МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский Авиационный Институт (Национальный Исследовательский Университет)»

Институт №3 Системы управления, информатика и электроэнергетика

Кафедра 304 «Вычислительные машины, системы и сети»

Отчет по лабораторной работе №2

по учебной дисциплине

«Конструирование компиляторов»

Выполнили:

студенты группы М30-411Б-21

Нуриев Н.Н.

Макаренков М.В.

Принял преподаватель:

доцент каф.304, к.т.н., Охотников Д.А.

Оглавление

[Задание 3](#_Toc192198953)

[Алфавит 4](#_Toc192198954)

[Граф состояний 4](#_Toc192198955)

[Грамматика языка 5](#_Toc192198956)

[Таблица переходов конечного автомата 7](#_Toc192198957)

[Результаты тестирования 8](#_Toc192198958)

[Код программы 10](#_Toc192198959)

# Задание

Разработать модуль приема и обработки данных о территории страны.

Каждое сообщение включает в себя площадь всей страны, название страны, площади всех входящих в нее округов, названия округов. Для каждого округа передается площадь, входящих в него районов, названия районов.

Реализовать модуль управления структурой со следующим интерфейсом:

addChild(parent, child); // добавить дочерний узел.

printStruct; // распечатать структуру.

checkStruct; / проверить структуру на ошибки.

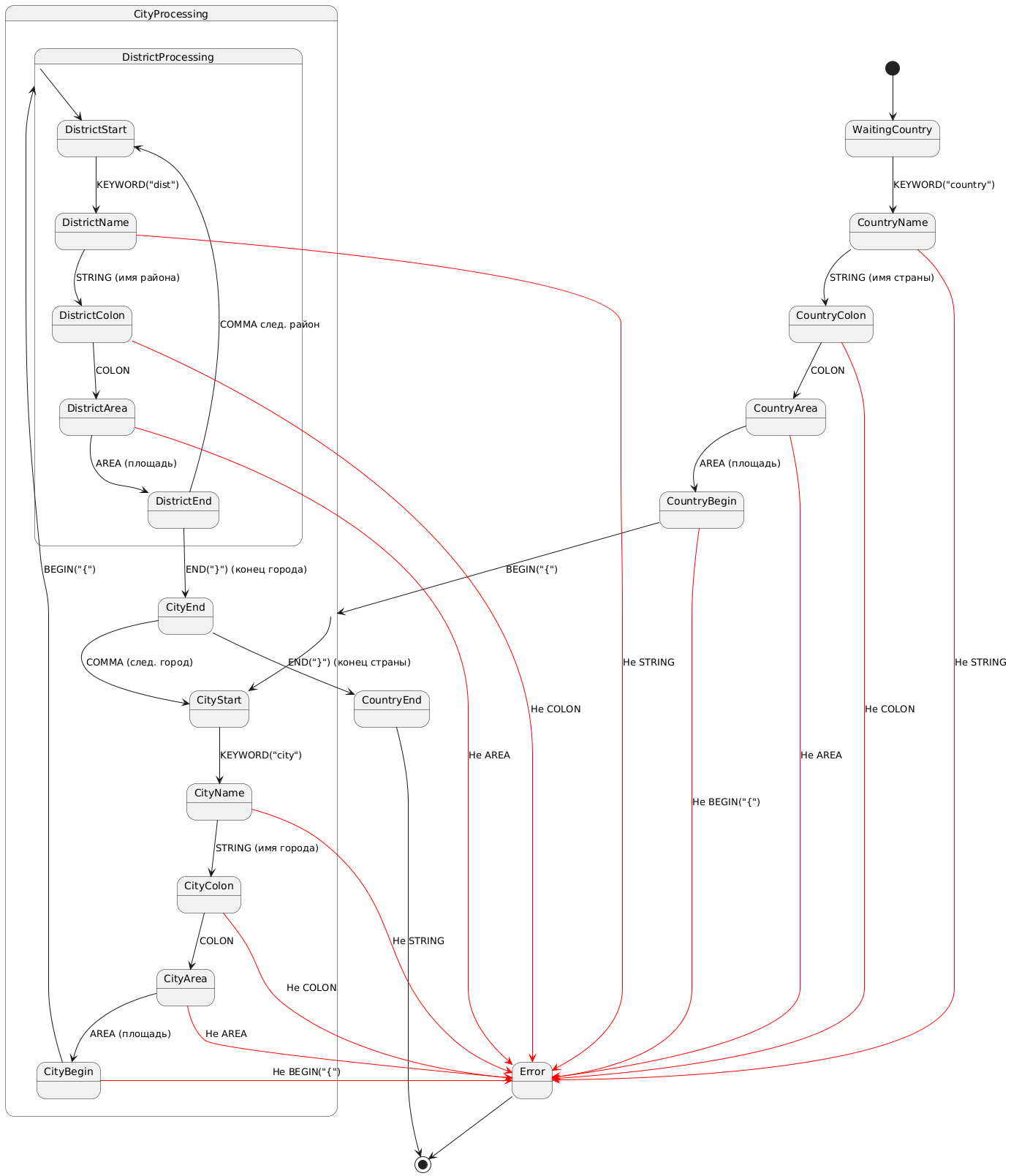
getRootNode; // возвращает указатель на корневой элемент структуры.

Площадь округа должна быть равна сумме площадей входящих в него районов. Площадь страны должна быть равна сумме площадей, входящих в нее округов. Площадь задается положительным числом, не превышающим значение 999. Количество округов в стране и районов в округе произвольное.

# Алфавит

T = { country city dist : , { } }

# Граф состояний



# Грамматика языка

WaitingCountry → country CountryName

CountryName → “name” CountryColon

CountryColon → : CountryArea

CountryArea → “digit” CountryBegin

CountryBegin → { CityStart

CityStart → city CityName

CityName → “name” CityColon

CityColon → : CityArea

CityArea → “digit” CityBegin

CityBegin → { DistrictStart

DistrictStart → dist DistrictName

DistrictName → “name” DistrictColon

DistrictColon → : DistrictArea

DistrictArea → “digit” DistrictEnd

DistrictEnd → , DistrcitStart

DistrictEnd → } CityEnd

CityEnd → , CityStart

CityEnd → } CountryEnd

CountryEnd → End

# Дерево разбора

Program

└── Country

├── "country"

├── "Russia"

├── ":"

├── "100"

├── "{"

├── CityList

│ └── City

│ ├── "city"

│ ├── "Moscow"

│ ├── ":"

│ ├── "60"

│ ├── "{"

│ ├── DistrictList

│ │ └── District

│ │ ├── "dist"

│ │ ├── "Center"

│ │ ├── ":"

│ │ └── "30"

│ └── "}"

└── "}"

# Результаты тестирования

Некорректные тесты:

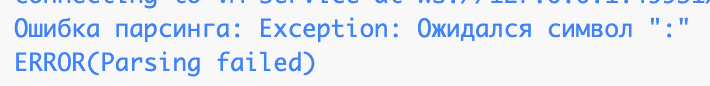
1. Не задана площадь страны и ‘:’ :

Входные данные:

Изображение выглядит как текст, Шрифт, снимок экрана, чек

Автоматически созданное описание

Результат:



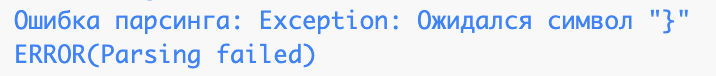
1. Нет закрыта страна } :

Входные данные:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Результат:



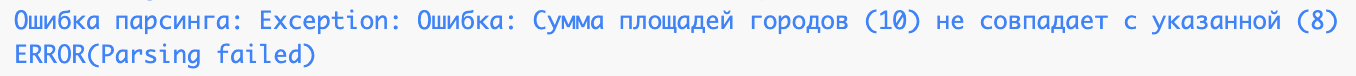
1. Несовпадение площадей

Входные данные:

Изображение выглядит как текст, Шрифт, снимок экрана, чек

Автоматически созданное описание

Результат:



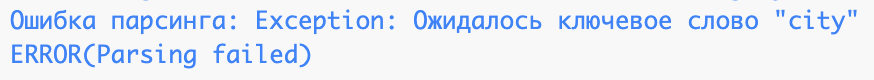
1. После запятой нет следующего города

Входные данные:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Результат:



Корректные тесты:

1. Входные данные:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Результат:

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

# Код программы

Лексер

class Token {

final String type;

final String value;

Token(this.type, this.value);

@override

String toString() => 'Token(type: $type, value: $value)';

}

class Lexer {

final String input;

int position = 0;

Lexer(this.input);

bool get isAtEnd => position >= input.length;

String get currentChar => isAtEnd ? '' : input[position];

void advance() => position++;

void skipWhitespace() {

while (!isAtEnd && currentChar.trim().isEmpty) {

advance();

}

}

Token nextToken() {

skipWhitespace();

if (isAtEnd) return Token('EOF', '');

if (currentChar == '{') {

advance();

return Token('BEGIN', '{');

}

if (currentChar == '}') {

advance();

return Token('END', '}');

}

if (currentChar == ':') {

advance();

return Token('COLON', ':');

}

if (currentChar == ',') {

advance();

return Token('COMMA', ',');

}

if (\_isLetter(currentChar)) {

String word = readString();

if (word == 'country' || word == 'city' || word == 'dist') {

return Token('KEYWORD', word);

}

return Token('STRING', word);

}

if (\_isDigit(currentChar)) {

return Token('AREA', readNumber());

}

return Token('ERR', 'Unexpected character: $currentChar');

}

bool \_isLetter(String char) => RegExp(r'[a-zA-Z]').hasMatch(char);

bool \_isDigit(String char) => RegExp(r'\d').hasMatch(char);

String readString() {

final buffer = StringBuffer();

while (!isAtEnd && \_isLetter(currentChar)) {

buffer.write(currentChar);

advance();

}

return buffer.toString();

}

String readNumber() {

final buffer = StringBuffer();

while (!isAtEnd && \_isDigit(currentChar)) {

buffer.write(currentChar);

advance();

}

return buffer.toString();

}

}

Парсер

import 'lexer.dart';

class Node {

final String type;

String name;

String area;

final List<Node> children = [];

Node(this.type, this.name, [this.area = '']);

void addChild(Node child) {

children.add(child);

}

int getTotalArea() {

if (children.isEmpty) {

return int.tryParse(area) ?? 0;

}

return children.fold(0, (sum, child) => sum + child.getTotalArea());

}

@override

String toString() {

return \_toStringWithIndent(0);

}

String \_toStringWithIndent(int indent) {

final indentStr = ' ' \* indent;

var result = '$indentStr$type($name${area.isNotEmpty ? ': $area' : ''})';

for (final child in children) {

result += '\n${child.\_toStringWithIndent(indent + 1)}';

}

return result;

}

}

class Parser {

final List<Token> tokens;

int position = 0;

Parser(this.tokens);

bool get isAtEnd => position >= tokens.length;

Token get currentToken => isAtEnd ? Token('EOF', '') : tokens[position];

void advance() => position++;

Node parse() {

try {

return parseCountry();

} catch (e) {

print('Ошибка парсинга: $e');

return Node('ERROR', 'Parsing failed');

}

}

Node parseCountry() {

if (currentToken.type != 'KEYWORD' || currentToken.value != 'country') {

throw Exception('Ожидалось ключевое слово "country"');

}

advance();

if (currentToken.type != 'STRING') {

throw Exception('Ожидалось имя страны');

}

String countryName = currentToken.value;

advance();

if (currentToken.type != 'COLON') {

throw Exception('Ожидался символ ":"');

}

advance();

if (currentToken.type != 'AREA') {

throw Exception('Ожидалась числовая область');

}

String countryArea = currentToken.value;

advance();

if (currentToken.type != 'BEGIN') {

throw Exception('Ожидался символ "{"');

}

advance();

Node countryNode = Node('COUNTRY', countryName, countryArea);

bool firstCity = true;

while (currentToken.type != 'END' && currentToken.type != 'EOF') {

if (!firstCity) {

if (currentToken.type == 'COMMA') {

advance();

} else {

throw Exception('Ожидалась запятая между городами');

}

}

countryNode.addChild(parseCity());

firstCity = false;

}

if (currentToken.type != 'END') {

throw Exception('Ожидался символ "}"');

}

advance();

int actualArea = countryNode.getTotalArea();

int expectedArea = int.parse(countryArea);

if (actualArea != expectedArea) {

throw Exception(

'Ошибка: Сумма площадей городов ($actualArea) не совпадает с указанной ($expectedArea)');

}

return countryNode;

}

Node parseCity() {

if (currentToken.type != 'KEYWORD' || currentToken.value != 'city') {

throw Exception('Ожидалось ключевое слово "city"');

}

advance();

if (currentToken.type != 'STRING') {

throw Exception('Ожидалось имя города');

}

String cityName = currentToken.value;

advance();

if (currentToken.type != 'COLON') {

throw Exception('Ожидался символ ":"');

}

advance();

if (currentToken.type != 'AREA') {

throw Exception('Ожидалась числовая область');

}

String cityArea = currentToken.value;

advance();

if (currentToken.type != 'BEGIN') {

throw Exception('Ожидался символ "{"');

}

advance();

Node cityNode = Node('CITY', cityName, cityArea);

bool firstDistrict = true;

while (currentToken.type != 'END' && currentToken.type != 'EOF') {

if (!firstDistrict) {

if (currentToken.type == 'COMMA') {

advance();

} else {

throw Exception('Ожидалась запятая между районами');

}

}

cityNode.addChild(parseDistrict());

firstDistrict = false;

}

if (currentToken.type != 'END') {

throw Exception('Ожидался символ "}"');

}

advance();

int actualArea = cityNode.getTotalArea();

int expectedArea = int.parse(cityArea);

if (actualArea != expectedArea) {

throw Exception(

'Ошибка: Сумма площадей районов ($actualArea) не совпадает с указанной ($expectedArea) для города $cityName');

}

return cityNode;

}

Node parseDistrict() {

if (currentToken.type != 'KEYWORD' || currentToken.value != 'dist') {

throw Exception('Ожидалось ключевое слово "dist"');

}

advance();

if (currentToken.type != 'STRING') {

throw Exception('Ожидалось имя района');

}

String districtName = currentToken.value;

advance();

if (currentToken.type != 'COLON') {

throw Exception('Ожидался символ ":"');

}

advance();

if (currentToken.type != 'AREA') {

throw Exception('Ожидалась числовая область');

}

String districtArea = currentToken.value;

advance();

return Node('DISTRICT', districtName, districtArea);

}

}

Main

import 'dart:io';

import 'lexer.dart';

import 'parser.dart';

void main() {

String input = File("main.bbr").readAsStringSync();

Lexer lexer = Lexer(input);

List<Token> tokens = [];

Token token;

while ((token = lexer.nextToken()).type != 'EOF') {

tokens.add(token);

// print(token);

}

Parser parser = Parser(tokens);

Node rootNode = parser.parse();

print(rootNode);

}