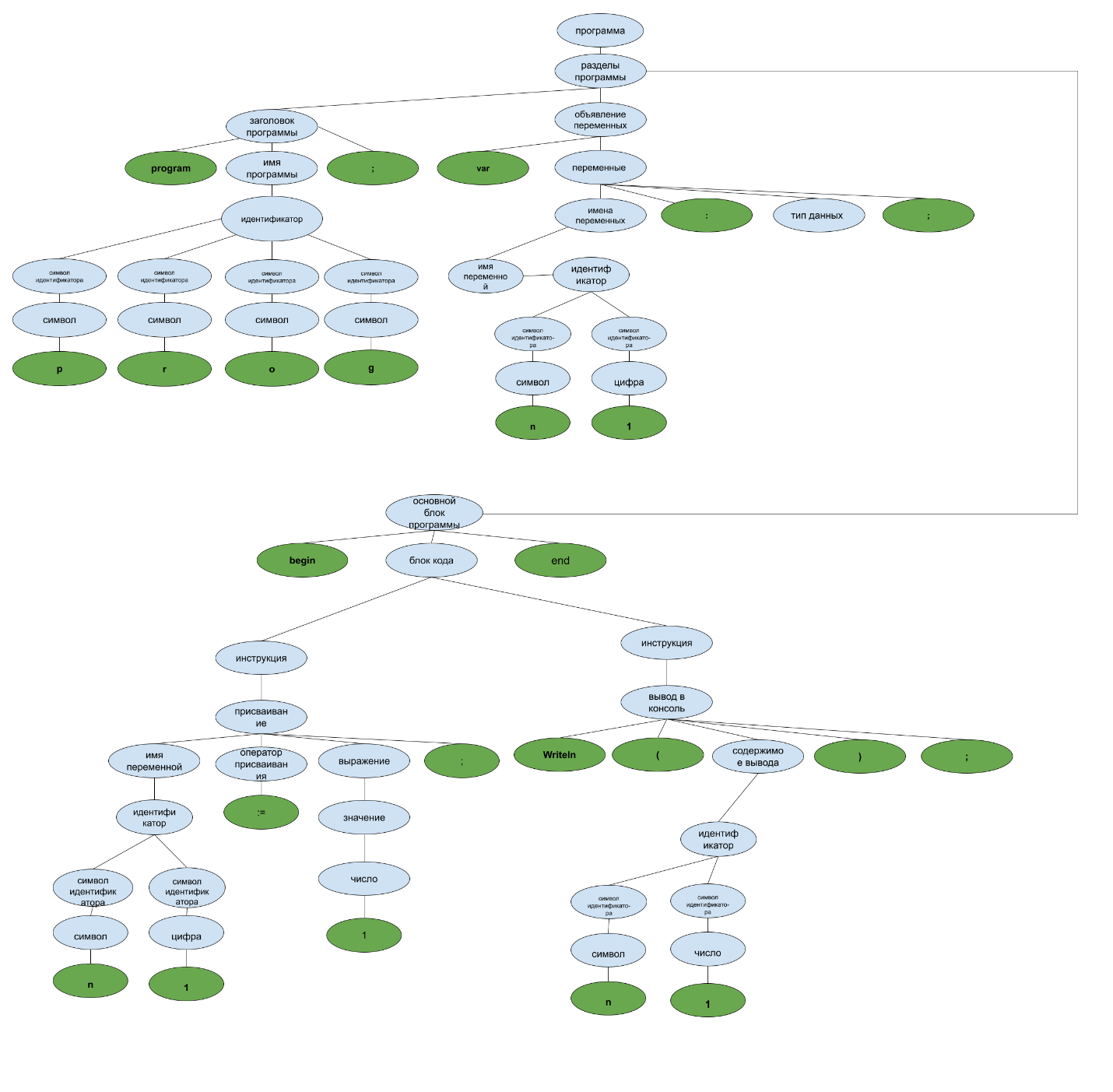


Рисунок 3 – Пример дерева разбора цепочки

# **Ошибки синтаксического анализатора**

Процесс синтаксического анализа может обнаружить ошибку, если встречена лексема, но не имеется продукции с этой лексемой в правой части в правилах грамматики, т.е. если встреченная лексема не может быть выведена в грамматике входного языка.

Таким образом, в процессе синтаксического анализа могут быть встречены ошибки:

1. Re-declaring variable exception – ошибка переопределения переменной: возникает, когда в текущем scope уже объявлена переменная и её пытаются переопределить
2. Undeclared variable exception – ошибка доступа к переменной: возникает, когда идёт обращение к переменной, которая ещё не была объявлена в текущем scope
3. Scope exception - ошибки области видимости, возникает, когда переменная используется вне своей области видимости или когда переменная с таким же именем уже объявлена в данной области
4. Subroutine exception - ошибки использования функций и процедур: к ним относятся вызов функции или процедуры с неправильным числом или типом аргументов, неправильным использованием возвращаемого значения и другие
5. Difference type exception – ошибка типов данных: возникает, когда тип данных, используемый в программе, не соответствует ожидаемому типу.

# **Заключение**

В ходе курсового проекта были решены следующие задачи:

1. Сформулирована неформальная постановка задачи.

2. Разработана грамматика подмножества входного языка TurboPascal.

3. Описаны контекстные условия языка TurboPascal.

4. Создан проект лексического анализатора, включающий в себя таблицы соответствий, ключевых и зарезервированных слов, типов лексем, конечный автомат.

5. Создан проект синтаксического анализатора, включающий в себя алгоритм синтаксического анализа, представление грамматики, проект алгоритма, пример дерева разбора для входной цепочки, анализ ошибок синтаксического анализатора.

# **Список литературы**

1. Ахо, А. В. Компиляторы: принципы, технологии и инструментарий компиляторов / А. В. Ахо, М. С. Лам, Р. Сети, Д. Д. Ульман. – М. : Вильямс, 2008. – 1184 с. URL: <https://e.lanbook.com/book/1298>.

2. Артемьева, И. Л. Теория вычислительных процессов и структур. Часть 1. Языки и способы их задания. Учебно-методическое пособие / И. Л. Артемьева. – Владивосток : Издательство Дальневосточного федерального университета, 2011. – 60 с.

3. Все о Паскале [электронный ресурс]. URL: [http://pascal.net.ru/Type-declarations.](http://pascal.net.ru/Type-declarations) [Дата обращения: 09.04.2024];

4. Руководство по языку программирования Python [электронный ресурс]. URL: <https://metanit.com/python/tutorial//> [Дата обращения: 09.04. 2024].

# **Исполнители работы**

В работе над проектом принимали участие студенты группы   
Б9121-09.03.04 Верховцов Д.О., Коринецкая Ю.В., Омельченко М.А.

Участниками коллектива выполнялись следующие задания:

|  |  |
| --- | --- |
| **ФИО** | **Разделы** |
| Верховцов Д.О. | Таблица соответствия (4?), 5.4, форматирование титула |
| Коринецкая Ю.В. | 1, 5.4, оформление |
| Омельченко М.А. | Всё остальное ;-) |