**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное ГОСУДАРСТВЕННОЕ бюджетное ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информатики и вычислительной техники

Кафедра ИиСП

**Курсовая работа**

**по дисциплине "Теория автоматов и формальных языков"**

**синтаксический анализатор кода**

Выполнили:

студенты группы: ПС-31

Колчин И. А.

Бочкарёв М. А.

Проверила:

Нехорошкова Л. Г.

г. Йошкар-Ола

2016

# Содержание

[Содержание 1](#_Toc472457541)

[Постановка задачи 2](#_Toc472457542)

[Правила языка 3](#_Toc472457543)

[Структура анализатора 5](#_Toc472457544)

[LL-LR бегунок 5](#_Toc472457545)

[LL таблицы 5](#_Toc472457546)

[Лексер 5](#_Toc472457547)

[Принцип работы 5](#_Toc472457548)

[Выводы 6](#_Toc472457549)

[Использованные инструменты 7](#_Toc472457550)

# Постановка задачи

Задача состоит в том, что нужно придумать свой собственный язык программирования, написать правила и программу, которая анализирует входной код и печатает : корректный код или нет.

# Правила языка

Грамматика принадлежит к LL-грамматикам. В грамматике есть условия, циклы, арифметические и логические операции, декларативный оператор, записи и вывода значений.

Курсовая работа разрабатывалась с применением техники TDD(test-driven development). Сначала писался тест для нового функционала, потом код или грамматическое правило, таблица для него. Постепенно добавлялись тесты и правила грамматики. Для каждого правила было написано несколько тестов.

start\_program : toplevel\_list

toplevel\_list : toplevel\_line right\_part\_toplevel\_list

right\_part\_toplevel\_list := toplevel\_list | ничего

toplevel\_line : COMMAND\_SEPARATOR | toplevel\_statement COMMAND\_SEPARATOR

toplevel\_statement : function\_declaration

function\_declaration := type\_reference ID parenthesis\_parameter\_list block

parenthesis\_parameter\_list := START\_LIST\_ARGUMENTS right\_part\_parenthesis\_parameter\_list

right\_part\_parenthesis\_parameter\_list := END\_LIST\_ARGUMENTS | parameter\_list END\_LIST\_ARGUMENTS

type\_reference := NAME\_FLOAT | NAME\_INTEGER | NAME\_CHAR | NAME\_STRING | NAME\_LOGIC

block := START\_BLOCK right\_part\_block

right\_part\_block := statement\_list END\_BLOCK | END\_BLOCK | ничего

statement\_line := statement COMMAND\_SEPARATOR

statement\_list := statement\_line right\_part\_statement\_list

right\_part\_statement\_list := statement\_line statement\_list | ничего

parameter\_decl := type\_reference ID

parameter\_list := parameter\_decl right\_part\_parameter\_list

right\_part\_parameter\_list := VARIABLE\_SEPARATOR parameter\_list | ничего

function\_call : ID START\_LIST\_ARGUMENTS variable\_list END\_LIST\_ARGUMENTS

variable\_list : variable right\_part\_variable\_list | ничего

right\_part\_variable\_list : VARIABLE\_SEPARATOR variable | ничего

variable : constant | ID

constant := BOOL | INT | FLOAT | CHAR | STRING

expression := constant | ID | function\_call

| START\_LIST\_ARGUMENTS expression END\_LIST\_ARGUMENTS

| expression EQUALS expression

| expression PLUS expression

| expression MINUS expression

| expression STAR expression

| expression DIVIDE expression

| expression LOGIC\_AND expression

| expression LOGIC\_OR expression

statement := PRINT START\_LIST\_ARGUMENTS expression END\_LIST\_ARGUMENTS

| READ START\_LIST\_ARGUMENTS expression END\_LIST\_ARGUMENTS

| type\_reference ID ASSIGN expression

| ID ASSIGN expression

| NAME\_RETURN expression

| IF\_OPERATOR START\_LIST\_ARGUMENTS expression END\_LIST\_ARGUMENTS block

else\_part

| WHILE\_OPERATOR START\_LIST\_ARGUMENTS expression END\_LIST\_ARGUMENTS block

else\_part := ELSE\_OPERATOR block | ничего

Пример программы:

float func(bool variable)

{

return 2.2;

};

int main()

{

bool condition = true;

int number = 1;

if(condition)

{

number = number + func();

}

else

{

number = number - 9;

};

float number2 = 0;

read(number2);

print(number2);

};

# Структура анализатора

Анализатор содержит в себе:

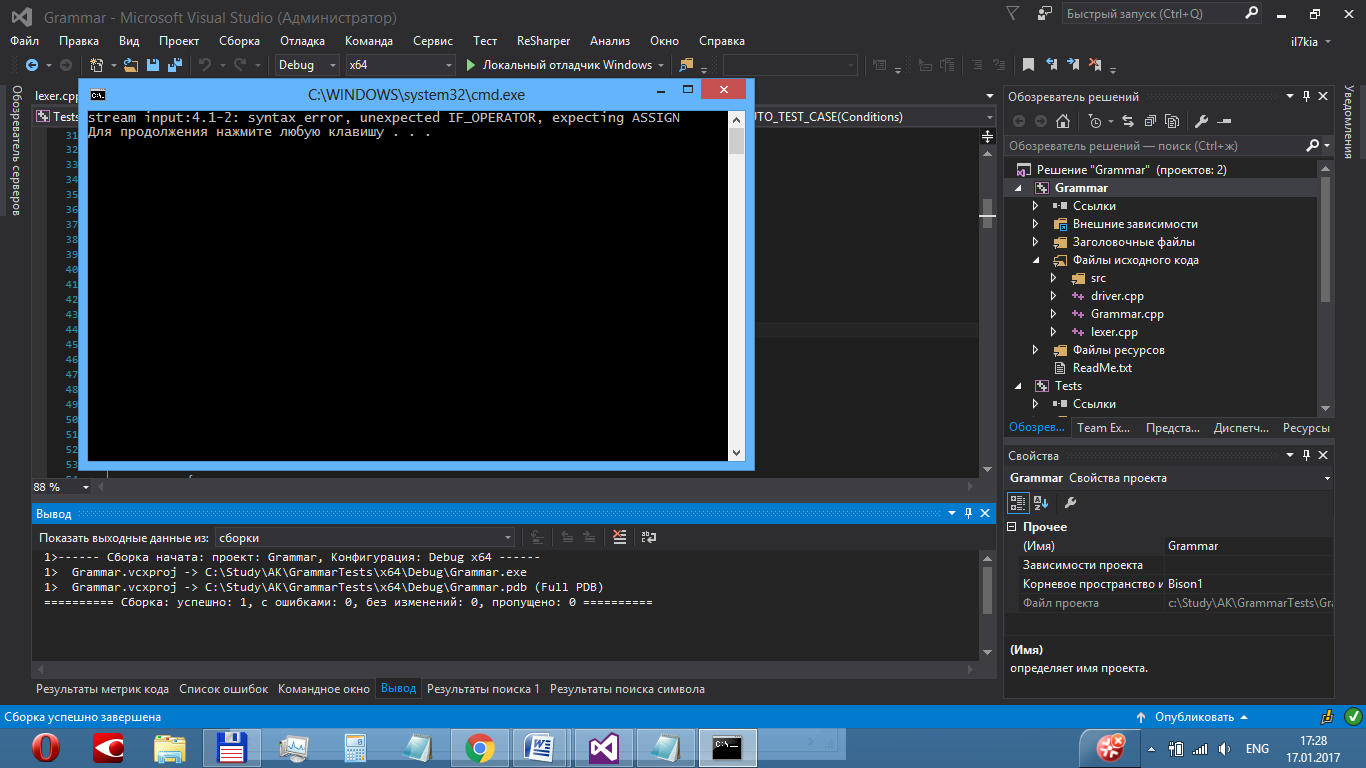
1. LL-LR бегунок
2. LL таблицы
3. Лексер

## LL-LR бегунок

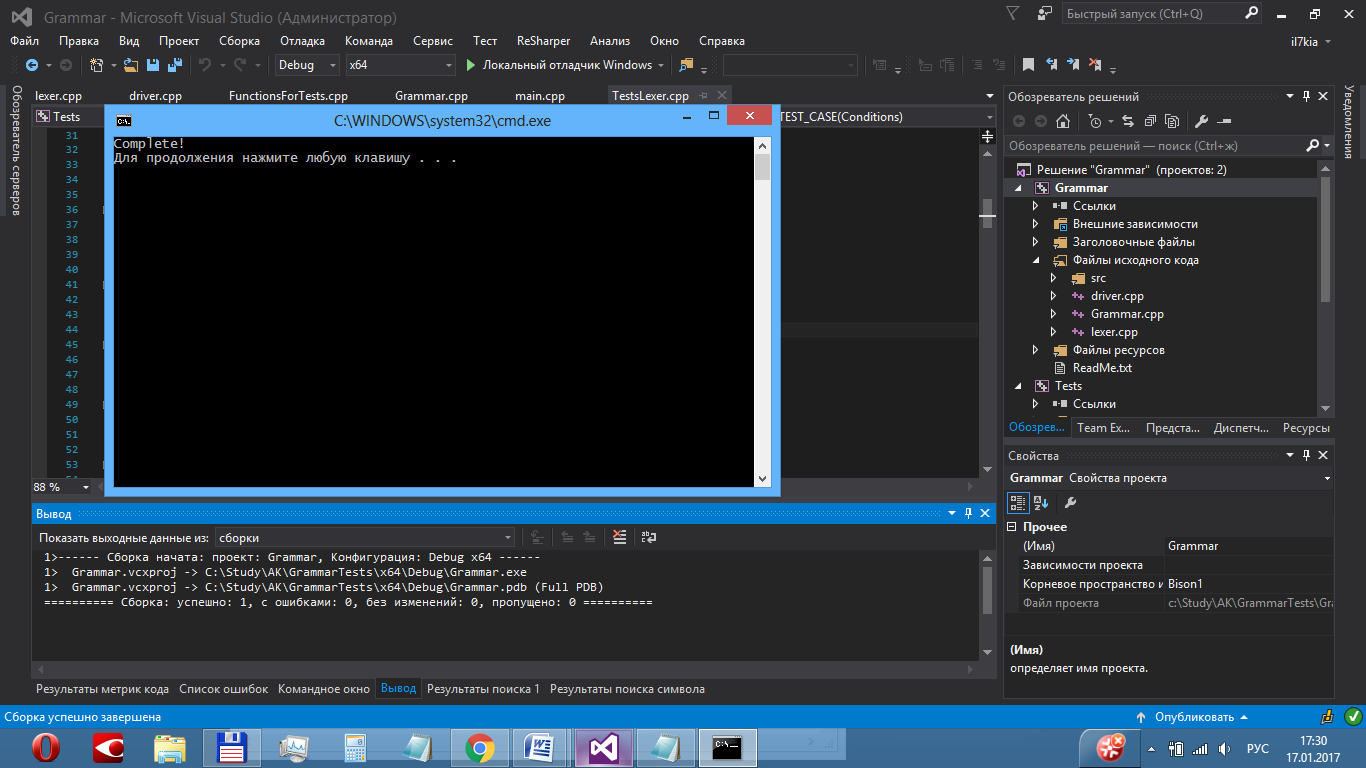
Он позволяет использовать LL или LR таблицу для анализа, что упростит разработку анализатора и позволит оптимизировать некоторые части грамматики. Например, если в грамматике есть привила, которые невозможно переписать с LR на LL, или переписывание значительно усложнит разработку, то наш анализатор решает этот вопрос следующим образом: можно переключатся между LL и LR таблицами при анализе.

## LL таблицы

**Пояснение**: только LL таблицы, LR таблиц нет, их, к сожалению, не успели сделать, было проще сделать всё на LL.

С использованием этих таблиц производится анализ. В случае ошибки выводится сообщение об ошибке. Например:

Наш анализатор может считывать данные из файла или из другого потока ввода(например, стандартный поток ввода в консоли), поэтому в начале написано stream input с указанием строки и позиции.

Если всё прошло успешно выводит:

## Лексер

Распознаёт лексемы ,и на их основе формирует токены для бегунка.

# Принцип работы

Сначала файл проходит через лексер. Потом лексер передаёт список токенов бегунку. Бегунок анализирует код, используя таблицы. В случае ошибки пишет какие были ошибки, если ошибок не было пишет "Complete!".

# Выводы

Благодаря, данной курсовой работе мы поняли как устроены и как создавать синтаксические анализаторы. Также мы узнали о разных видах грамматик и их особенностях, что тоже было нами использовано при разработке анализатора. К сожалению, мы использовали только LL, хотя наш бегунок работает и с LR тоже. Гибридный вариант, позволил бы сделать нашу грамматику более читаемой, но нам не хватило бы времени. LL таблицы гораздо быстрее пишутся, поэтому мы решили написать грамматику на LL.

# Использованные инструменты

1. Язык C++, IDE Visual studio
2. Библиотека Boost (C++)(для тестов)