**Введение**

Цель курсового проекта – разработка учебного транслятора LAO-2017.

Транслятор – программа, преобразующий исходный код на одном языке программирования в исходный код на другом языке – целевом языке.

Исходный код — текст программы, написанный на языке программирования.

Язык программирования — формальная знаковая система, предназначенная для записи компьютерных программ. Знаковая система определяет набор лексических, синтаксических и семантических правил написания программы.

Для достижения цели были поставлены следующие задачи:

1. Разработка лексического анализатора, используя конечные автоматы, описанная в главе 3.
2. Разработка синтаксического анализатора, используя автомат с магазинной памятью, описанная в главе 4.
3. Разработка семантического анализатора, описанная в главе 5.
4. Разработка программы по преобразованию выражений в обратную польскую запись, описанная в главе 6.
5. Создание генератора кода на языке Assembler, описанное в главе 7.

**Глава 1. Спецификация языка программирования**

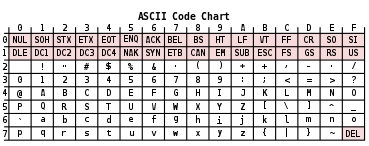
* 1. **Характеристика языка программирования**

Язык программирования LAO-2017 является процедурным, универсальным и строго типизированным, не объектно-ориентированным.

* 1. **Алфавит языка**

Базовая таблица символов для языка LAO-2017 представлена на рис. 1.1 (ASCII, 8 bit).

Рис. 1.1 Базовая таблица символов языка LAO-2017



В языке LAO-2017 используются не все символы представленной кодировки. Запрещенные символы представлены в таблице 1.1

Табл. 1.1 Запрещенные символы языка LAO-2017

|  |  |
| --- | --- |
| Символ | Символ в 16 с/с |
| Управляющие символы | 0x00 – 0x08, 0x0b – 0x1f, 0x7f |
| ! | 0x21 |
| # | 0x23 |
| $ | 0x24 |
| % | 0x25 |
| & | 0x26 |
| ' | 0x27 |
| @ | 0x40 |
| [ | 0x5b |
| \ | 0x5c |
| ] | 0x5d |
| ^ | 0x5e |
| \_ | 0x5f |
| ` | 0x60 |
| | | 0x7c |
| ~ | 0x7e |

Если был допущен запрещенный символ – транслятор игнорирует его, выдавая об этом сообщение в протокол. Ошибка не приводит к остановке транслятора.

* 1. **Применяемые сепараторы**

Язык LAO-2017 разрешает использовать сепараторы для написания исходного кода. Они представлены в табл. 1.2

Табл. 1.2 Символы-сепараторы языка LAO-2017

|  |  |  |
| --- | --- | --- |
| **Символ** | **Символ в 16с/с** | **Описание** |
| ( ) | 0x28 ; 0x29 | Приоритетность операций, параметры |
| ( ) | 0x28 ; 0x29 | Параметры |
| { } | 0x7B, 0x7D | Программный блок |
| "" | 0x22 | Строковый литерал |
| ; | 0x3B | Разделитель инструкций |
| Пробел | 0x20 | Разделитель инструкций |
| , | 0x2c | Разделитель инструкций, параметров функций |

* 1. **Применяемые кодировки**

Язык LAO-2017 использует кодировку ASCII, описанную в п.1.2.

* 1. **Типы данных**

В языке LAO-2017 разрешены типы данных, представленные в табл. 1.3

Табл. 1.3 Типы данных языка LAO-2017

|  |  |
| --- | --- |
| **Тип** | **Описание** |
| num | Целочисленный, беззнаковый тип данных, по умолчанию равен 0. |
| string | Строковый тип данных. Максимально 255 символов. Последний символ \0 – окончание строки. |

* 1. **Преобразование типов данных**

Язык LAO-2017 не поддерживает преобразование типов данных.

* 1. **Идентификаторы**

В языке LAO-2017 разрешается использовать идентификаторы.

Идентификатор должен быть написан на допустимых символах языка, описание которых приведено в п.1.2. Имя идентификатора может содержать цифры, но они не должны находиться на первой позиции. Максимальная длина идентификатора переменной – 5, а функции – 10. При превышение максимальной длины идентификатора, оно урезается. Идентификатор не должен совпадать с ключевыми словами языка.

* 1. **Литералы**

В языке LAO-2017 поддерживаются 2 вида литералов: целочисленные и строковые. Краткое описание литералов приведено в табл. 1.4

Таблица 1.4 Краткое описание литералов

|  |  |
| --- | --- |
| **Литерал** | **Описание** |
| Целочисленный num | Не имеют дробных частей или экспонент |
| Строковый string | Если символ заключен в двойные кавычки, то он инициализируется как string |

В языке LAO-2017 литералы могут быть только rvalue.

* 1. **Объявление данных и область видимости**

Операторы могут быть объявлены только локально. В языке LAO-2017 область видимости сверху вниз, слева направо.

* 1. **Инициализация данных**

В языке LAO-2017 используется следующий синтаксис объявления переменных: var <тип данных> <идентификатор>;

Для инициализации переменной используется следующая инструкция:

<идентификатор> = <значение>;

По умолчанию переменная типа num инициализируется нулем, а переменная типа string – строкой нулевой длины.

* 1. **Инструкции языка**

Язык LAO-2017 предусматривает инструкции языка, описанные в табл. 1.4

Табл. 1.4 Инструкции языка LAO-2017

|  |  |
| --- | --- |
| Инструкция | Назначение |
| var <тип данных>; <идентификатор>=<значение>; | Объявление переменной |
| function<тип данных><идентификатор>(<тип данных><идентификатор>,…) | Объявление внешних функций |
| <идентификатор>=<значение>; | Присвоение значений |
| console(<литерал>/<идентификатор>); | Потоковый вывод |
| return <литерал>/<идентификатор>; | Возврат из подпрограммы |
| if (<integer-идентификатор> / <integer-литерал> < <integer-идентификатор> / <integer-литерал>)  {  …  }  else  {  …  } | Условная инструкция |

В языке LAO-2017 вложенные условные конструкции не допускаются. Одновременное объявление и присвоение значения переменной не предусмотрено.

* 1. **Операции языка**

В языке LAO-2017 предусмотрены операции, описанные в табл. 1.5.

Табл. 1.5 Операции языка LAO-2017

|  |  |
| --- | --- |
| **Операция** | **Описание** |
| + | Бинарный, сложение |
| - | Бинарный, разность |
| \* | Бинарный, умножение |
| ( ) | Приоритет |
| < | Бинарный, меньше |

Данные операции имеют приоритет. Максимальное значение приоритета, «1», принимают ( ), приоритет «2» имеет операция умножения, а минимальный, «3» - сложение и разность. Бинарная операция «меньше» не имеет приоритета.

* 1. **Выражения и их вычисления**

Выражения могут быть записаны в несколько строк. Круглые скобки в выражении используются для изменения приоритета операций. Также не допускается запись двух подряд идущих арифметических операций.

* 1. **Программные конструкции языка**

В языке LAO-2017 можно использовать конструкции языка, описанные в табл. 1.6

Табл. 1.6 Конструкции языка LAO-2017

|  |  |
| --- | --- |
| **Конструкция** | **Описание** |
| enter  {  <текст программы>  return <литерал>/<идентификатор>;  } | Главная функция (точка входа) |
| if(<условие>)  {  <текст программы>  }  else  {  <текст программы>  } | Оператор условия |
| function<тип данных><идентификатор>(<тип данных><идентификатор>,…)  {  <текст программы>  return <литерал>/<идентификатор>;  } | Функция |

Применение данных конструкций представлено в контрольном примере, который находится в п.1.25.

* 1. **Область видимости идентификаторов**

Язык LAO-2017 поддерживает локальную внутри программных блоков функций область видимости идентификаторов.

* 1. **Семантические проверки**

Семантические проверки позволяют нам проверить то, что не выполняется на этапе лексического и синтаксического анализа. Перечень семантических проверок представлен в таблице 1.7

Табл. 1.7 Семантические проверки

|  |  |
| --- | --- |
| **№** | **Проверка** |
| 1 | Несоответствие передаваемых параметров функции |
| 2 | Повторное объявление идентификатора |
| 3 | Ошибка в возвращаемом значении |
| 4 | Отсутствие передаваемых параметров в функции |
| 5 | Тип данных результата выражения не соответствует присваиваемому идентификатору |

* 1. **Распределение оперативной памяти на этапе выполнения**

Все переменные размещаются в куче.

* 1. **Стандартная библиотека и ее состав**

В таблице 1.7 приведено описание функций стандартной библиотеки.

Табл. 1.7 Описание функций стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| consolen | 0 | num a - число | Функция выводит на консоль число a |
| consoles | 0 | str a - строка | Функция выводит на консоль строку a |
| sum | a+b | num a, num b - числа | Функция выводит в консоль сумму a и b |

Стандартная библиотека написана на языке программирования C++.

* 1. **Ввод и вывод данных**

Средства вывода описаны в табл. 1.4. Средства ввода в языке LAO-2017 не предусмотрены.

* 1. **Точка входа**

Язык LAO-2017 должен содержать только одну точку входа, которая определяется наличием функции enter. При инициализации более одной или менее одной – выдается ошибка лексического анализатора.

* 1. **Препроцессор**

В языке LAO-2017 препроцессор не предусмотрен.

* 1. **Соглашения о вызовах**

В языке LAO-2017 в качестве соглашения о вызовах функций используется stdcall.

Особенности stdcall:

-все параметры функции передаются через стек;

-память высвобождает вызываемый код;

-занесение в стек параметров идёт справа налево.

* 1. **Объектный код**

Для языка LAO-2017 предусмотрен принцип построения объектного кода на основе промежуточного кода с помощью Microsoft Micro Assembler.

* 1. **Классификация сообщений транслятора**

В табл. 1.8 описана префиксная классификация сообщений транслятора для языка LAO-2017.

Табл. 1.8 Префиксная классификация сообщений транслятора LAO-2017

|  |  |
| --- | --- |
| **Префикс** | **Описание** |
| [LA] | Префикс ошибки лексического анализатора |
| [SA] | Префикс ошибки синтаксического анализатора |
| [SemA] | Префикс ошибки семантического анализатора |
| [SYSTEM] | Префикс фатальной ошибки |
| [IN] | Префикс ошибки входных данных |
| [PARM] | Префикс входных параметров |

Префиксная классификация позволяет понять, на каком этапе произошла ошибка.

Табл. 1.9 Системные(фатальные) сообщения транслятора LAO-2017

|  |  |
| --- | --- |
| **Номер ошибки** | **Описание** |
| 0 | Недопустимый код ошибки |
| 1 | Системный сбой |
| 2 | Недопустимое количество ошибок |

Фатальные ошибки возникают при необработанном исключении.

* 1. **Контрольный пример**

Пример программы на языке LAO-2017:

function num razn(num a, num b)

{

var num c;

c=a-b;

return c;

}

enter

{

var num i;

i = 5;

var num t;

t = 1;

var num k;

k = razn(i,t);

console("k");

console(k);

if(i<0)

{

console("less0");

}

else

{

console("more0");

}

var string v;

v ="first";

console("second");

var num s;

s = sum(2,2);

console(s);

return 0;

}

**Глава 2. Структура транслятора**

**2.1 Компоненты транслятора, их назначение и принципы взаимодействия**

Схема, демонстрирующая работу транслятора LAO-2017, представлена на рис. 2.1

Рис. 2.1 Схема транслятора LAO-2017



Первая фаза работы транслятора – лексический анализ.

На вход лексического анализатора поступает исходный код. В результате лексического анализа формируется таблица лексем и таблица идентификаторов.

Вторая фаза работы транслятора – синтаксический анализ.

На вход синтаксического анализатора поступает таблица лексем. В результате синтаксического анализа формируется дерево разбора.

Семантический анализ следует за синтаксическим. На вход семантического анализатора поступает таблица лексем и таблица идентификаторов.

На генератор промежуточного кода поступает таблица лексем, таблица идентификаторов и дерево разбора. На выходе формируется промежуточный код.

На вход генератора объектного кода поступает промежуточный код. На выходе формируется объектный код.

На всех этапах работы транслятора формируется протокол работы.

**2.2 Перечень входных параметров транслятора**

В трансляторе LAO-2017 предусмотрены входные параметры, представленные в табл. 2.1.

Табл. 2.1 Входные параметры транслятора LAO-2017

|  |  |
| --- | --- |
| **Входной параметр** | **Описание** |
| -in:\*.txt | Обязательный параметр. Указывает транслятору, где находится исходный код. |
| -log: | Необязательный параметр. Указывает транслятору, в какой файл выводить протокол работы транслятора. При отсутствии за основу берется параметр –in: |
| -out: | Имя файла объектного кода |
| -lex | Необязательный параметр. Вывод таблицы лексем в файл протокола |
| -id | Необязательный параметр. Вывод таблицы идентификаторов в файл протокола |
| -tree | Необязательный параметр. Вывод дерева разбора в протокол. |

При отсутствии обязательных параметров генерируется исключение. Дальнейшая работа транслятора невозможна.

**2.3 Перечень протоколов, формируемых транслятором и их содержимое**

В языке LAO-2017 транслятор формирует протоколы работы, описанные в таблице 2.2.

Табл. 2.2 Описание протоколов транслятора LAO-2017

|  |  |
| --- | --- |
| **Протокол** | **Назначение** |
| Лексического и синтаксического анализатора | Формирует в протокол таблицу лексем (ТЛ) и таблицу идентификаторов (ТИ). Каждый идентификатор ТЛ ссылается на ТИ.  ТЛ формируется как входная таблица синтаксического анализатора, ТИ формируется для всего транслятора.  Формирует в протокол пошаговую работу магазинного автомата с деревом разбора, для последующего разбора генератором кода. |
| Генератора кода | Формирует список структур промежуточного кода по таблице лексем, полученную из лексического анализатора. Из структур формируется объектный код ассемблера |

В протоколе работы лексического и синтаксического анализатора можно увидеть краткую информацию об исходном коде на языке LAO-2017.

**ГЛАВА 3. РАЗРАБОТКА ЛЕКСИЧЕСКОГО АНАЛИЗАТОРА**

**3.1 Структура лексического анализатора**

Лексический анализатор (сканер) – это программа, которая читает исходный код и выделяет в нем лексемы входного языка.

Схема работы лексического анализатора для языка LAO-2017 приведена на рис. 1.3.

Рис. 3.1 Схема работы лексического анализатора

Исходный

код

Лексический анализатор

Таблица лексем

Таблица идентификаторов

Обязательный параметр для лексического анализатора – файл с исходным кодом.

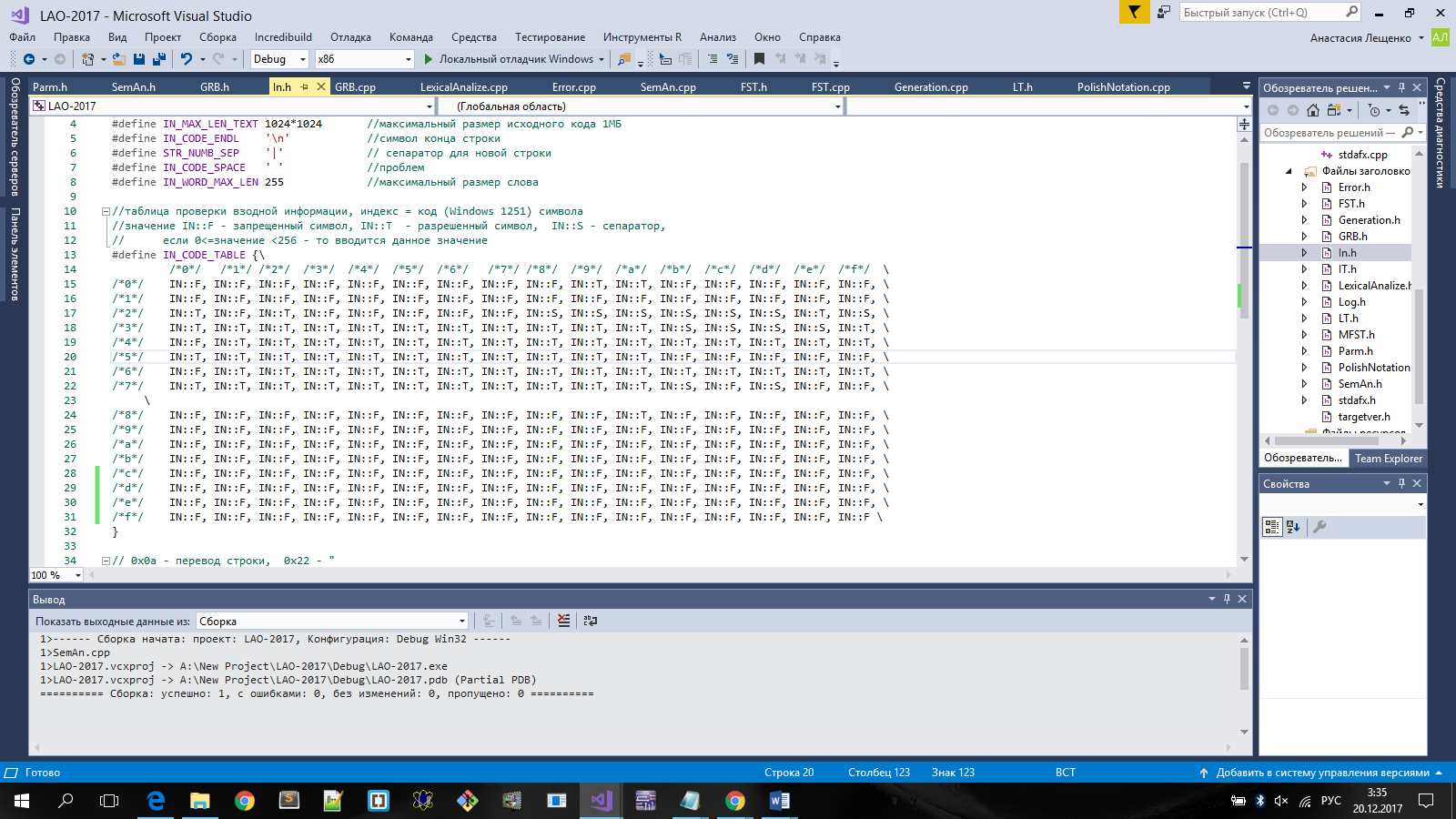
Выходными параметрами являются таблицы лексем и идентификаторов, которые заполняются в процессе лексического анализа.

**3.2 Контроль входных символов**

На языке LAO-2017 исходный файл посимвольно проверяется на допустимость.

Таблица контроля входных символов представлена на рис. 3.2

Табл. 3.2



Принцип работы таблицы состоит в следующем: каждому элементу в таблице соответствует значение в 16 с/с такой же, как и в таблице кодировок ASCII, представленной на рис. 1.1.

Каждый символ проверяется на допустимость. Если символ имеет метку T или S, то он записывается в строку char\*, в случае F – позиция данного символа записывается в структуру ошибок и на консоль выводится информация об ошибке. Максимальное количество недопустимых символов для языка LAO-2017 составляет 10.

**3.3 Удаление избыточных символов**

Алгоритм удаления избыточных символов для языка LAO-2017 не предусмотрен, так как все единицы языка, на которые он разбивается в процессе лексического анализа, записываются в очередь.

**3.4 Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем, регулярных выражений и конечных автоматов**

Каждой единице языка соответствует своя лексема. Данное соотношение для языка LAO-2017 представлено в таблице 3.1

Табл.3.1 Соотношение единиц языка и лексем

|  |  |  |
| --- | --- | --- |
| Единица языка | Лексема | Примечание |
| num | n | Целочисленный тип данных |
| string | s | Строковый тип данных |
| var | v | Объявление переменной |
| function | f | Объявление функции |
| return | r | Возврат значения из функции |
| enter | e | Главная функция |
| console | c | Функция вывода переменной целочисленного и строкового типа |
| if | h | Условная конструкция |
| else | q |
| + | + | Знаки арифметических операций |
| - | - |
| \* | \* |
| = | = | Оператор присваивания |
| < | < | Условный оператор |
| { | } | Блок условной конструкции и функции |
| } | } |
| ( | ( | Изменение приоритетности в выражении и отделение параметров функций |
| ) | ) |
| ; | ; | Сепараторы |
| , | , |

Каждой фразе соответствует автомат, по которому происходит разбор выражения. Перечень конечных автоматов представлен в приложении А.

На рис. 3.3 представлен пример графа перехода состояний КА.

Рисунок 3.3

m

S3

u

n

S2

S1

S0

Аналогичным образом можно построить автоматы для КА для остальных ключевых слов языка LAO-2017.

**3.5 Основные структуры данных**

Основные структуры данных лексического анализатора для языка LAO-2017 приведены в приложении Б.

Основная структура данных – TABLES. Она содержит экземпляр таблицы лексем и таблицы идентификаторов.

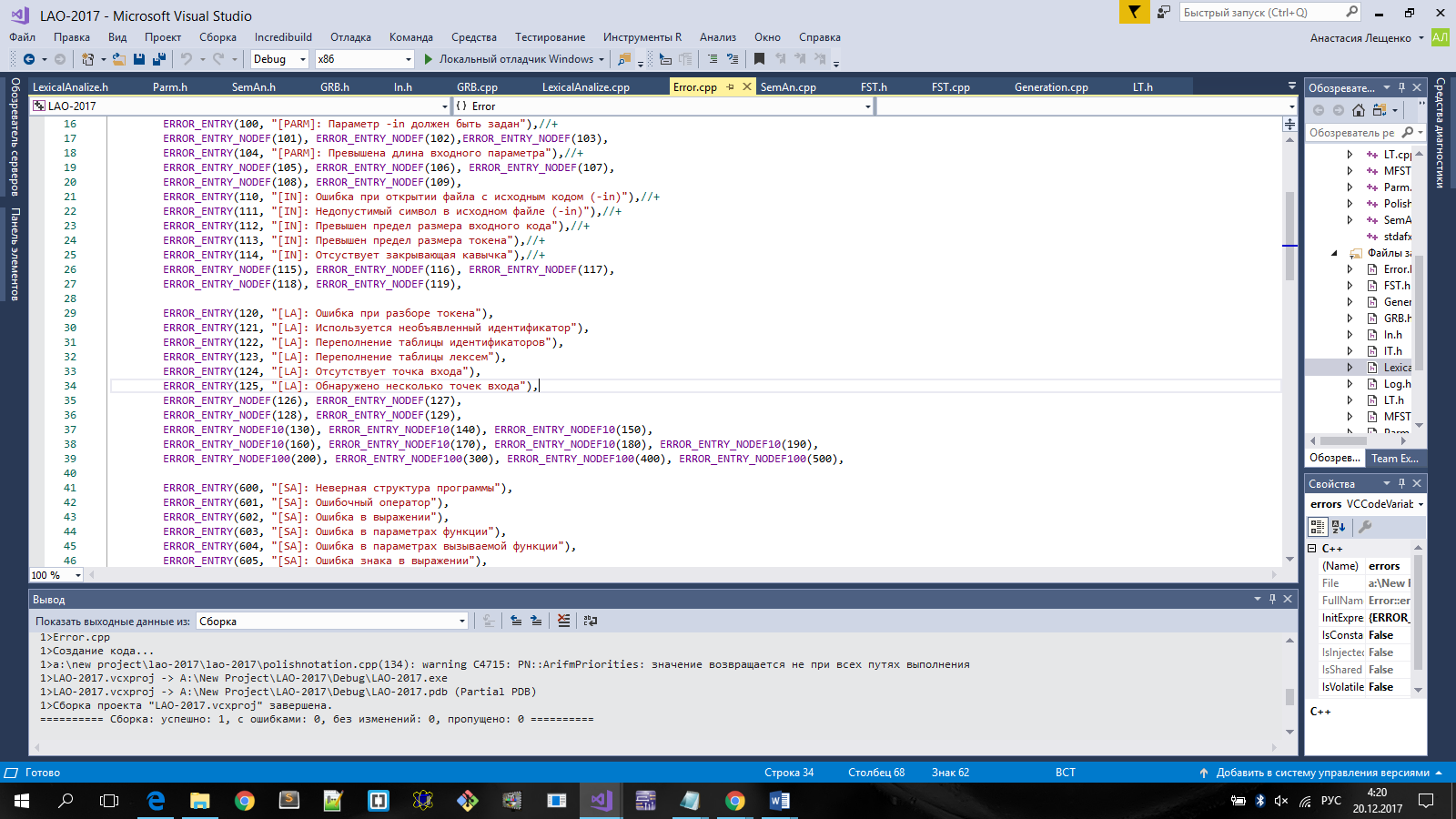
**3.6 Принцип обработки ошибок**

При обнаружении ошибки, работа транслятора LAO-2017 прекращает свою работу. Код ошибки и информация о ней выводится на консоль и в протокол.

**3.7 Структура и перечень сообщений лексического анализатора**

Перечень и содержание ошибок лексического анализатора представлен на рисунке 3.4

Рисунок 3.4



Здесь представлены номер ошибки, префикс и сообщение об ошибке лексического анализатора.

**3.8 Параметры лексического анализатора и режимы его работы**

Входным параметром лексического анализа являются две очереди, сформированные на этапе разбора исходного кода на допустимые символы.

**3.9 Алгоритм лексического анализа**

Изначально лексический анализатор пробует разобрать токен при помощи набора конечных автоматов. В случае, если автомат не был подобран, запоминается номер строки, в которой находился этот токен и в последствии будет выведено сообщение об ошибке. Если токен разобран, то дальнейшие действия, которые будут с ним производиться, будут зависеть от того, чем он является.

В случае, если токен является знаком арифметической операции либо функцией стандартной библиотеки, то он заносится в таблицу идентификаторов.

В случае, если является литералом, то заносится в таблицу идентификаторов с именем “literal<n>”, где n является номером литерала.

Когда встречаем токен, являющийся ключевым словом, которое отвечает за тип данных, заносим лексему, соответствующую ему, в таблицу лексем и запоминаем тип данных, которому он соответствует. В последствии, когда встречаем идентификатор, заносим его в таблицу идентификаторов с соответствующим ему типом данных и именем вида “xxxxxyyyzz”, где x – символы из имени идентификатора, y – постфикс из имени функции, в которой он находится и z – номер функции.

**3.10 Контрольный пример**

Распечатка таблиц лексем и идентификаторов, полученных в протоколе после разбора представлена в приложении В.

**ГЛАВА 4. РАЗРАБОТКА СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА**

**4.1 Структура синтаксического анализатора**

Синтаксический анализатор – программа, которая выполняет синтаксический анализ.

Схема работы синтаксического анализатора представлена на рис. 4.1

Стрелки сверху-вниз, слева-направо.

Рис. 4.1 Схема работы синтаксического анализатора



Входным параметром синтаксического анализатора является таблица лексем.

Результатом работы синтаксического анализатора является дерево разбора.

**4.2 Контекстно-свободная грамматика, описывающая синтаксис языка**

Синтаксис языка LAO-2017 описывается при помощи контекстно-свободной грамматики.

Контекстно-свободная грамматика – грамматика типа 2 по иерархии Хомского. Данная грамматика имеет вид ,

где

T – множество терминальных символов,

N – множество нетерминальных символов,

P – множество правил переходов,

S – стартовый символ.

В контекстно-свободной грамматике правила имеют вид: ,

где

,

,

 - словарь грамматики .

Грамматика языка LAO-2017 представлена в приложении Г.

В структуре терминальные символы помечены словом TS и все они являются сепараторами, знаками арифметических операций, либо строчными буквами. Нетерминальные же символы помечены словом NS и являются заглавными буквами латинского алфавита.

**4.3 Построение конечного магазинного автомата**

Автомат с магазинной памятью — это конечный автомат, который использует стек для хранения состояний.

Формальное описание МП-автомата:



 - множество состояний;

 - алфавит входных символов;

 - специальный алфавит магазинных символов;

-функция переходов автомата , где  - множество подмножеств ;

 - начальное состояние автомата;

- начальное состояние магазина (маркер дна);

- множество конечных состояний.

**4.4 Основные структуры данных**

Основные структуры данных описаны в приложении 1.

**4.5 Описание алгоритма синтаксического разбора**

Упрощенный алгоритм синтаксического разбора представлен на рис. 4.2



Рис. 4.2

Принцип работы синтаксического разбора транслятора LAO-2017 приведен ниже.

1. В магазин записывается стартовый символ.

2. На основе полученной таблицы лексем формируется входная лента.

3. Запускается автомат и выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке.

4. Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется с ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала.

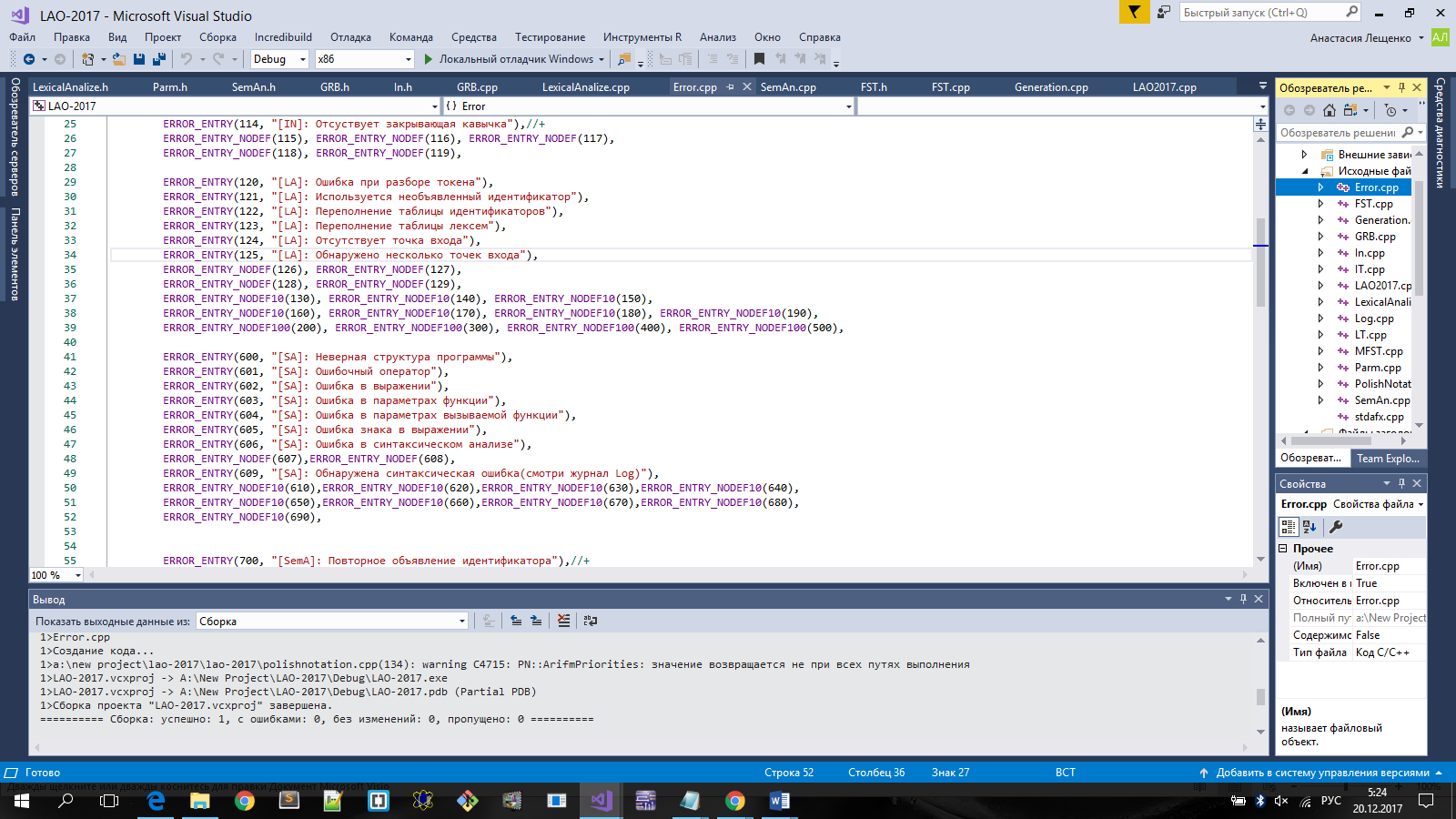
5. Если в магазине встретился нетерминал, переходим к пункту 3.

6. Если символ достиг символа дна стека, и лента в этот момент имеет символ дна стека, то синтаксический анализ выполнен успешно. Иначе генерируется ошибка.

**4.6 Структура и перечень сообщений синтаксического анализатора**

На рис. 4.3 представлен перечень сообщений синтаксического анализатора.

Рис. 4.3



Структура ошибки синтаксического анализатора: номер, префикс и текст.

**4.7 Параметры синтаксического анализатора и режимы его работы**

Входными данными для языка LAO-2017 являются таблицы лексем и идентификаторов. Выходными данными синтаксического анализатора являются дерево разбора и трассировка разбора исходного кода.

**4.8 Принцип обработки ошибок**

Изначально, если анализатор разбирает часть исходного кода по какому-либо правилу, то запоминает его. При возникновении ошибки синтаксический анализатор откатиться назад до правила, при помощи которого разбор был успешным, если это возможно. После чего пытается применить последующие правила из грамматики. В случае если правило невозможно подобрать, выводится сообщение об ошибке.

**4.9 Контрольный пример**

Распечатка протокола разбора и дерева разбора для языка LAO-2017 представлено в приложении 2.

**ГЛАВА 5. РАЗРАБОТКА СЕМАНТИЧЕСКОГО АНАЛИЗАТОРА**

* 1. **Структура семантического анализатора**

Структура семантического анализатора представлена на рис. 5.1

Рис. 5.1



Семантический анализ для языка LAO-2017 частично выполняется на фазе лексического анализа, но значительное количество проверок происходит уже непосредственно на самом этапе семантического анализа, которые рассмотрены далее.

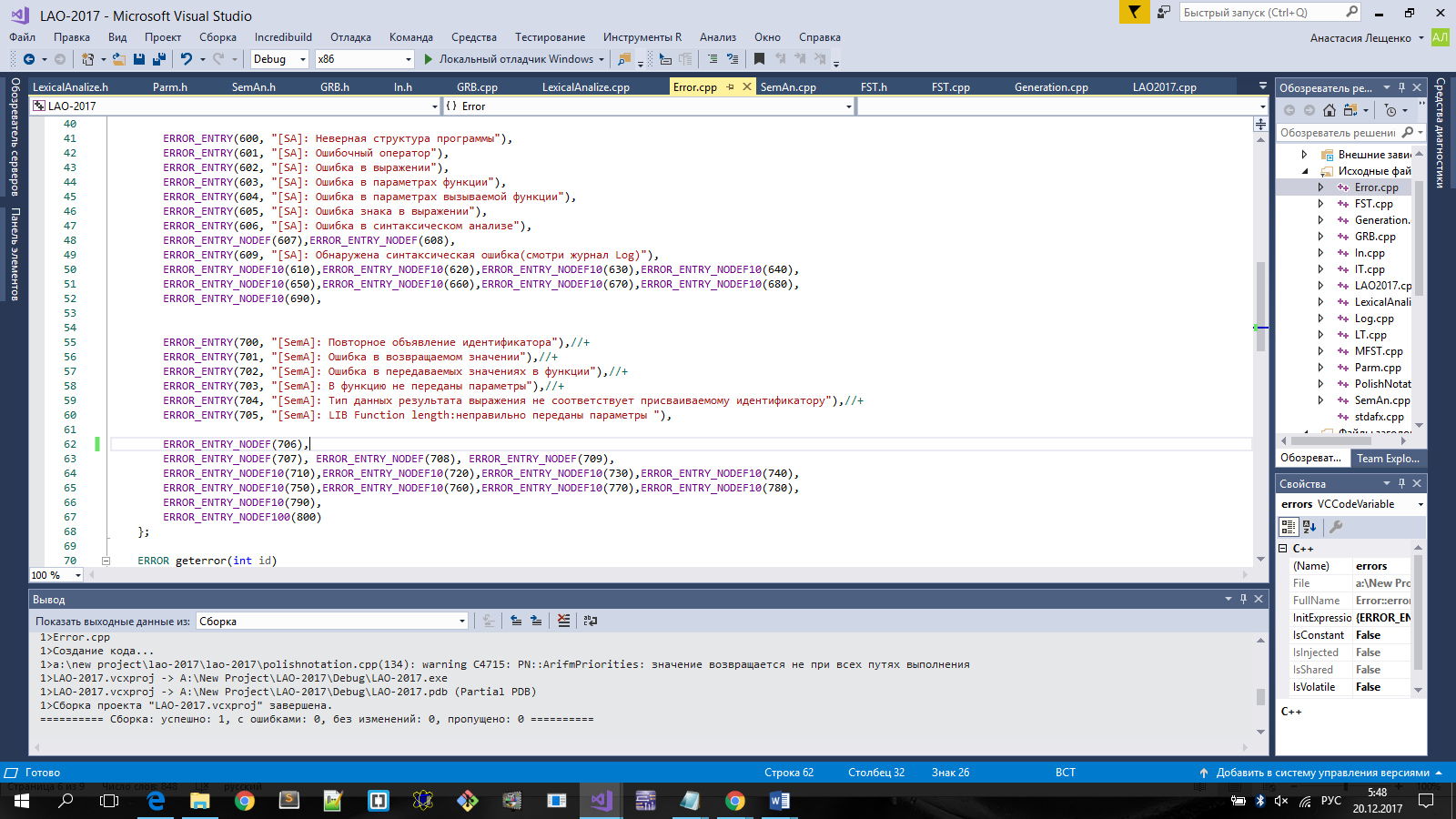
* 1. **Функции семантического анализатора**

Функции семантического анализатора аналогичны ошибкам, приведённым на рис. 5.2. То есть каждой ошибке соответствует функция, которая проверяет наличие данной ошибки в исходном коде на языке LAO-2017.

* 1. **Структура и перечень сообщений семантического анализатора**

Перечень сообщений семантического анализатора для языка LAO-2017 представлен на рис. 5.2

Рис. 5.2



Ошибка состоит из номера, префикса и теста.

* 1. **Принцип обработки ошибок**

При обнаружении хотя бы одной ошибки транслятор проведёт только семантический анализ, после чего завершит свою работу. В случае обнаружения нескольких ошибок они будут записаны в массив. По окончании семантического анализа либо достижении максимального количества ошибок в массиве они будут выведены. Максимально может быть обнаружено 10 ошибок в исходном коде.

**ГЛАВА 6. ПРЕОБРАЗОВАНИЕ ВЫРАЖЕНИЙ**

**6.1 Выражения, допускаемые языком**

В языке программирования LAO-2017 допускаются вычисление выражений со стандартными типами данных: num и string. В выражениях могут присутствовать операции, которые также описаны в пункте 1.12.

Каждая операция в выражении имеет свой приоритет. Значения приоритетов представлены в таблице 6.1.

Таблица 6.1. Приоритеты операций

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( | 1 |
| ) | 1 |
| + | 2 |
| - | 2 |
| \* | 3 |

**6.2 Польская запись и принцип ее построения**

Польская запись удобна для вычисления выражений и как промежуточная форма представления выражений в трансляторе.

Алгоритм построения:

-читаем очередной символ;

-если он является идентификатором или литералом, то добавляем его к выходной строке;

-если символ является символом функции, то помещаем его в стек;

-если символ является открывающей скобкой, то она помещается в стек;

-если символ является закрывающей скобкой, то выталкиваем из стека в выходную строку все символы пока не встретим открывающую скобку. При этом обе скобки удаляются и не попадают в выходную строку;

-как только входная лента закончится все символы из стека выталкиваются в выходную строку;

-в случае если встречаются операции, то выталкиваем из стека в выходную строку все операции, которые имеют выше приоритетность чем последняя операция;

-также, если идентификатор является именем функции, то он заменяется на спецсимвол «@».

**6.3 Программная реализация обработки выражений**

Программная реализация преобразования выражений к польской записи представлена в приложении 3.

**6.4 Контрольный пример**

Примеры преобразования выражений к польской записи представлены в таблице 6.2.

Таблица 6.2. Примеры выражений в польской записи

|  |  |
| --- | --- |
| Выражение | Выражение в польской записи |
| 2\*(3+7) | 237+\* |
| i(i,k) | ik@ |

**Глава 7. Генерация кода**

**7.1 Структура генератора кода**

Структура генератора ассемблерного кода представлена на рисунке 7.1

Рисунок 7.1



Генератор кода – часть транслятора, которая выполняет генерацию ассемблерного кода на основе полученных данных, которые были получены на предыдущих этапах трансляции.

Входными данными для генератора кода являются дерево разбора и таблица идентификаторов.

**7.2 Алгоритм генерации объектного кода**

Алгоритм генерации объектного кода выглядит следующим образом:

В первую очередь на основе таблицы идентификаторов идет заполнение поля .const литералами. Результат представлен ниже.

.const

literal0 DWORD 5

literal1 DWORD 1

literal2 BYTE "k", 0

literal3 DWORD 0

literal4 BYTE "less0", 0

literal5 BYTE "more0", 0

literal6 BYTE "first", 0

literal7 BYTE "second", 0

literal8 DWORD 2

literal9 DWORD 2

literal10 DWORD 0

Далее идет обход таблицы лексем, ищется объявление переменных. В случае если был найден литерал ‘v’, идет объявление данной переменной в поле .data.

Результат представлен ниже

.data

craz0 DWORD ?

ient DWORD ?

tent DWORD ?

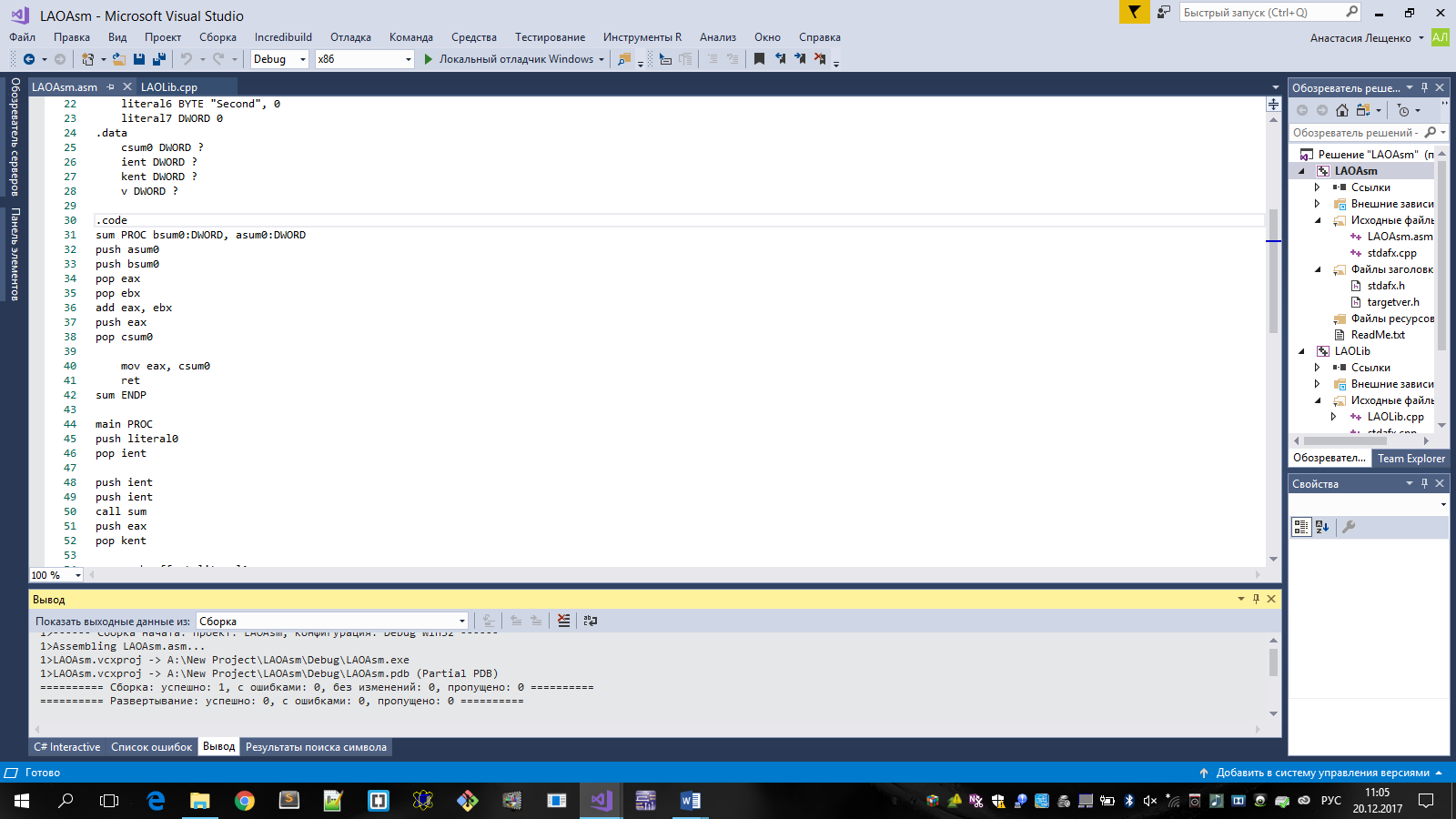
kent DWORD ?

v DWORD ?

s DWORD ?

Далее идет обход таблицы лексем с целью описать функции и конструкции языка LAO-2017.

Результат данного этапа представлен ниже



## **7.3 Алгоритм создания шаблонов выражений из польской записи**

Для того, чтобы преобразовать выражения из польской записи к виду, понятному языку ассемблера использовался следующий алгоритм:

-если встречаем идентификатор, то заталкиваем его в стек;

-если встречаем знак арифметической операции достаём один операнд в регистр eax, второй в ebx. Затем выполняем арифметическую операцию, соответствующую встреченному знаку. Результат полученный в регистре eax заталкиваем в стек;

-если встречаем символ “@”, соответствующий процедуре, — вызываем её. После выполнения процедуры результат помещается в регистр eax, поэтому заталкиваем его в стек;

-если встречаем символ “;”, достаём из стека результат вычисления выражения и помещаем в переменную, которая находилась перед символом “=”.

Пример полученного выражения представлены ниже.

Рис. 7.5

.code

razn PROC braz0:DWORD, araz0:DWORD

push araz0

push braz0

pop ebx

pop eax

sub eax, ebx

push eax

pop craz0

mov eax, craz0

ret

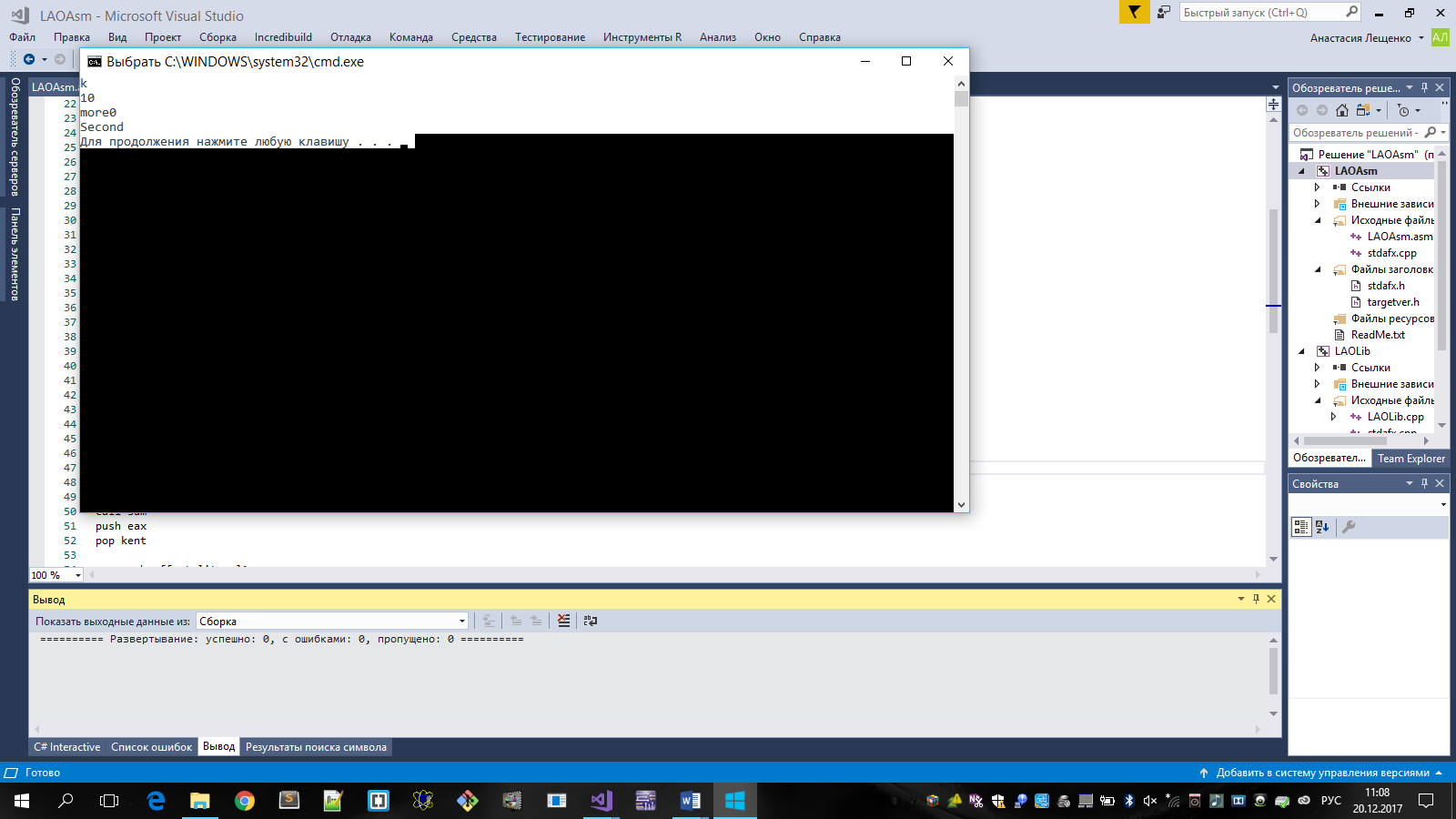
razn ENDP

## **Контрольный пример**

Контрольный пример ассемблерного кода представлен в приложении 4.

Результат генерации отображен на рисунке 7.6

Рис. 7.6



**Глава 8. Тестирование транслятора**

**8.1 Тестирование лексического анализатора**

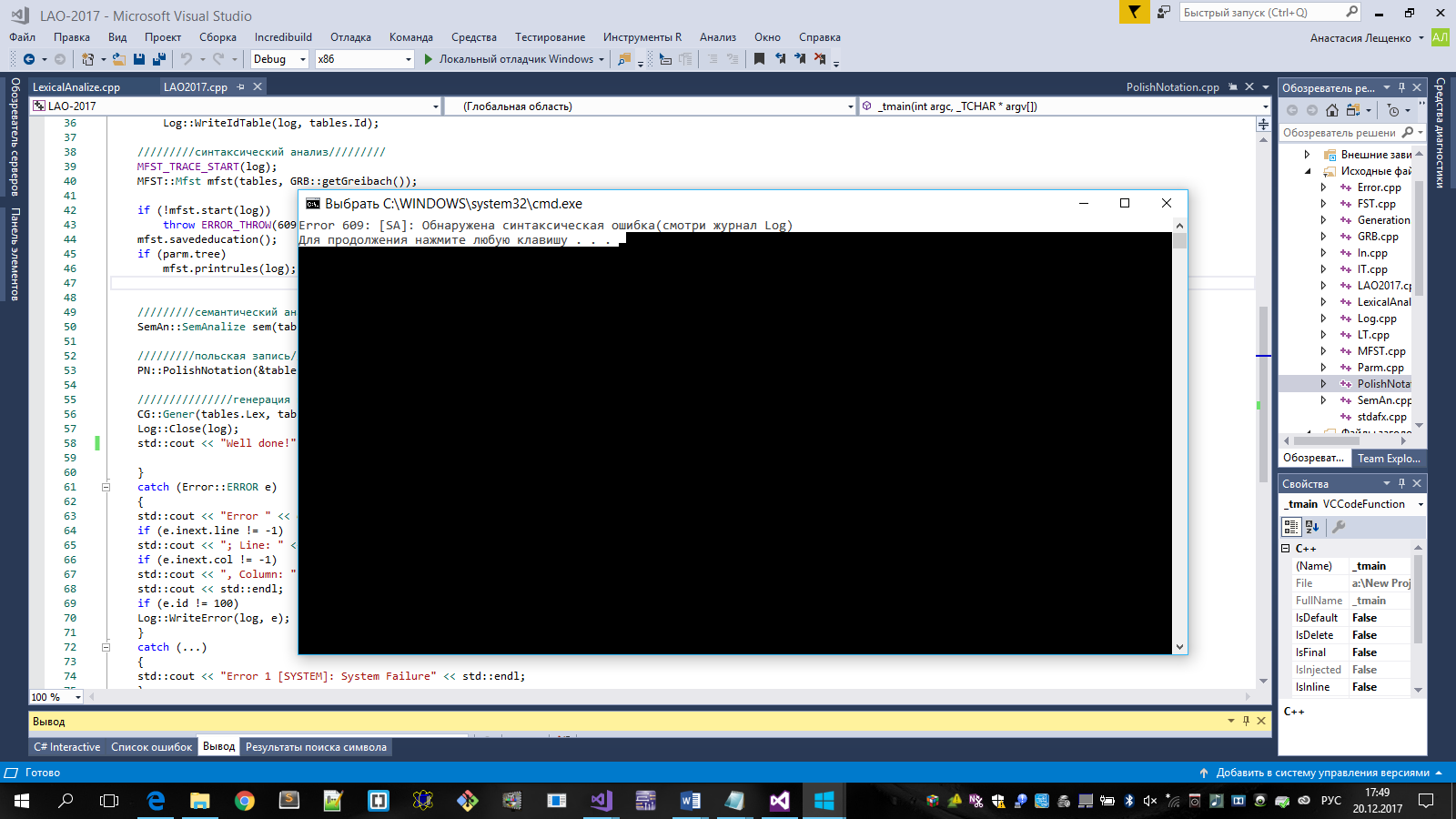
**8.2 Тестирование синтаксического анализатора**

Объявление без «var»:

var num t;

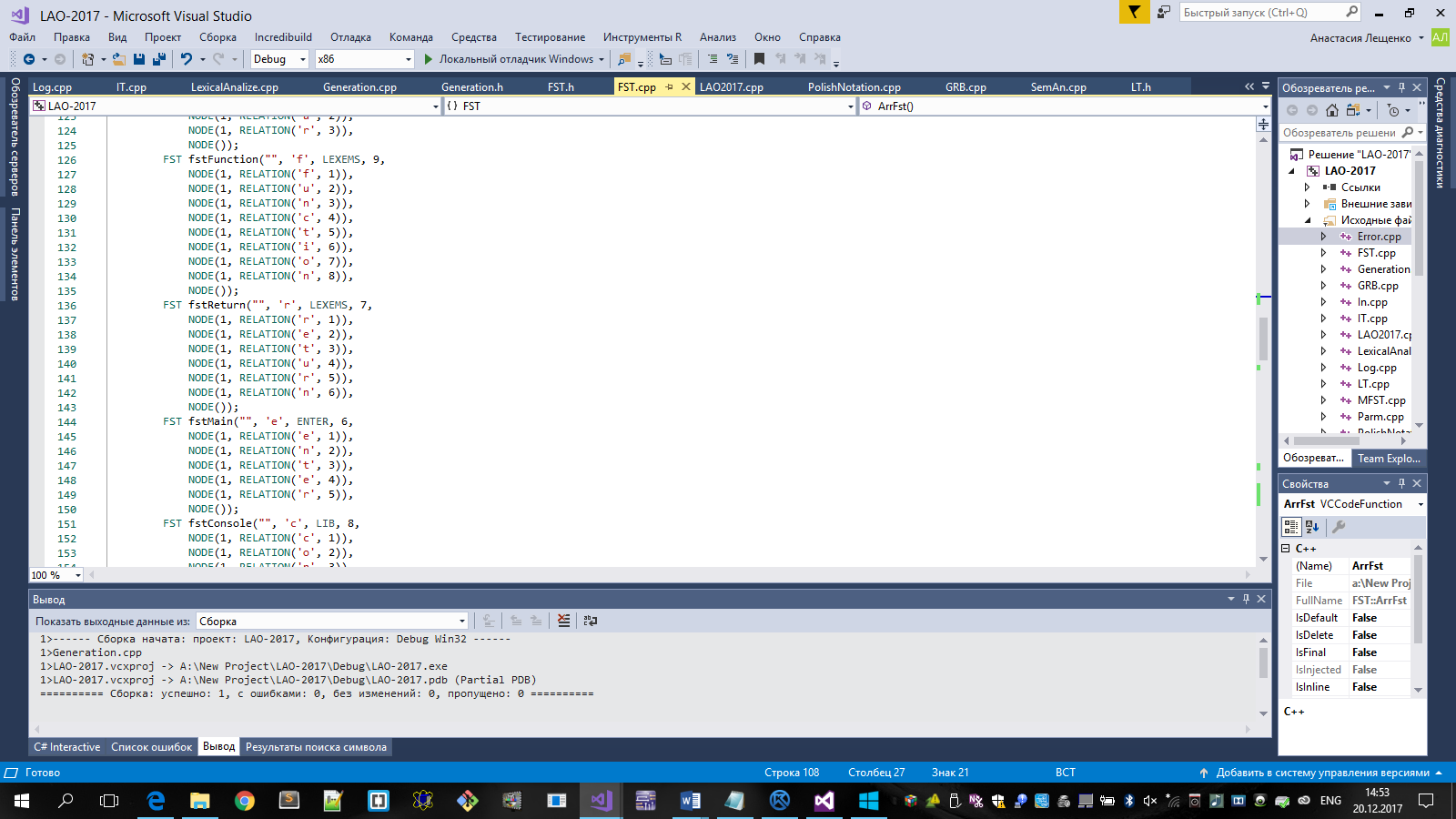
t = 1;

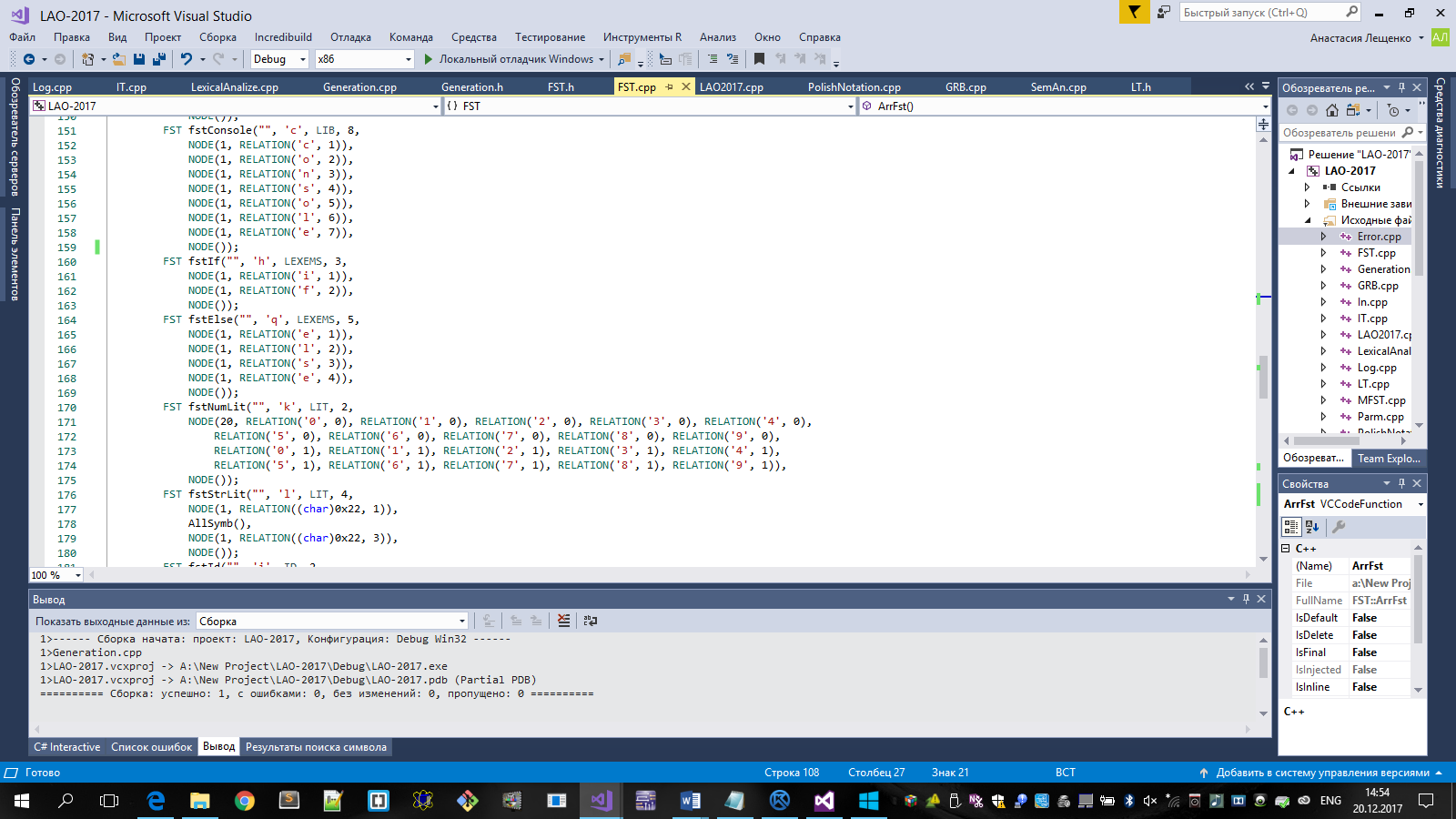
num k;

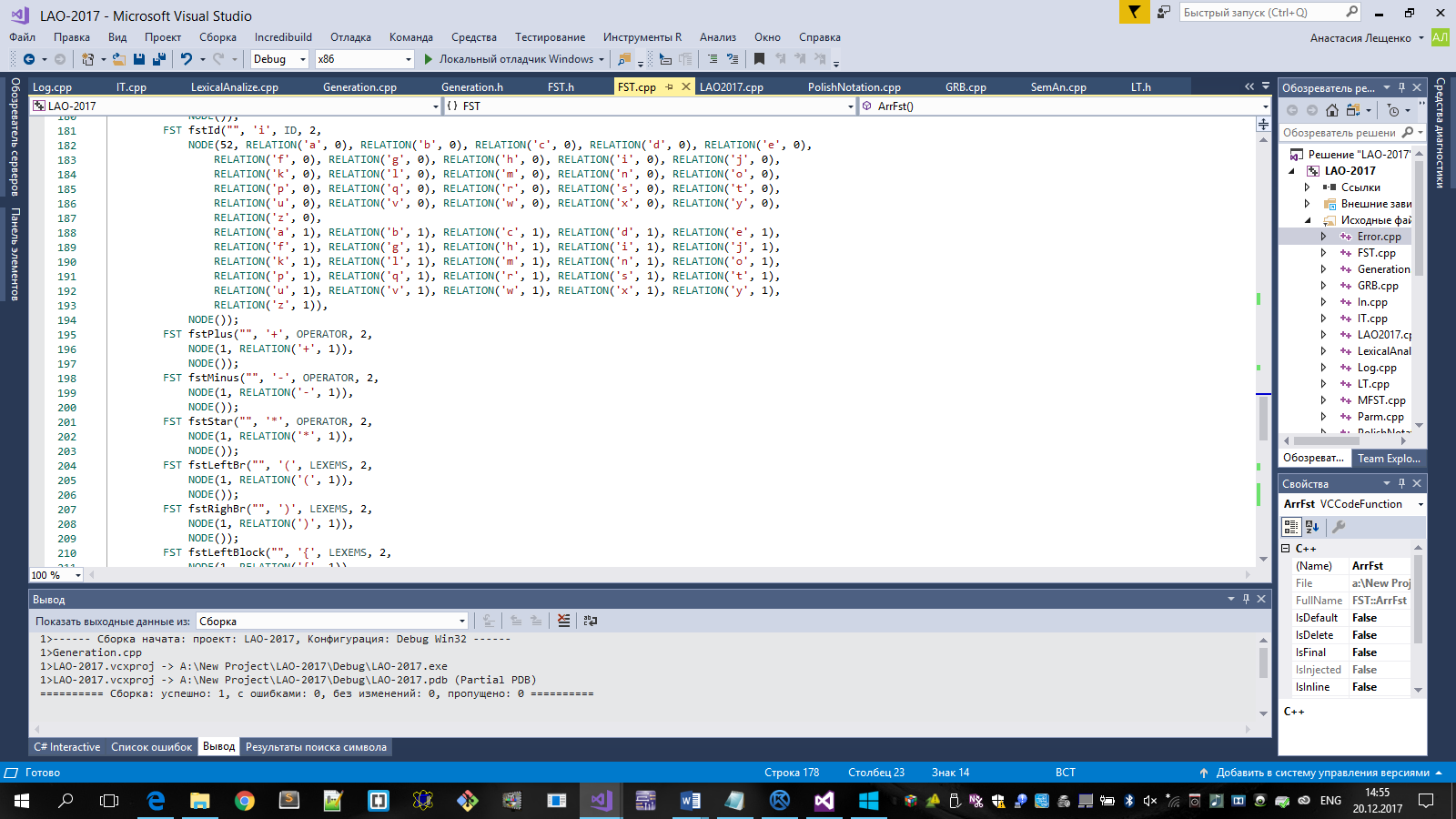


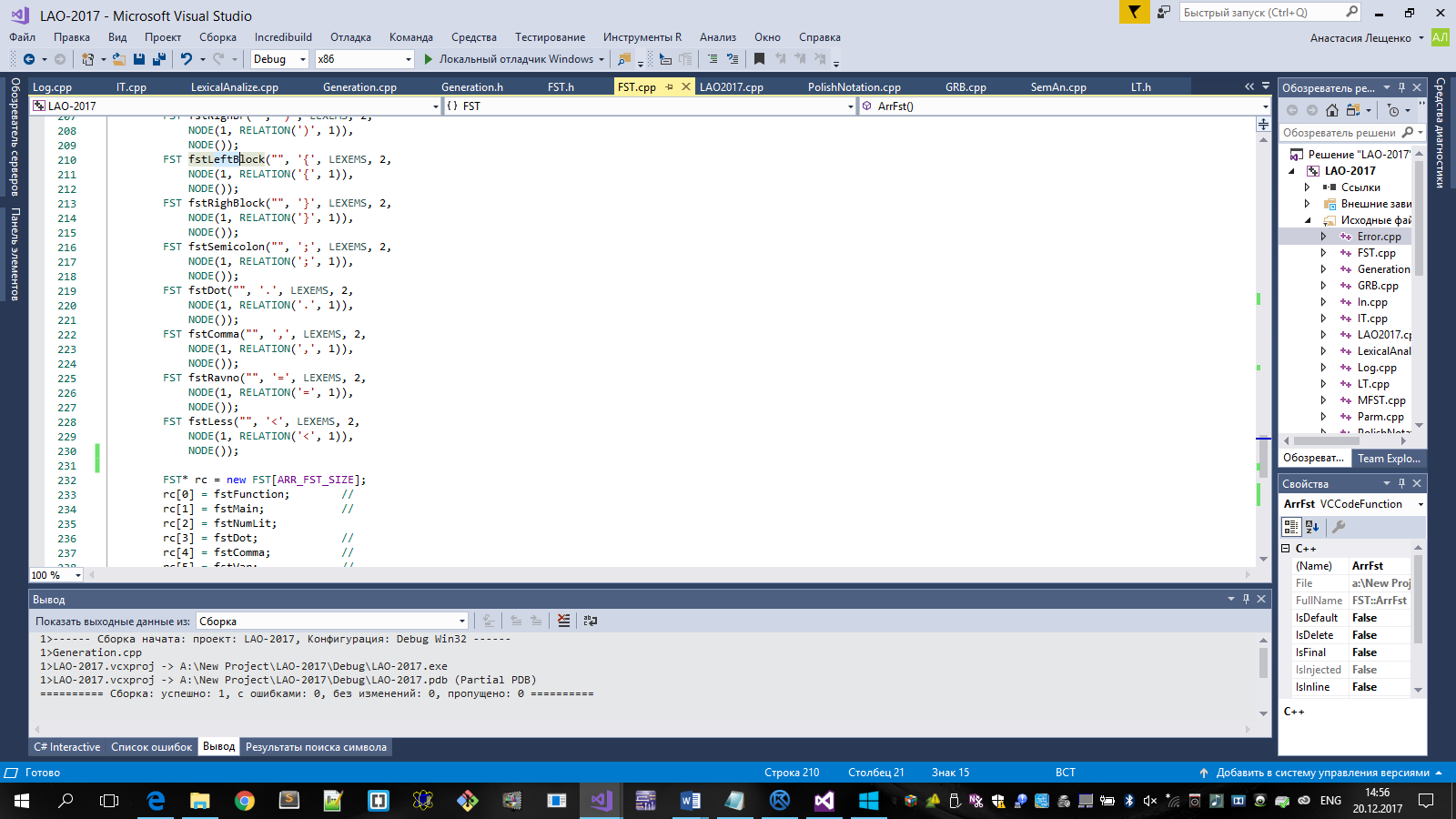
**8.3 Тестирование семантического анализатора**

**Приложение А**

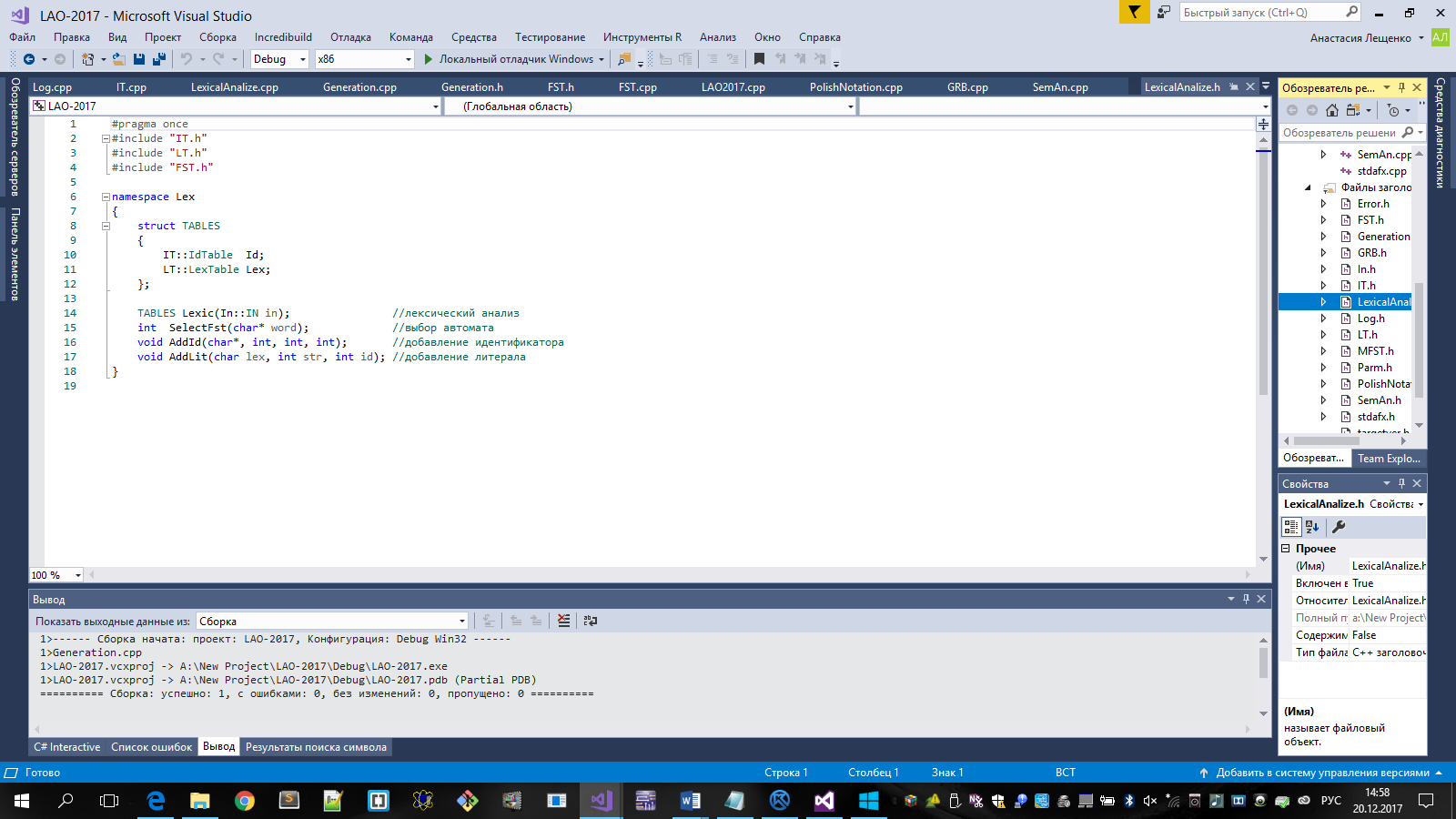
 



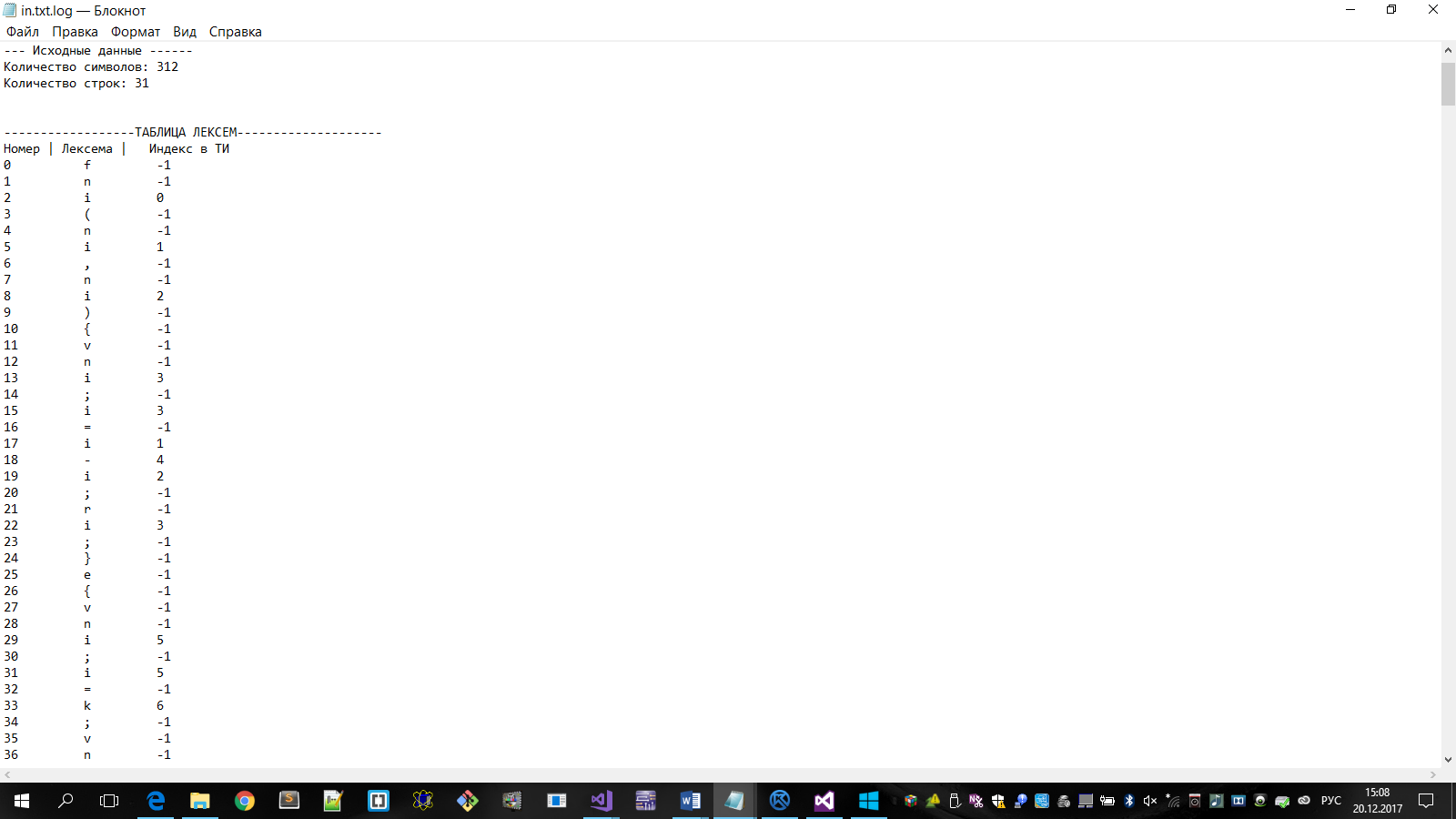
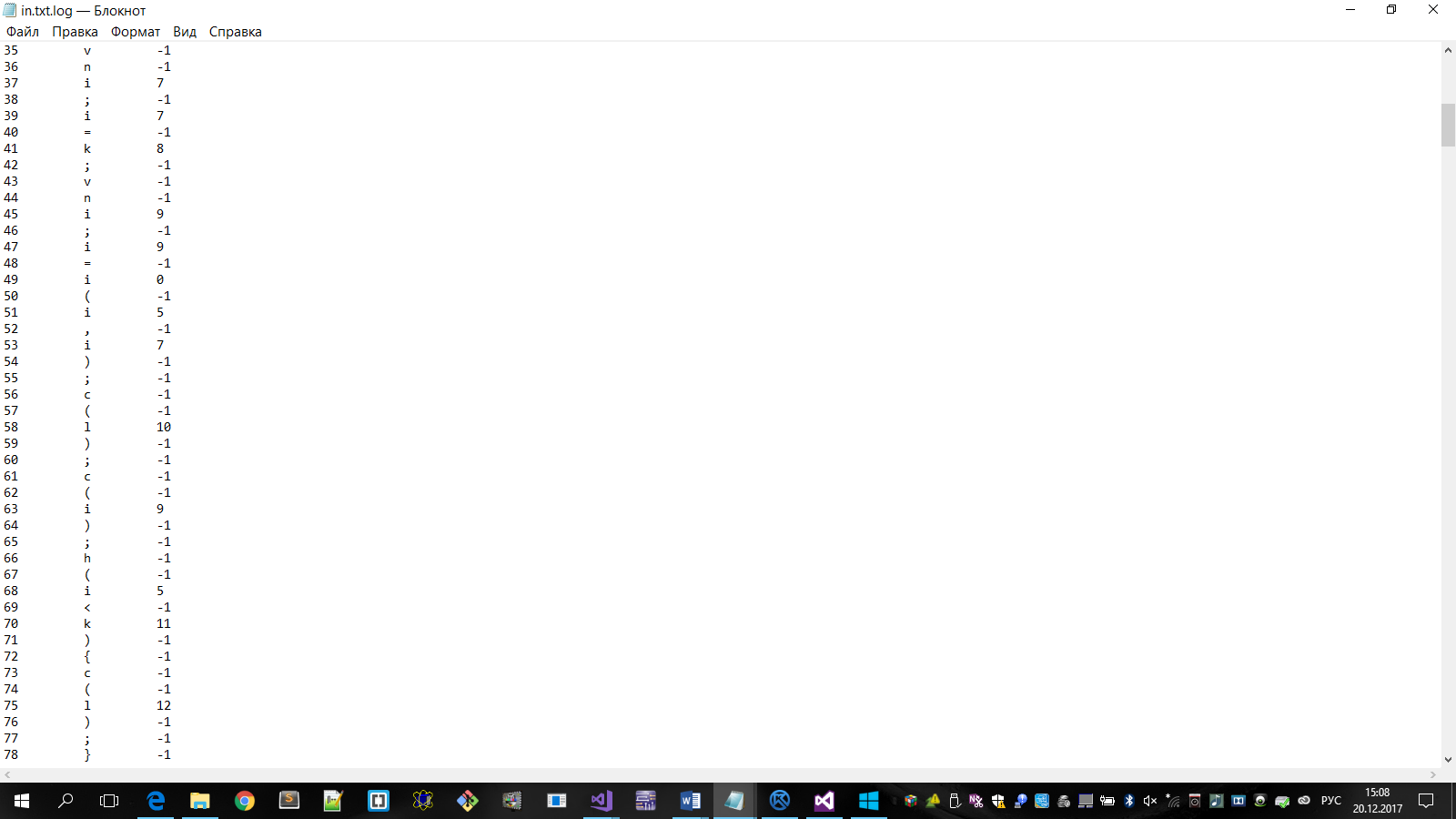
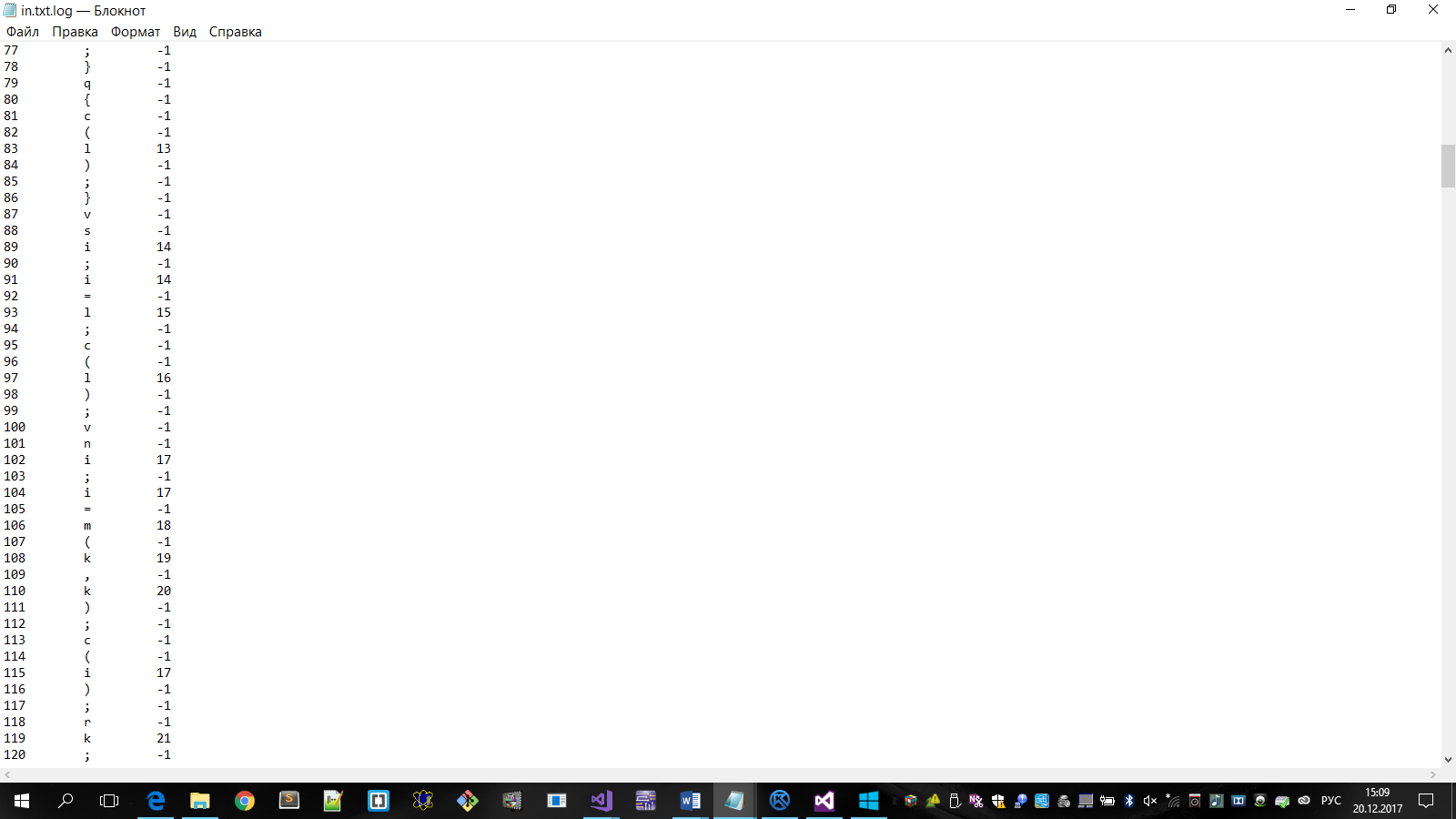




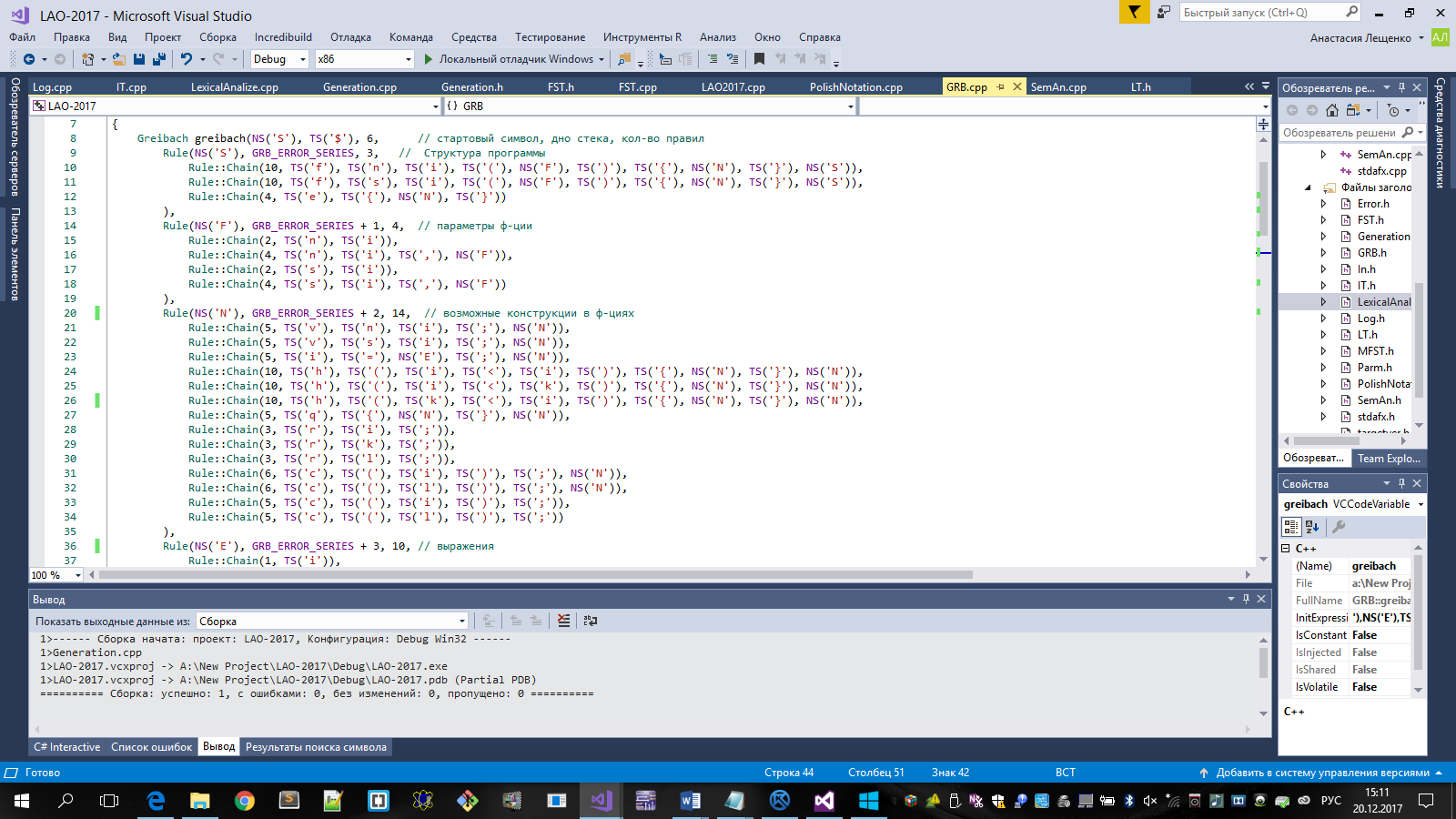
**Приложение Б**

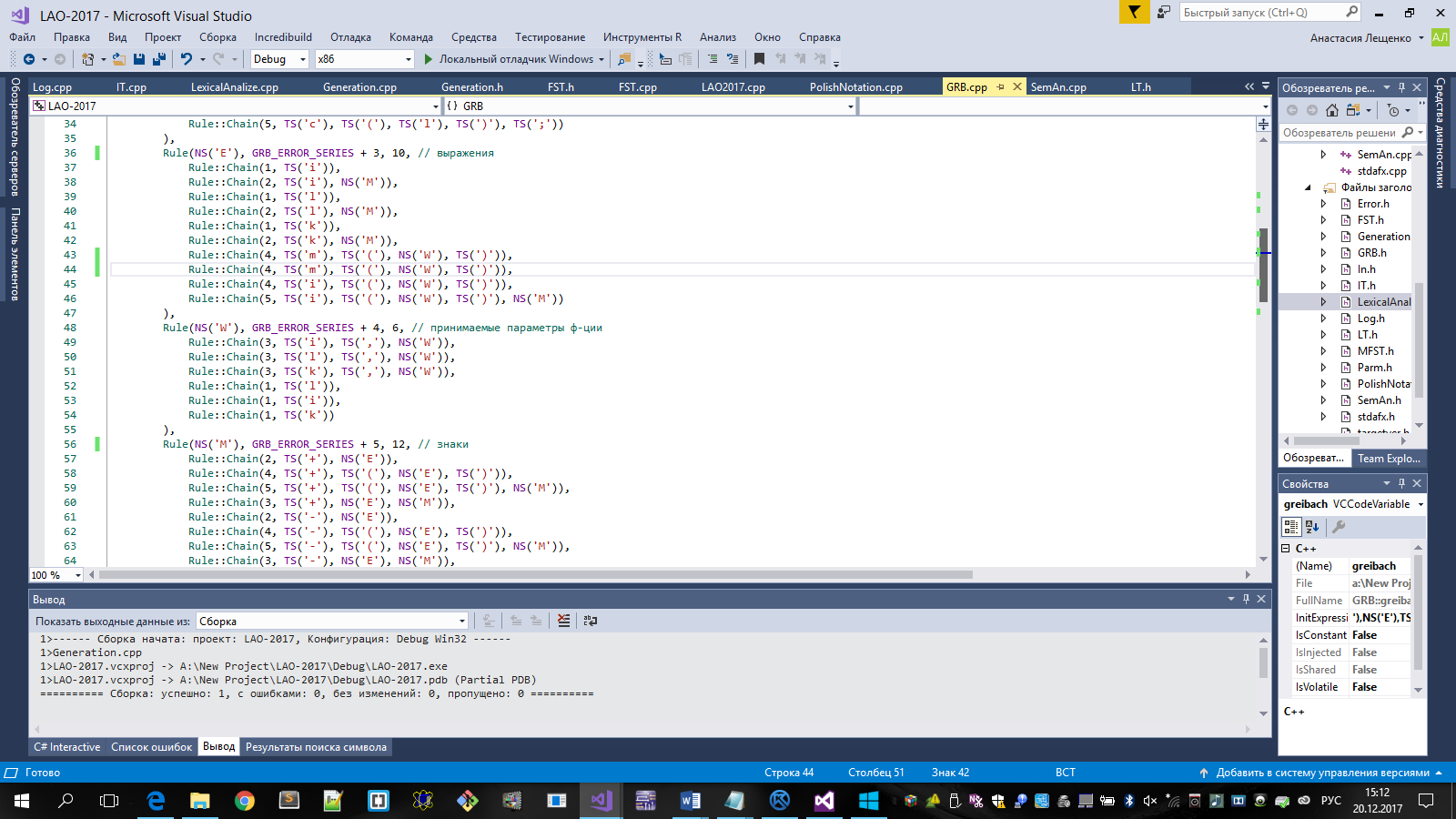


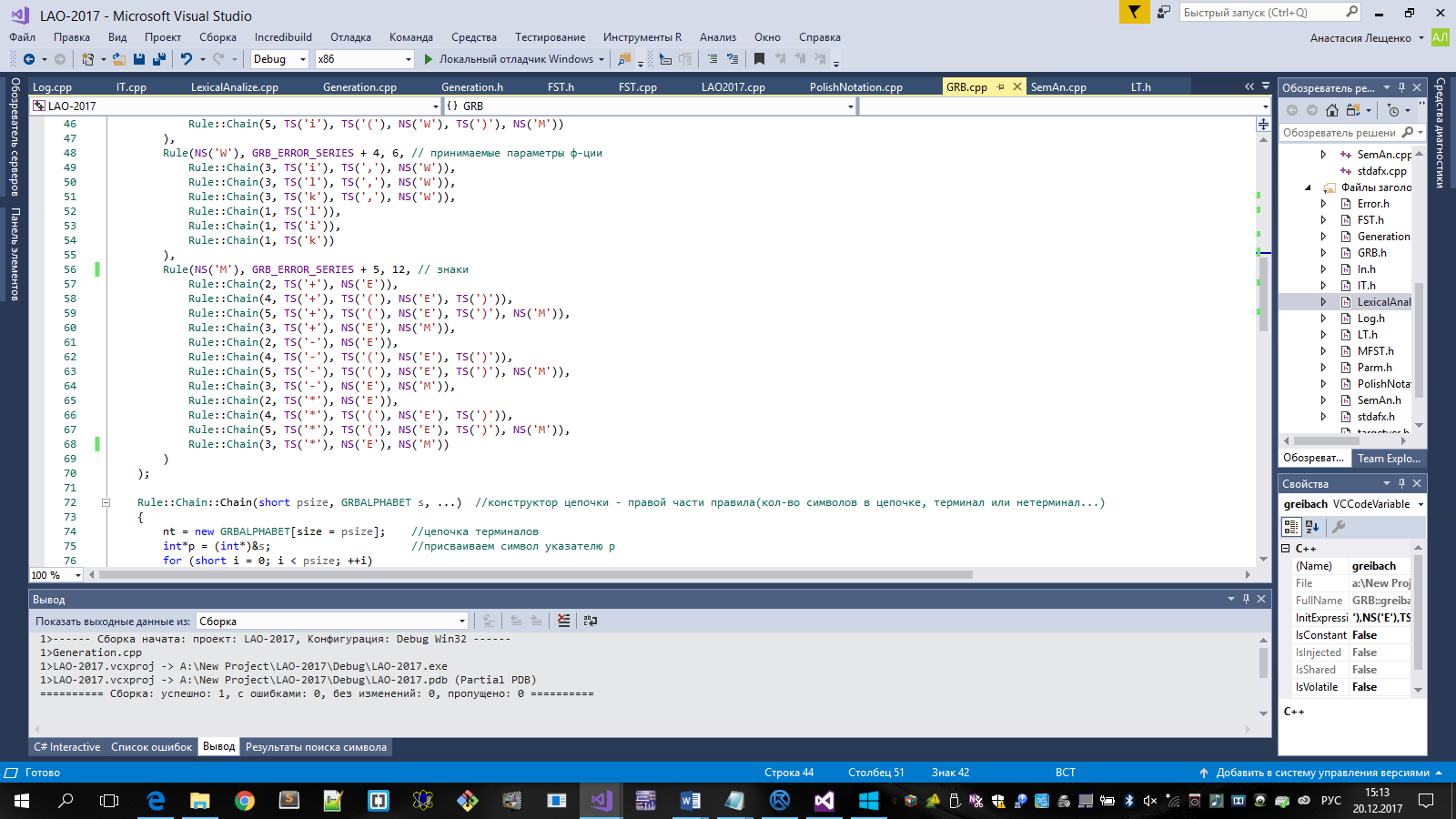
**Приложение В**

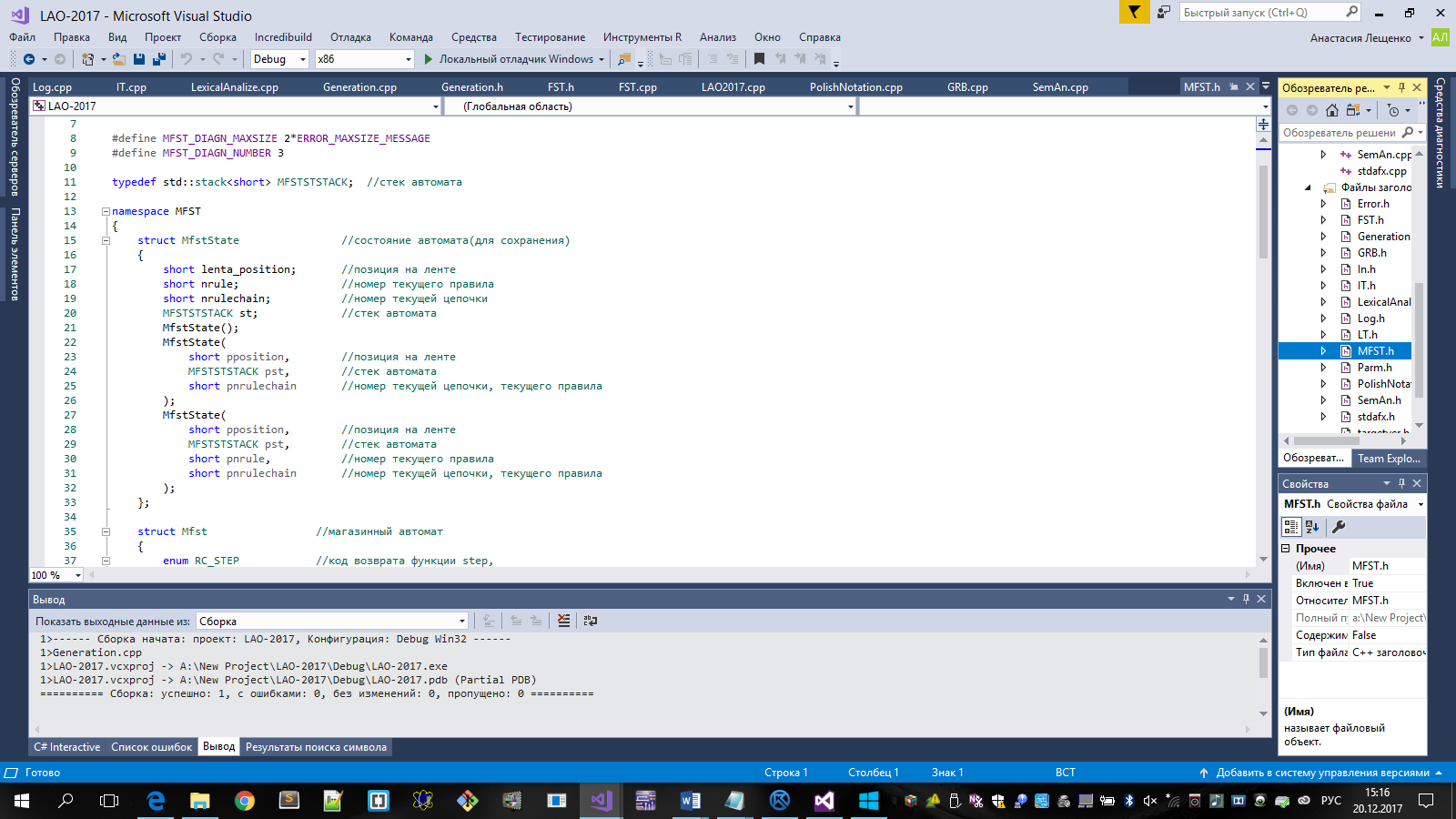
**Приложение Г**

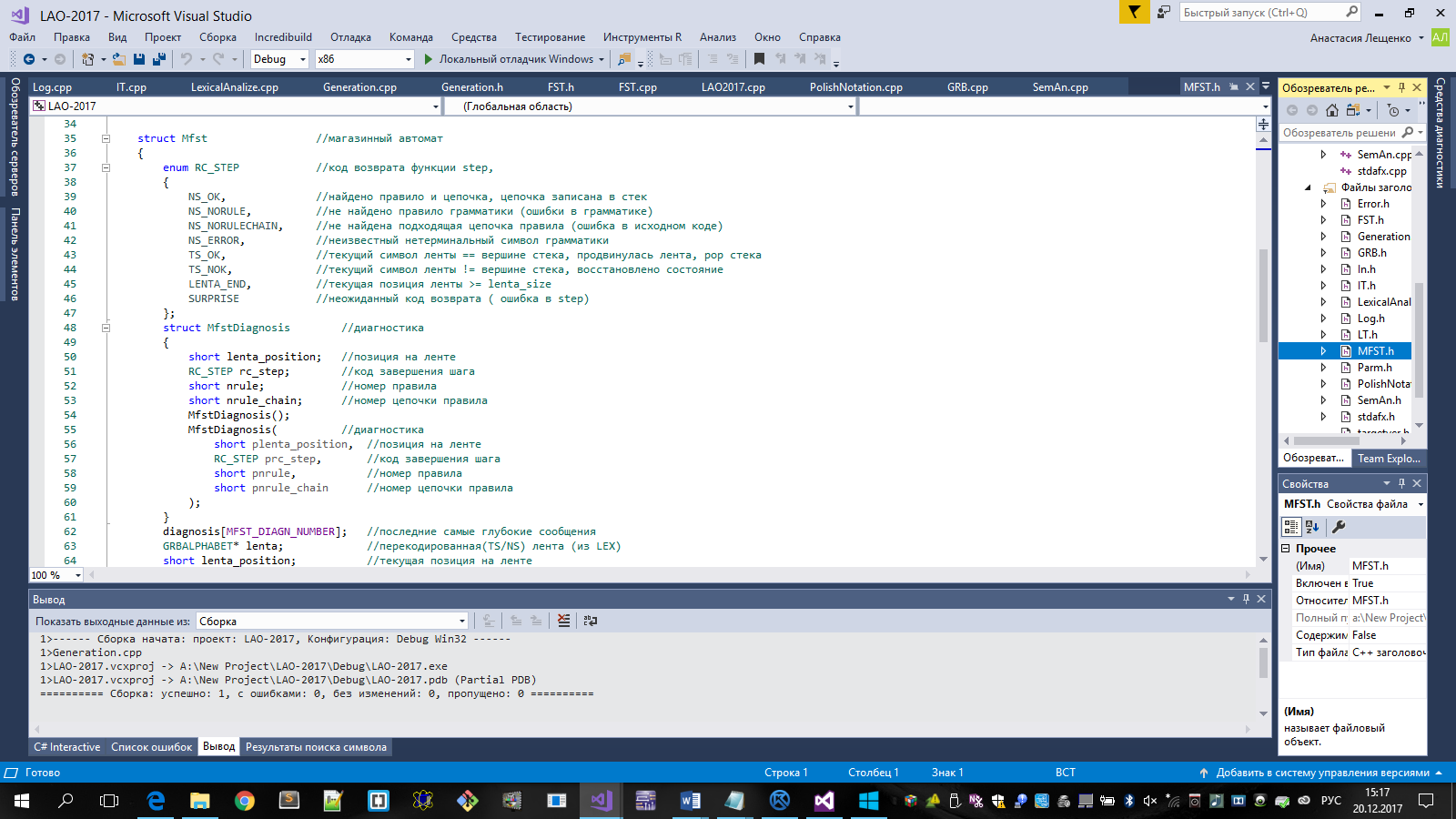


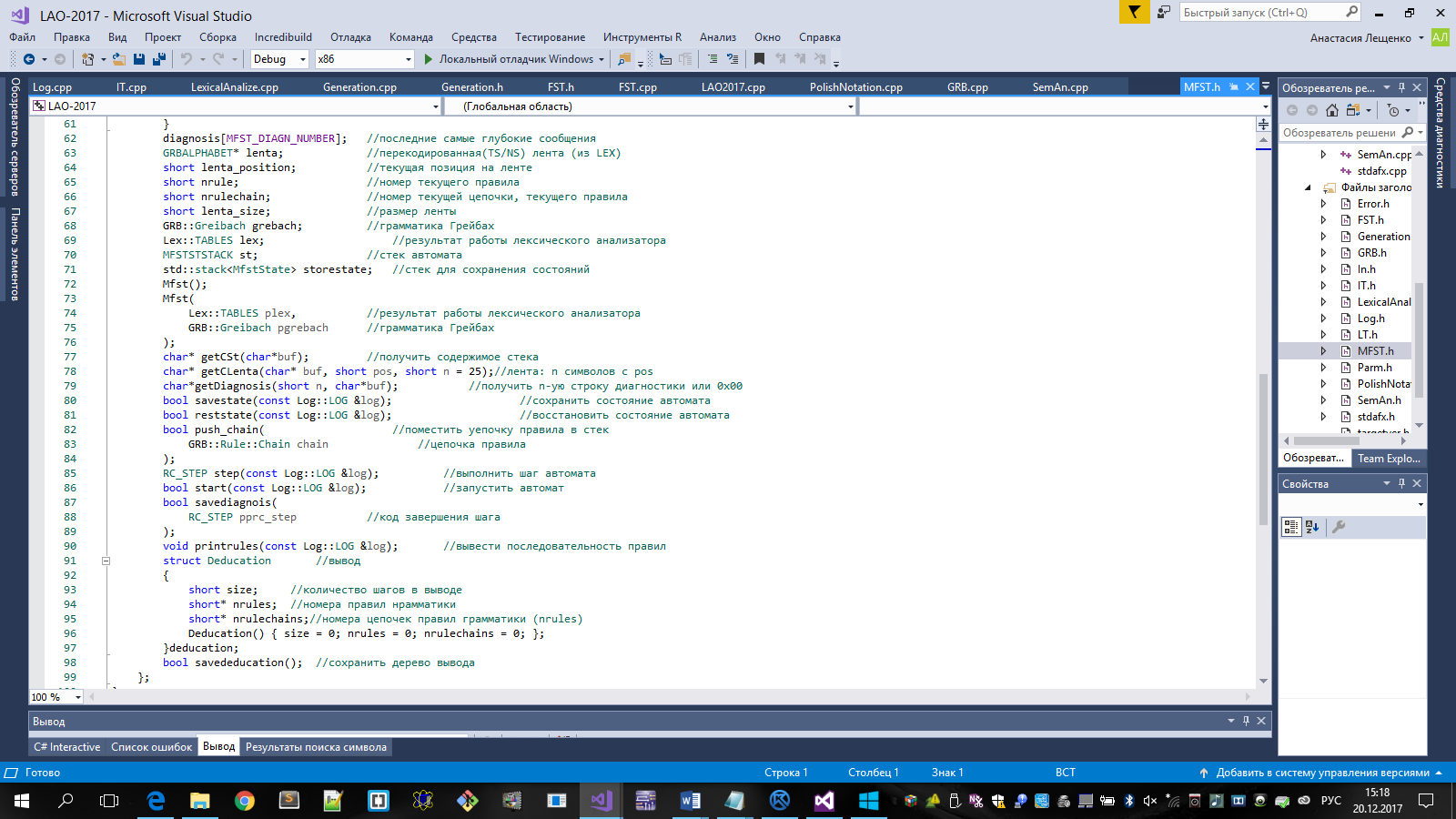




**Приложение 1**







**Приложение 2**

0 :S->fni(F){N}S fni(ni,ni){vni;i=i-i;ri;} S$

1 : SAVESTATE: 1

1 : fni(ni,ni){vni;i=i-i;ri;} fni(F){N}S$

2 : ni(ni,ni){vni;i=i-i;ri;}e ni(F){N}S$

3 : i(ni,ni){vni;i=i-i;ri;}e{ i(F){N}S$

4 : (ni,ni){vni;i=i-i;ri;}e{v (F){N}S$

5 : ni,ni){vni;i=i-i;ri;}e{vn F){N}S$

6 :F->ni ni,ni){vni;i=i-i;ri;}e{vn F){N}S$

7 : SAVESTATE: 2

7 : ni,ni){vni;i=i-i;ri;}e{vn ni){N}S$

8 : i,ni){vni;i=i-i;ri;}e{vni i){N}S$

9 : ,ni){vni;i=i-i;ri;}e{vni; ){N}S$

10 : TS\_NOK/NS\_NORULECHAIN

10 : RESSTATE

10 : ni,ni){vni;i=i-i;ri;}e{vn F){N}S$

11 :F->ni,F ni,ni){vni;i=i-i;ri;}e{vn F){N}S$

12 : SAVESTATE: 2

12 : ni,ni){vni;i=i-i;ri;}e{vn ni,F){N}S$

13 : i,ni){vni;i=i-i;ri;}e{vni i,F){N}S$

14 : ,ni){vni;i=i-i;ri;}e{vni; ,F){N}S$

15 : ni){vni;i=i-i;ri;}e{vni;i F){N}S$

16 :F->ni ni){vni;i=i-i;ri;}e{vni;i F){N}S$

17 : SAVESTATE: 3

17 : ni){vni;i=i-i;ri;}e{vni;i ni){N}S$

18 : i){vni;i=i-i;ri;}e{vni;i= i){N}S$

19 : ){vni;i=i-i;ri;}e{vni;i=k ){N}S$

20 : {vni;i=i-i;ri;}e{vni;i=k; {N}S$

21 : vni;i=i-i;ri;}e{vni;i=k;v N}S$

22 :N->vni;N vni;i=i-i;ri;}e{vni;i=k;v N}S$

23 : SAVESTATE: 4

23 : vni;i=i-i;ri;}e{vni;i=k;v vni;N}S$

24 : ni;i=i-i;ri;}e{vni;i=k;vn ni;N}S$

25 : i;i=i-i;ri;}e{vni;i=k;vni i;N}S$

26 : ;i=i-i;ri;}e{vni;i=k;vni; ;N}S$

27 : i=i-i;ri;}e{vni;i=k;vni;i N}S$

28 :N->i=E;N i=i-i;ri;}e{vni;i=k;vni;i N}S$

29 : SAVESTATE: 5

29 : i=i-i;ri;}e{vni;i=k;vni;i i=E;N}S$

30 : =i-i;ri;}e{vni;i=k;vni;i= =E;N}S$

31 : i-i;ri;}e{vni;i=k;vni;i=k E;N}S$

32 :E->i i-i;ri;}e{vni;i=k;vni;i=k E;N}S$

33 : SAVESTATE: 6

33 : i-i;ri;}e{vni;i=k;vni;i=k i;N}S$

34 : -i;ri;}e{vni;i=k;vni;i=k; ;N}S$

35 : TS\_NOK/NS\_NORULECHAIN

35 : RESSTATE

35 : i-i;ri;}e{vni;i=k;vni;i=k E;N}S$

36 :E->iM i-i;ri;}e{vni;i=k;vni;i=k E;N}S$

37 : SAVESTATE: 6

37 : i-i;ri;}e{vni;i=k;vni;i=k iM;N}S$

38 : -i;ri;}e{vni;i=k;vni;i=k; M;N}S$

39 :M->-E -i;ri;}e{vni;i=k;vni;i=k; M;N}S$

40 : SAVESTATE: 7

40 : -i;ri;}e{vni;i=k;vni;i=k; -E;N}S$

41 : i;ri;}e{vni;i=k;vni;i=k;v E;N}S$

42 :E->i i;ri;}e{vni;i=k;vni;i=k;v E;N}S$

43 : SAVESTATE: 8

43 : i;ri;}e{vni;i=k;vni;i=k;v i;N}S$

44 : ;ri;}e{vni;i=k;vni;i=k;vn ;N}S$

45 : ri;}e{vni;i=k;vni;i=k;vni N}S$

46 :N->ri; ri;}e{vni;i=k;vni;i=k;vni N}S$

47 : SAVESTATE: 9

47 : ri;}e{vni;i=k;vni;i=k;vni ri;}S$

48 : i;}e{vni;i=k;vni;i=k;vni; i;}S$

49 : ;}e{vni;i=k;vni;i=k;vni;i ;}S$

50 : }e{vni;i=k;vni;i=k;vni;i= }S$

51 : e{vni;i=k;vni;i=k;vni;i=i S$

52 :S->e{N} e{vni;i=k;vni;i=k;vni;i=i S$

53 : SAVESTATE: 10

53 : e{vni;i=k;vni;i=k;vni;i=i e{N}$

54 : {vni;i=k;vni;i=k;vni;i=i( {N}$

55 : vni;i=k;vni;i=k;vni;i=i(i N}$

56 :N->vni;N vni;i=k;vni;i=k;vni;i=i(i N}$

57 : SAVESTATE: 11

57 : vni;i=k;vni;i=k;vni;i=i(i vni;N}$

58 : ni;i=k;vni;i=k;vni;i=i(i, ni;N}$

59 : i;i=k;vni;i=k;vni;i=i(i,i i;N}$

60 : ;i=k;vni;i=k;vni;i=i(i,i) ;N}$

61 : i=k;vni;i=k;vni;i=i(i,i); N}$

62 :N->i=E;N i=k;vni;i=k;vni;i=i(i,i); N}$

63 : SAVESTATE: 12

63 : i=k;vni;i=k;vni;i=i(i,i); i=E;N}$

64 : =k;vni;i=k;vni;i=i(i,i);c =E;N}$

65 : k;vni;i=k;vni;i=i(i,i);c( E;N}$

66 :E->k k;vni;i=k;vni;i=i(i,i);c( E;N}$

67 : SAVESTATE: 13

67 : k;vni;i=k;vni;i=i(i,i);c( k;N}$

68 : ;vni;i=k;vni;i=i(i,i);c(l ;N}$

69 : vni;i=k;vni;i=i(i,i);c(l) N}$

70 :N->vni;N vni;i=k;vni;i=i(i,i);c(l) N}$

71 : SAVESTATE: 14

71 : vni;i=k;vni;i=i(i,i);c(l) vni;N}$

72 : ni;i=k;vni;i=i(i,i);c(l); ni;N}$

73 : i;i=k;vni;i=i(i,i);c(l);c i;N}$

74 : ;i=k;vni;i=i(i,i);c(l);c( ;N}$

75 : i=k;vni;i=i(i,i);c(l);c(i N}$

76 :N->i=E;N i=k;vni;i=i(i,i);c(l);c(i N}$

77 : SAVESTATE: 15

77 : i=k;vni;i=i(i,i);c(l);c(i i=E;N}$

78 : =k;vni;i=i(i,i);c(l);c(i) =E;N}$

79 : k;vni;i=i(i,i);c(l);c(i); E;N}$

80 :E->k k;vni;i=i(i,i);c(l);c(i); E;N}$

81 : SAVESTATE: 16

81 : k;vni;i=i(i,i);c(l);c(i); k;N}$

82 : ;vni;i=i(i,i);c(l);c(i);h ;N}$

83 : vni;i=i(i,i);c(l);c(i);h( N}$

84 :N->vni;N vni;i=i(i,i);c(l);c(i);h( N}$

85 : SAVESTATE: 17

85 : vni;i=i(i,i);c(l);c(i);h( vni;N}$

86 : ni;i=i(i,i);c(l);c(i);h(i ni;N}$

87 : i;i=i(i,i);c(l);c(i);h(i< i;N}$

88 : ;i=i(i,i);c(l);c(i);h(i<k ;N}$

89 : i=i(i,i);c(l);c(i);h(i<k) N}$

90 :N->i=E;N i=i(i,i);c(l);c(i);h(i<k) N}$

91 : SAVESTATE: 18

91 : i=i(i,i);c(l);c(i);h(i<k) i=E;N}$

92 : =i(i,i);c(l);c(i);h(i<k){ =E;N}$

93 : i(i,i);c(l);c(i);h(i<k){c E;N}$

94 :E->i i(i,i);c(l);c(i);h(i<k){c E;N}$

95 : SAVESTATE: 19

95 : i(i,i);c(l);c(i);h(i<k){c i;N}$

96 : (i,i);c(l);c(i);h(i<k){c( ;N}$

97 : TS\_NOK/NS\_NORULECHAIN

97 : RESSTATE

97 : i(i,i);c(l);c(i);h(i<k){c E;N}$

98 :E->iM i(i,i);c(l);c(i);h(i<k){c E;N}$

99 : SAVESTATE: 19

99 : i(i,i);c(l);c(i);h(i<k){c iM;N}$

100 : (i,i);c(l);c(i);h(i<k){c( M;N}$

101 : TNS\_NORULECHAIN/NS\_NORULE

101 : RESSTATE

101 : i(i,i);c(l);c(i);h(i<k){c E;N}$

102 :E->i(W) i(i,i);c(l);c(i);h(i<k){c E;N}$

103 : SAVESTATE: 19

103 : i(i,i);c(l);c(i);h(i<k){c i(W);N}$

104 : (i,i);c(l);c(i);h(i<k){c( (W);N}$

105 : i,i);c(l);c(i);h(i<k){c(l W);N}$

106 :W->i,W i,i);c(l);c(i);h(i<k){c(l W);N}$

107 : SAVESTATE: 20

107 : i,i);c(l);c(i);h(i<k){c(l i,W);N}$

108 : ,i);c(l);c(i);h(i<k){c(l) ,W);N}$

109 : i);c(l);c(i);h(i<k){c(l); W);N}$

110 :W->i,W i);c(l);c(i);h(i<k){c(l); W);N}$

111 : SAVESTATE: 21

111 : i);c(l);c(i);h(i<k){c(l); i,W);N}$

112 : );c(l);c(i);h(i<k){c(l);} ,W);N}$

113 : TS\_NOK/NS\_NORULECHAIN

113 : RESSTATE

113 : i);c(l);c(i);h(i<k){c(l); W);N}$

114 :W->i i);c(l);c(i);h(i<k){c(l); W);N}$

115 : SAVESTATE: 21

115 : i);c(l);c(i);h(i<k){c(l); i);N}$

116 : );c(l);c(i);h(i<k){c(l);} );N}$

117 : ;c(l);c(i);h(i<k){c(l);}q ;N}$

118 : c(l);c(i);h(i<k){c(l);}q{ N}$

119 :N->c(i);N c(l);c(i);h(i<k){c(l);}q{ N}$

120 : SAVESTATE: 22

120 : c(l);c(i);h(i<k){c(l);}q{ c(i);N}$

121 : (l);c(i);h(i<k){c(l);}q{c (i);N}$

122 : l);c(i);h(i<k){c(l);}q{c( i);N}$

123 : TS\_NOK/NS\_NORULECHAIN

123 : RESSTATE

123 : c(l);c(i);h(i<k){c(l);}q{ N}$

124 :N->c(l);N c(l);c(i);h(i<k){c(l);}q{ N}$

125 : SAVESTATE: 22

125 : c(l);c(i);h(i<k){c(l);}q{ c(l);N}$

126 : (l);c(i);h(i<k){c(l);}q{c (l);N}$

127 : l);c(i);h(i<k){c(l);}q{c( l);N}$

128 : );c(i);h(i<k){c(l);}q{c(l );N}$

129 : ;c(i);h(i<k){c(l);}q{c(l) ;N}$

130 : c(i);h(i<k){c(l);}q{c(l); N}$

131 :N->c(i);N c(i);h(i<k){c(l);}q{c(l); N}$

132 : SAVESTATE: 23

132 : c(i);h(i<k){c(l);}q{c(l); c(i);N}$

133 : (i);h(i<k){c(l);}q{c(l);} (i);N}$

134 : i);h(i<k){c(l);}q{c(l);}v i);N}$

135 : );h(i<k){c(l);}q{c(l);}vs );N}$

136 : ;h(i<k){c(l);}q{c(l);}vsi ;N}$

137 : h(i<k){c(l);}q{c(l);}vsi; N}$

138 :N->h(i<i){N}N h(i<k){c(l);}q{c(l);}vsi; N}$

139 : SAVESTATE: 24

139 : h(i<k){c(l);}q{c(l);}vsi; h(i<i){N}N}$

140 : (i<k){c(l);}q{c(l);}vsi;i (i<i){N}N}$

141 : i<k){c(l);}q{c(l);}vsi;i= i<i){N}N}$

142 : <k){c(l);}q{c(l);}vsi;i=l <i){N}N}$

143 : k){c(l);}q{c(l);}vsi;i=l; i){N}N}$

144 : TS\_NOK/NS\_NORULECHAIN

144 : RESSTATE

144 : h(i<k){c(l);}q{c(l);}vsi; N}$

145 :N->h(i<k){N}N h(i<k){c(l);}q{c(l);}vsi; N}$

146 : SAVESTATE: 24

146 : h(i<k){c(l);}q{c(l);}vsi; h(i<k){N}N}$

147 : (i<k){c(l);}q{c(l);}vsi;i (i<k){N}N}$

148 : i<k){c(l);}q{c(l);}vsi;i= i<k){N}N}$

149 : <k){c(l);}q{c(l);}vsi;i=l <k){N}N}$

150 : k){c(l);}q{c(l);}vsi;i=l; k){N}N}$

151 : ){c(l);}q{c(l);}vsi;i=l;c ){N}N}$

152 : {c(l);}q{c(l);}vsi;i=l;c( {N}N}$

153 : c(l);}q{c(l);}vsi;i=l;c(l N}N}$

154 :N->c(i);N c(l);}q{c(l);}vsi;i=l;c(l N}N}$

155 : SAVESTATE: 25

155 : c(l);}q{c(l);}vsi;i=l;c(l c(i);N}N}$

156 : (l);}q{c(l);}vsi;i=l;c(l) (i);N}N}$

157 : l);}q{c(l);}vsi;i=l;c(l); i);N}N}$

158 : TS\_NOK/NS\_NORULECHAIN

158 : RESSTATE

158 : c(l);}q{c(l);}vsi;i=l;c(l N}N}$

159 :N->c(l);N c(l);}q{c(l);}vsi;i=l;c(l N}N}$

160 : SAVESTATE: 25

160 : c(l);}q{c(l);}vsi;i=l;c(l c(l);N}N}$

161 : (l);}q{c(l);}vsi;i=l;c(l) (l);N}N}$

162 : l);}q{c(l);}vsi;i=l;c(l); l);N}N}$

163 : );}q{c(l);}vsi;i=l;c(l);v );N}N}$

164 : ;}q{c(l);}vsi;i=l;c(l);vn ;N}N}$

165 : }q{c(l);}vsi;i=l;c(l);vni N}N}$

166 : TNS\_NORULECHAIN/NS\_NORULE

166 : RESSTATE

166 : c(l);}q{c(l);}vsi;i=l;c(l N}N}$

167 :N->c(i); c(l);}q{c(l);}vsi;i=l;c(l N}N}$

168 : SAVESTATE: 25

168 : c(l);}q{c(l);}vsi;i=l;c(l c(i);}N}$

169 : (l);}q{c(l);}vsi;i=l;c(l) (i);}N}$

170 : l);}q{c(l);}vsi;i=l;c(l); i);}N}$

171 : TS\_NOK/NS\_NORULECHAIN

171 : RESSTATE

171 : c(l);}q{c(l);}vsi;i=l;c(l N}N}$

172 :N->c(l); c(l);}q{c(l);}vsi;i=l;c(l N}N}$

173 : SAVESTATE: 25

173 : c(l);}q{c(l);}vsi;i=l;c(l c(l);}N}$

174 : (l);}q{c(l);}vsi;i=l;c(l) (l);}N}$

175 : l);}q{c(l);}vsi;i=l;c(l); l);}N}$

176 : );}q{c(l);}vsi;i=l;c(l);v );}N}$

177 : ;}q{c(l);}vsi;i=l;c(l);vn ;}N}$

178 : }q{c(l);}vsi;i=l;c(l);vni }N}$

179 : q{c(l);}vsi;i=l;c(l);vni; N}$

180 :N->q{N}N q{c(l);}vsi;i=l;c(l);vni; N}$

181 : SAVESTATE: 26

181 : q{c(l);}vsi;i=l;c(l);vni; q{N}N}$

182 : {c(l);}vsi;i=l;c(l);vni;i {N}N}$

183 : c(l);}vsi;i=l;c(l);vni;i= N}N}$

184 :N->c(i);N c(l);}vsi;i=l;c(l);vni;i= N}N}$

185 : SAVESTATE: 27

185 : c(l);}vsi;i=l;c(l);vni;i= c(i);N}N}$

186 : (l);}vsi;i=l;c(l);vni;i=m (i);N}N}$

187 : l);}vsi;i=l;c(l);vni;i=m( i);N}N}$

188 : TS\_NOK/NS\_NORULECHAIN

188 : RESSTATE

188 : c(l);}vsi;i=l;c(l);vni;i= N}N}$

189 :N->c(l);N c(l);}vsi;i=l;c(l);vni;i= N}N}$

190 : SAVESTATE: 27

190 : c(l);}vsi;i=l;c(l);vni;i= c(l);N}N}$

191 : (l);}vsi;i=l;c(l);vni;i=m (l);N}N}$

192 : l);}vsi;i=l;c(l);vni;i=m( l);N}N}$

193 : );}vsi;i=l;c(l);vni;i=m(k );N}N}$

194 : ;}vsi;i=l;c(l);vni;i=m(k, ;N}N}$

195 : }vsi;i=l;c(l);vni;i=m(k,k N}N}$

196 : TNS\_NORULECHAIN/NS\_NORULE

196 : RESSTATE

196 : c(l);}vsi;i=l;c(l);vni;i= N}N}$

197 :N->c(i); c(l);}vsi;i=l;c(l);vni;i= N}N}$

198 : SAVESTATE: 27

198 : c(l);}vsi;i=l;c(l);vni;i= c(i);}N}$

199 : (l);}vsi;i=l;c(l);vni;i=m (i);}N}$

200 : l);}vsi;i=l;c(l);vni;i=m( i);}N}$

201 : TS\_NOK/NS\_NORULECHAIN

201 : RESSTATE

201 : c(l);}vsi;i=l;c(l);vni;i= N}N}$

202 :N->c(l); c(l);}vsi;i=l;c(l);vni;i= N}N}$

203 : SAVESTATE: 27

203 : c(l);}vsi;i=l;c(l);vni;i= c(l);}N}$

204 : (l);}vsi;i=l;c(l);vni;i=m (l);}N}$

205 : l);}vsi;i=l;c(l);vni;i=m( l);}N}$

206 : );}vsi;i=l;c(l);vni;i=m(k );}N}$

207 : ;}vsi;i=l;c(l);vni;i=m(k, ;}N}$

208 : }vsi;i=l;c(l);vni;i=m(k,k }N}$

209 : vsi;i=l;c(l);vni;i=m(k,k) N}$

210 :N->vni;N vsi;i=l;c(l);vni;i=m(k,k) N}$

211 : SAVESTATE: 28

211 : vsi;i=l;c(l);vni;i=m(k,k) vni;N}$

212 : si;i=l;c(l);vni;i=m(k,k); ni;N}$

213 : TS\_NOK/NS\_NORULECHAIN

213 : RESSTATE

213 : vsi;i=l;c(l);vni;i=m(k,k) N}$

214 :N->vsi;N vsi;i=l;c(l);vni;i=m(k,k) N}$

215 : SAVESTATE: 28

215 : vsi;i=l;c(l);vni;i=m(k,k) vsi;N}$

216 : si;i=l;c(l);vni;i=m(k,k); si;N}$

217 : i;i=l;c(l);vni;i=m(k,k);c i;N}$

218 : ;i=l;c(l);vni;i=m(k,k);c( ;N}$

219 : i=l;c(l);vni;i=m(k,k);c(i N}$

220 :N->i=E;N i=l;c(l);vni;i=m(k,k);c(i N}$

221 : SAVESTATE: 29

221 : i=l;c(l);vni;i=m(k,k);c(i i=E;N}$

222 : =l;c(l);vni;i=m(k,k);c(i) =E;N}$

223 : l;c(l);vni;i=m(k,k);c(i); E;N}$

224 :E->l l;c(l);vni;i=m(k,k);c(i); E;N}$

225 : SAVESTATE: 30

225 : l;c(l);vni;i=m(k,k);c(i); l;N}$

226 : ;c(l);vni;i=m(k,k);c(i);r ;N}$

227 : c(l);vni;i=m(k,k);c(i);rk N}$

228 :N->c(i);N c(l);vni;i=m(k,k);c(i);rk N}$

229 : SAVESTATE: 31

229 : c(l);vni;i=m(k,k);c(i);rk c(i);N}$

230 : (l);vni;i=m(k,k);c(i);rk; (i);N}$

231 : l);vni;i=m(k,k);c(i);rk;} i);N}$

232 : TS\_NOK/NS\_NORULECHAIN

232 : RESSTATE

232 : c(l);vni;i=m(k,k);c(i);rk N}$

233 :N->c(l);N c(l);vni;i=m(k,k);c(i);rk N}$

234 : SAVESTATE: 31

234 : c(l);vni;i=m(k,k);c(i);rk c(l);N}$

235 : (l);vni;i=m(k,k);c(i);rk; (l);N}$

236 : l);vni;i=m(k,k);c(i);rk;} l);N}$

237 : );vni;i=m(k,k);c(i);rk;} );N}$

238 : ;vni;i=m(k,k);c(i);rk;} ;N}$

239 : vni;i=m(k,k);c(i);rk;} N}$

240 :N->vni;N vni;i=m(k,k);c(i);rk;} N}$

241 : SAVESTATE: 32

241 : vni;i=m(k,k);c(i);rk;} vni;N}$

242 : ni;i=m(k,k);c(i);rk;} ni;N}$

243 : i;i=m(k,k);c(i);rk;} i;N}$

244 : ;i=m(k,k);c(i);rk;} ;N}$

245 : i=m(k,k);c(i);rk;} N}$

246 :N->i=E;N i=m(k,k);c(i);rk;} N}$

247 : SAVESTATE: 33

247 : i=m(k,k);c(i);rk;} i=E;N}$

248 : =m(k,k);c(i);rk;} =E;N}$

249 : m(k,k);c(i);rk;} E;N}$

250 :E->m(W) m(k,k);c(i);rk;} E;N}$

251 : SAVESTATE: 34

251 : m(k,k);c(i);rk;} m(W);N}$

252 : (k,k);c(i);rk;} (W);N}$

253 : k,k);c(i);rk;} W);N}$

254 :W->k,W k,k);c(i);rk;} W);N}$

255 : SAVESTATE: 35

255 : k,k);c(i);rk;} k,W);N}$

256 : ,k);c(i);rk;} ,W);N}$

257 : k);c(i);rk;} W);N}$

258 :W->k,W k);c(i);rk;} W);N}$

259 : SAVESTATE: 36

259 : k);c(i);rk;} k,W);N}$

260 : );c(i);rk;} ,W);N}$

261 : TS\_NOK/NS\_NORULECHAIN

261 : RESSTATE

261 : k);c(i);rk;} W);N}$

262 :W->k k);c(i);rk;} W);N}$

263 : SAVESTATE: 36

263 : k);c(i);rk;} k);N}$

264 : );c(i);rk;} );N}$

265 : ;c(i);rk;} ;N}$

266 : c(i);rk;} N}$

267 :N->c(i);N c(i);rk;} N}$

268 : SAVESTATE: 37

268 : c(i);rk;} c(i);N}$

269 : (i);rk;} (i);N}$

270 : i);rk;} i);N}$

271 : );rk;} );N}$

272 : ;rk;} ;N}$

273 : rk;} N}$

274 :N->ri; rk;} N}$

275 : SAVESTATE: 38

275 : rk;} ri;}$

276 : k;} i;}$

277 : TS\_NOK/NS\_NORULECHAIN

277 : RESSTATE

277 : rk;} N}$

278 :N->rk; rk;} N}$

279 : SAVESTATE: 38

279 : rk;} rk;}$

280 : k;} k;}$

281 : ;} ;}$

282 : } }$

283 : $

284 : LENTA\_END

285 : ------>LENTA\_END

-------------------------------------------------------------------------------------

0 :всего строк 122, синтаксический анализ выполнен без ошибок

0 : S->fni(F){N}S

4 : F->ni,F

7 : F->ni

11 : N->vni;N

15 : N->i=E;N

17 : E->iM

18 : M->-E

19 : E->i

21 : N->ri;

25 : S->e{N}

27 : N->vni;N

31 : N->i=E;N

33 : E->k

35 : N->vni;N

39 : N->i=E;N

41 : E->k

43 : N->vni;N

47 : N->i=E;N

49 : E->i(W)

51 : W->i,W

53 : W->i

56 : N->c(l);N

61 : N->c(i);N

66 : N->h(i<k){N}N

73 : N->c(l);

79 : N->q{N}N

81 : N->c(l);

87 : N->vsi;N

91 : N->i=E;N

93 : E->l

95 : N->c(l);N

100 : N->vni;N

104 : N->i=E;N

106 : E->m(W)

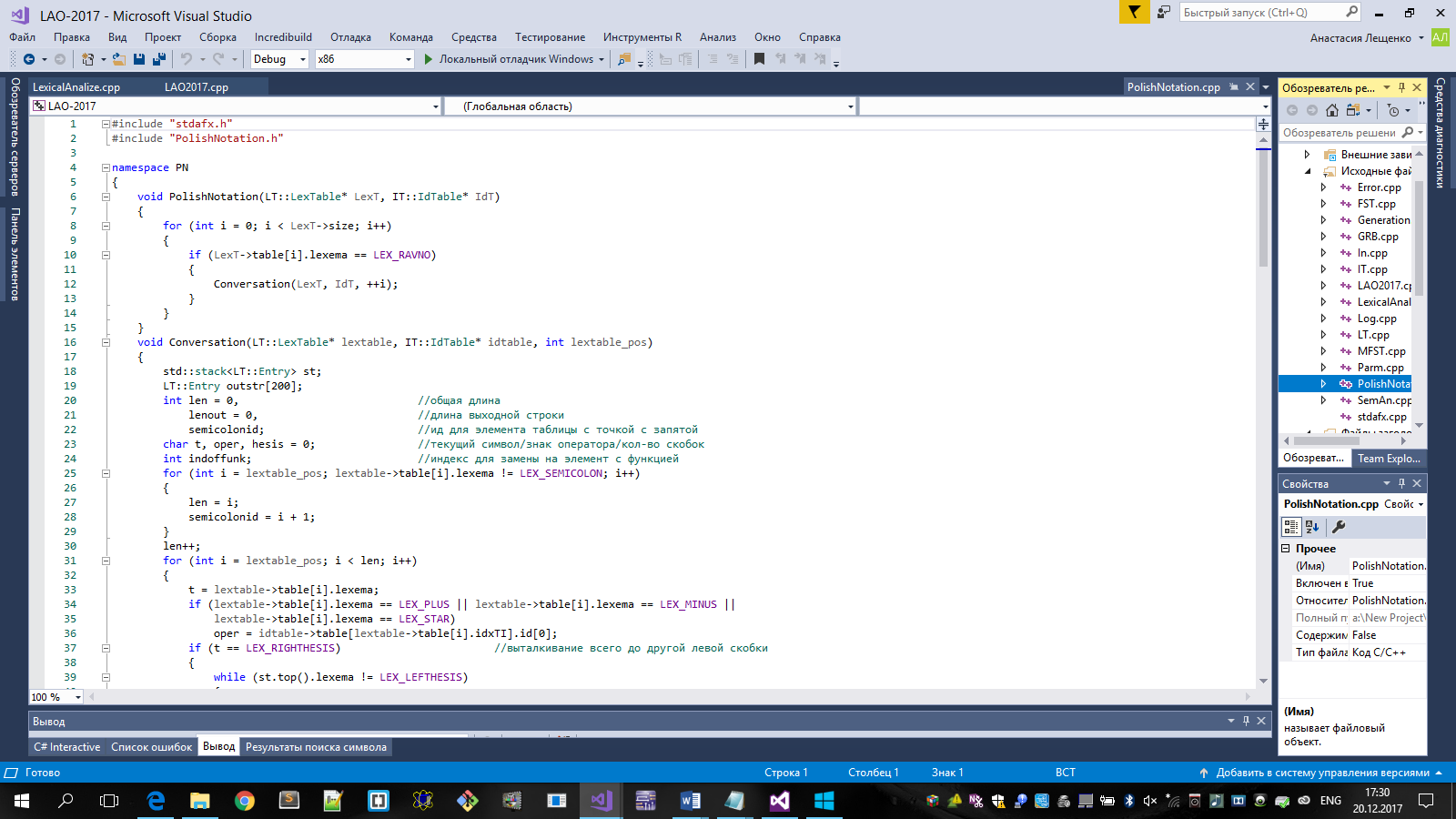
108 : W->k,W

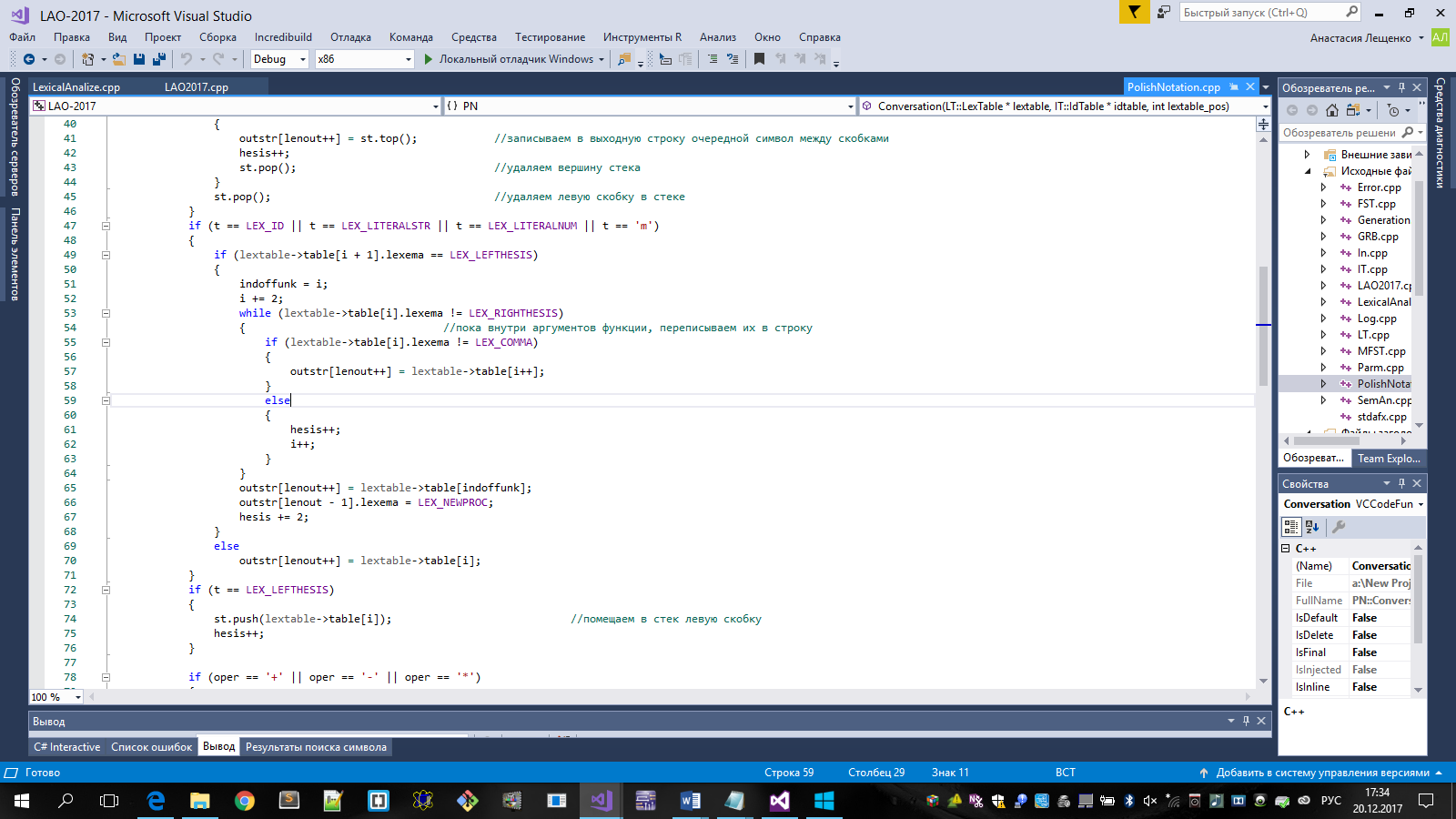
110 : W->k

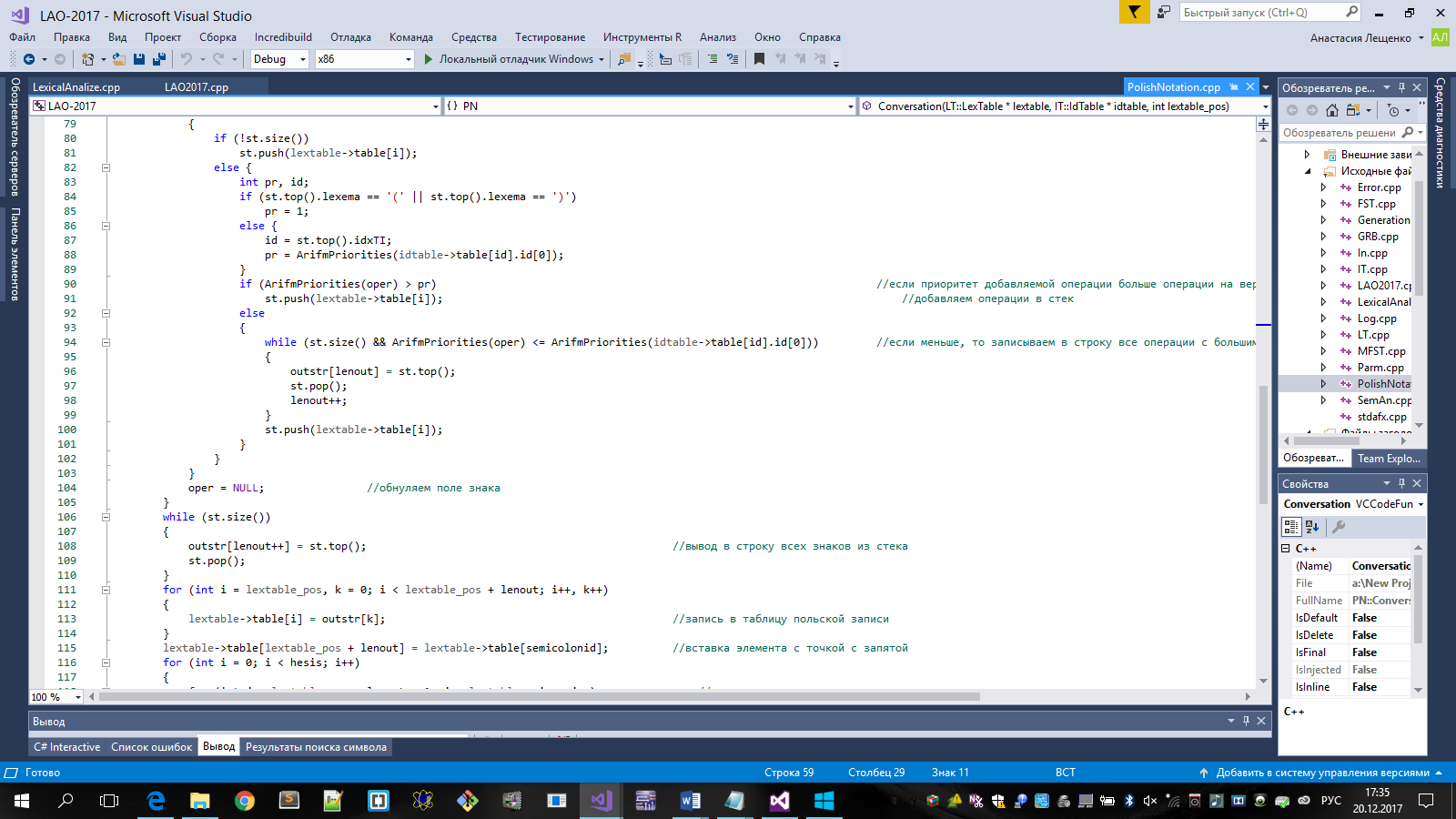
113 : N->c(i);N

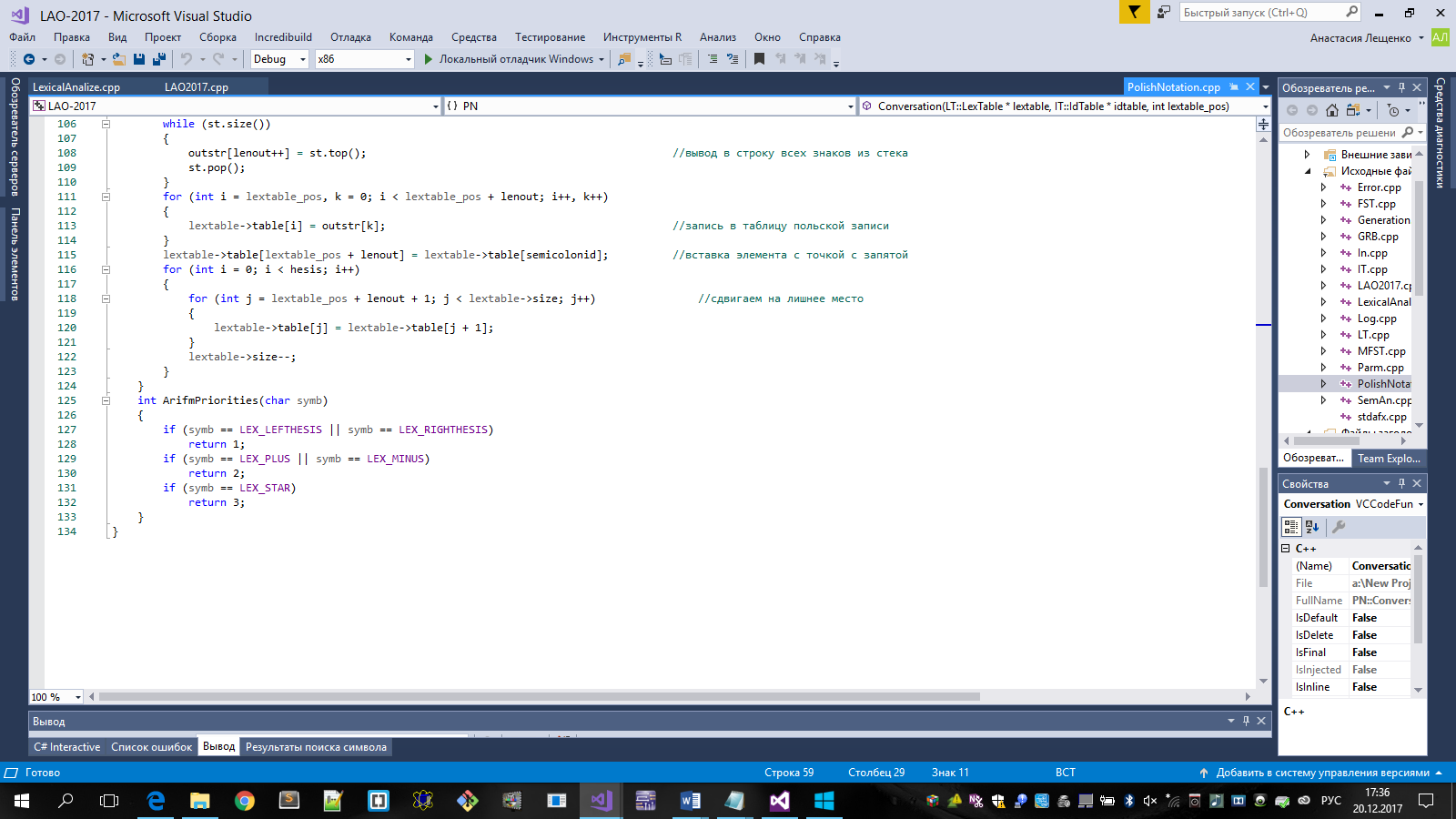
118 : N->rk;

**Приложение 3**









**Приложение 4**

.586

.model flat, stdcall

includelib libucrt.lib

includelib kernel32.lib

ExitProcess PROTO :DWORD

includelib ../Debug/LAOLib.lib

consoles PROTO :DWORD

consolen PROTO :DWORD

sum PROTO :DWORD, :DWORD

.stack 4096

.const

literal0 DWORD 5

literal1 DWORD 1

literal2 BYTE "k", 0

literal3 DWORD 0

literal4 BYTE "less0", 0

literal5 BYTE "more0", 0

literal6 BYTE "first", 0

literal7 BYTE "second", 0

literal8 DWORD 2

literal9 DWORD 2

literal10 DWORD 0

.data

craz0 DWORD ?

ient DWORD ?

tent DWORD ?

kent DWORD ?

v DWORD ?

s DWORD ?

.code

razn PROC braz0:DWORD, araz0:DWORD

push araz0

push braz0

pop ebx

pop eax

sub eax, ebx

push eax

pop craz0

mov eax, craz0

ret

razn ENDP

main PROC

push literal0

pop ient

push literal1

pop tent

push ient

push tent

call razn

push eax

pop kent

push offset literal2

call consoles

push kent

call consolen

mov eax, ient

cmp eax, literal3

jb less0

ja more0

less0:

push offset literal4

call consoles

jmp flag0

more0:

push offset literal5

call consoles

flag0:

push offset literal6

pop v

push offset literal7

call consoles

push literal8

push literal9

call sum

push eax

pop s

push s

call consolen

push 0

call ExitProcess

main ENDP

end main

**Используемая литература**

1. https://cppstudio.com
2. https:://msdn.microsoft.com

**Заключение**

Таким образом, разработка данного транслятора позволила изучить дополнительный материал, касающийся языков программирования, учитывая различные тонкости.

В итоге, все полученные знания были закреплены на практике.