МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора BED-2020»

Выполнил студент Батурель Евгений Дмитриевич

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Пацей Н.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты ст.пр. Наркевич Аделина Сергеевна

(учен. степень, звание, должность, подпись, Ф.И.О.)

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2020

Содержание

Введение 5

[Глава 1. Спецификация языка программирования 6](#_Toc58808109)

[1.1 Характеристика языка программирования 6](#_Toc58808110)

[1.2 Алфавит языка 6](#_Toc58808111)

[1.3 Применяемые сепараторы 7](#_Toc58808112)

[1.4 Применяемые кодировки 8](#_Toc58808113)

[1.5 Типы данных 8](#_Toc58808114)

[1.6 Преобразование типов данных 9](#_Toc58808115)

[1.7 Идентификаторы 9](#_Toc58808116)

[1.8 Литералы 10](#_Toc58808117)

[1.9 Объявления данных и область видимости 10](#_Toc58808118)

[1.10 Инициализация данных 10](#_Toc58808119)

[1.11 Инструкции языка 11](#_Toc58808120)

[1.12 Операции языка 11](#_Toc58808121)

[1.13 Выражения и их вычисления 12](#_Toc58808122)

[1.14 Программные конструкции языка 12](#_Toc58808123)

[1.15 Область видимости идентификаторов 12](#_Toc58808124)

[1.16 Семантические проверки 12](#_Toc58808125)

[1.17 Распределение оперативной памяти на этапе выполнения 13](#_Toc58808126)

[1.18 Стандартная библиотека и её состав 13](#_Toc58808127)

[1.19 Ввод и вывод данных 14](#_Toc58808128)

[1.20 Точка входа 14](#_Toc58808129)

[1.21 Препроцессор 14](#_Toc58808130)

[1.22 Соглашения о вызовах 14](#_Toc58808131)

[1.23 Объектный код 14](#_Toc58808132)

[1.24 Классификация сообщений транслятора 14](#_Toc58808133)

[1.25 Контрольный пример 14](#_Toc58808134)

[Глава 2. Структура транслятора 15](#_Toc58808135)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc58808136)

[2.2 Перечень входных параметров транслятора 16](#_Toc58808137)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 16](#_Toc58808138)

[Глава 3. Разработка лексического анализатора 17](#_Toc58808139)

[3.1 Структура лексического анализатора 17](#_Toc58808140)

[3.2 Контроль входных символов 17](#_Toc58808141)

[3.3 Удаление избыточных символов 18](#_Toc58808142)

[3.4. Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем 18](#_Toc58808143)

[3.6 Принцип обработки ошибок 20](#_Toc58808144)

[3.7 Структура и перечень сообщений лексического анализатора 20](#_Toc58808145)

[Глава 4. Разработка синтаксического анализатора 22](#_Toc58808146)

[4.1 Структура синтаксического анализатора. 22](#_Toc58808147)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 22](#_Toc58808148)

[4.3 Построение конеченого магазинного автомата 23](#_Toc58808149)

[4.4 Основные структуры данных 24](#_Toc58808150)

[4.5 Описание алгоритма синтаксического разбора 24](#_Toc58808151)

[4.6 Структура и перечень сообщений синтаксического анализатора 24](#_Toc58808152)

[4.7 Параметры синтаксического анализатора и режимы его работы 25](#_Toc58808153)

[4.8 Принцип обработки ошибок 25](#_Toc58808154)

[4.9 Контрольный пример 25](#_Toc58808155)

[Глава 5. Разработка семантического анализатора 26](#_Toc58808156)

[5.1 Структура семантического анализатора 26](#_Toc58808158)

[5.2 Функции семантического анализатора 26](#_Toc58808159)

[5.3 Структура и перечень сообщений семантического анализатора 26](#_Toc58808160)

[5.4 Принцип обработки ошибок 27](#_Toc58808161)

[5.5 Контрольный пример 27](#_Toc58808162)

[Глава 6. Преобразование выражений 28](#_Toc58808163)

[6.1 Выражения, допускаемые языком 28](#_Toc58808164)

[6.2 Польская запись и принцип ее построения 28](#_Toc58808165)

[6.3 Программная реализация обработки выражений 29](#_Toc58808166)

[6.4 Контрольный пример 29](#_Toc58808167)

[Глава 7. Генерация кода 30](#_Toc58808168)

[7.1 Структура генератора кода 30](#_Toc58808169)

[7.2 Представление типов данных в памяти 30](#_Toc58808170)

[7.3 Особенности алгоритма генерации кода 31](#_Toc58808171)

[Сгенерированный код можно посмотреть в приложении Е. 31](#_Toc58808172)

[8. Тестирование транслятора 32](#_Toc58808173)

[Заключение 35](#_Toc58808174)

[Приложение А 36](#_Toc58808175)

[Приложение Б 37](#_Toc58808176)

[Приложение В 39](#_Toc58808177)

[Приложение Г 41](#_Toc58808178)

[Приложение Д 43](#_Toc58808179)

[Приложение Е 45](#_Toc58808180)

[Литература 48](#_Toc58808181)

**Введение**

Транслятор – это комплекс отдельных программ, позволяющих преобразовывать исходный код на одном языке программирования в исходный код на другом языке программирования.

Классический транслятор состоит из следующих частей:

* лексический анализатор;
* синтаксический анализатор;
* семантический анализатор;
* генератор кода, или интерпретатор.

Все части транслятора, взаимодействуя между собой, обрабатывают входной текст и строят для него эквивалентный текст на понятном компьютеру языке программирования.

Исходя из цели курсового проекта, были определены следующие задачи:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка выражений;

– генерация кода на язык Assembler;

– тестирование транслятора.

Решения каждой из поставленных задач будут приведены в соответствующих главах курсового проекта, а именно:

1. спецификация языка программирования;
2. структура транслятора;
3. лексический анализатор;
4. синтаксический анализатор;
5. семантический анализатор;
6. преобразование выражений;
7. генерация кода;
8. тестирование транслятора.

Язык программирования BED-2020 предназначен для работы с консолью, выполнения простейших арифметических операций и операций сравнения над логическими переменными.

Пояснительная записка описывает правила и требования в использовании, принцип работы, реализацию разработанного языка программирования и компилятора.

# Глава 1. Спецификация языка программирования

* 1. **Характеристика языка программирования**

Язык программирования BED-2020 является универсальным языком высокого уровня. Он является транслируемым, не объектно-ориентированным. Язык строго типизируемый, что говорит о невозможности преобразования типов, транслируемым языком программирования. В BED-2020 используется три типа данных: целочисленные (integer), строковые (string), а также беззнаковые целочисленные (unsigned integer).

* 1. **Алфавит языка**

Совокупность символов, используемых в языке, называется алфавитом языка. Исходный код BED-2020 может содержать символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, символы арифметических операций и операций сравнения, а также символы-разделители. Русские символы разрешены только в строковых литералах. Алфавит языка BED-2020 формально представлен как совокупность правил в форме Бэкуса-Наура в таблице 1.1.

Таблица 1.1 – Алфавит языка программирования BED-2020

|  |
| --- |
| Правило |
| <строчная буква латинского алфавита>::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z |
| <прописная буква латинского алфавита>::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z |
| <цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| <строчная буква русского алфавита>::= а|б|в|г|д|е|ё|ж|з|и|й|к|л|м|н|о|п|р|с|т|у|ф|х|ц|ч|ш|щ|ъ|ь|э|ю|я |
| <прописная буква русского алфавита>::= А|Б|В|Г|Д|Е|Ё|Ж|З|И|Й|К|Л|М|Н|О|П|Р|С|Т|У|Ф|Х|Ц|Ч|Ш|Щ|Ъ|Ь|Э|Ю|Я |
| <арифметическая операция>::= +|-|\*|/ |
| <символ- сепаратор>::= ' '|,|(|)|{|}|;|: |
| <логическая операция>::= >|<|==|!=|<=|>= |

* 1. **Применяемые сепараторы**

Сепараторы необходимы для разделения операций языка. Сепараторы, используемые в языке программирования BED-2020, приведены в таблице 1.2.

Таблица 1.2 - Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ;  ‘ ‘(пробел) | Разделение инструкций |
| , | Разделение параметров функции |
| >, <  ==  !=  <=, >= | Символы сравнения |
| = | Оператор присвоения |
| +  -  \*  / | Арифметические операции |
| {  } | Программный блок инструкций |
| (  ) | Параметры функций/приоритетность операций (в выражениях) |

* 1. **Применяемые кодировки**

При трансляции применяется кодировка Windows-1251. Кодировка представлена на рисунке 1.1

Рисунок 1.1 – кодировка Windows-1251



* 1. **Типы данных**

Пользовательские типы данных не поддерживаются. Допускается использование фундаментальных типов данных определенных в таблице 1.3.

Таблица 1.3 - Фундаментальные типы данных

|  |  |
| --- | --- |
| Тип данных | Описание |
| integer | Является целочисленным типом данных. Этот тип данных занимает 4 байта. Предусмотрен для объявления целочисленных данных. Инициализация по умолчанию: 0. Диапазон допустимых значений: от -231 до 231 – 1.  Предусмотрены следующие арифметические операции:  + - бинарная операция суммирования(num+num);  - - бинарная операция вычитания(num-num);  \* - бинарная операция умножения(num\*num);  / - бинарная операция деления(num/num). |
| string | Является строковым типом данных. Предназначен для работы с символами, каждый символ занимает 1 байт. Максимальное количество символов – 255, включая символ окончания строки.  Инициализация по умолчанию: нулевой символ.  Операции над данными строкового типа: возможно присваивание строковому идентификатору значения другого строкового идентификатора, строкового литерала или значения строковой функции. |
| Unsigned integer | Является беззнаковым целочисленным типом данных. Этот тип данных занимает 4 байта. Диапазон значений от 0 до 232 – 1. Предназначен для объявления целочисленных положительных данных.. Инициализация по умолчанию: 0.  Предусмотрены следующие арифметические операции:  + - бинарная операция суммирования(num+num);  - - бинарная операция вычитания(num-num);  \* - бинарная операция умножения(num\*num);  / - бинарная операция деления(num/num). |

* 1. **Преобразование типов данных**

Преобразование не поддерживается, все типы данных определены однозначно и не могут быть преобразованы в другие, так как язык BED-2020 является строго типизируемым.

* 1. **Идентификаторы**

Для именования функций, параметров и переменных используются идентификаторы. Не предусмотрены зарезервированные идентификаторы. Имя идентификатора составляется следующим образом:

* состоит из символов латинского алфавита [a..z].
* максимальная длина идентификатора равна 10 и не должна превышать это значение. При превышении максимально значения длина идентификатора усекается до 10.
  1. **Литералы**

Предусмотрены числовые (integer и unsigned integer) и строковые (string) литералы. Правила записи приведены в таблице 1.4.

Таблица 1.4 - Правила записи литералов

|  |  |
| --- | --- |
| Тип литерала | Описание литерала |
| Числовые | Максимально допустимое значение для integer равно . Минимально допустимое значение равно . Могут состоять только из цифр [0..9] без дробной части. При выходе за пределы допустимости будет выведена соответствующая ошибка.  Максимально допустимое значение для unsigned integer равно . Минимально допустимое значение равно 0. Могут состоять только из цифр [0..9] без дробной части. При выходе за пределы допустимости будет выведена соответствующая ошибка. |
| Строковые | Состоит из символов, заключенных в  ' '(одинарные кавычки). Максимальное число которых не может превышать . В случае превышения длины литерала работа транслятора прекращается. |

* 1. **Объявления данных и область видимости**

В языке программирования BED-2020 переменная должна быть объявлена до ее использования. Областью видимости переменной является блок функции, в которой она определена. Не допустимо объявление глобальных переменных. Область видимости схожа с областью видимости C++, то есть сверху вниз.

* 1. **Инициализация данных**

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть идентификаторы, литералы, выражения и вызовы функций. Предусмотрены значения по умолчанию, если переменные не инициализированы: значение 0 для целочисленного и беззнакового целочисленного типов данных и строка длины 0 ("") для строкового типа данных.

* 1. **Инструкции языка**

В языке программирования BED-2020 применяются инструкции, представ-ленные в таблице 1.5.

Таблица 1.5 - Инструкции языка

|  |  |
| --- | --- |
| Инструкция языка | Синтаксис |
| Объявление функции | <тип данных>function<идентификатор>(<тип данных><идентификатор>, …)  {  …  } |
| Объявление переменной | <тип данных><идентификатор> |
| Присваивание | string и integer/unsigned integer:  <идентификатор> = <литерал>;  <идентификатор> = <выражение>;  <идентификатор 1> = <идентификатор 2>; |
| Вывод данных | output<идентификатор / литерал> |
| Условный оператор | If(<выражение>){…} |

* 1. **Операции языка**

Все операции в языке BED-2020 применимы только к целочисленным типам данных.

Наибольшую приоритетность арифметических операций имеют операции умножения и деления, а сложение и вычитание меньшую. При одинаковом приоритете первой выполнится операция расположенная левее. Изменить приоритетность можно с помощью круглых скобок.

Операции в языке программирования BED-2020 приведены в таблице 1.6.

Таблица 1.6 - Операции языка

|  |  |
| --- | --- |
| Тип | Операторы |
| Арифметические | () – приоритетность операций  + ̶ сложение  - ̶ вычитание  \* ̶ умножение  / ̶ деление |
| Сравнения | > - больше  < - меньше  >= - больше или равно  <= - меньше или равно  == - равно  != - не равно |

* 1. **Выражения и их вычисления**

Предусмотрены следующие правила составления выражений:

* + выражения читаются слева направо и записываются в одну строку;
  + реализация выражений происходит с помощью обратной польской записи;
  + для изменения приоритета операция используются круглые скобки.
  1. **Программные конструкции языка**

Программные конструкции представлены в таблице 1.7.

Таблица 1.7 - Программные конструкции языка

|  |  |
| --- | --- |
| Конструкция | Представление в языке |
| Главная функция | main  {  <инструкции языка>  } |
| Функция | <тип данных> function <идентификатор>(<тип данных> <идентификатор>, …)  {  return<идентификатор>или<литерал>;  } |
| Блок | {  …  } |

* 1. **Область видимости идентификаторов**

Все идентификаторы обязаны быть объявленными внутри функции. Вне функции объявление идентификаторов недопустимо. Глобальных переменных нет, только локальные. Параметры видны только внутри функции, в которой объявлены.

* 1. **Семантические проверки**

Перечень семантических проверок, предусмотренных языком, приведен в таблице 1.8.

Таблица 1.8 - Перечень семантических проверок

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Наличие функции main, как точки входа в программу |
| 2 | Наличие только одной точки входа |
| 3 | Переопределение идентификаторов |
| 4 | Использование идентификаторов без их объявления |
| 5 | Проверка на количество параметров объявляемых функций и ограничения применяемые к ним |
| 6 | Правильность передаваемых в функцию параметров: количество, типы |
| 7 | Правильность строковых выражений |
| 8 | Превышение размера строковых и числовых литералов |
| 9 | Тип данных переменной должен совпадать с типом значения, которое присваивается этому типу |

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

Все переменные размещаются в стеке. Таблица лексем и таблица идентификаторов сохраняются в структуры с выделенной под них динамической памятью, которая очищается по окончанию работы транслятора.

* 1. **Стандартная библиотека и её состав**

В языке BED-2020 присутствует стандартная библиотека, которая автоматически подключается при трансляции исходного кода в язык ассемблера. Можно пользоваться двумя видами функций стандартной библиотеки: функция вычисления времени и функция вычисления даты. Содержимое библиотеки и описание функций представлено в таблице 1.9.

Таблица 1.9 - Стандартная библиотека языка BED-2020

|  |  |
| --- | --- |
| Функция | Описание |
| void function date(); | Функция - процедура.  Принимаемых параметров нет.  Возвращаемого значения нет.  Выводит в консоль текущую дату. |
| void function time(); | Функция - процедура.  Принимаемых параметров нет.  Возвращаемого значения нет.  Выводит в консоль текущее время. |

* 1. **Ввод и вывод данных**

Ввод данных не предусмотрен. Вывод данных осуществляется с помощью ключевого слова output. В качестве аргумента принимаются числовые и строковые идентификаторы, так же литералы:

output<идентификатор>;

output<литерал>;

* 1. **Точка входа**

В языке BED-2020 точкой входа является ключевое слово “main”. Точка входа не может отсутствовать и должна быть единственной.

* 1. **Препроцессор**

В языке BED-2020 препроцессоры не предусмотрены.

* 1. **Соглашения о вызовах**

При генерации кода используется соглашение \_stdcall, в котором все параметры передаются в стек справа налево. Освобождением памяти занимается вызываемая подпрограмма, которая очищает стек.

* 1. **Объектный код**

Исходный код языка транслируется в язык ассемблера.

* 1. **Классификация сообщений транслятора**

Транслятор генерирует сообщения об ошибках пользователю и выводит их в файл протокола. В соответствии с префиксами будут различаться сообщения, представленные в таблице 1.10.

Таблица 1.10 - Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибки |
| 0 - 99 | Системные ошибки |
| 100 - 109 | Ошибки параметров |
| 110 - 119 | Ошибки открытия и чтения файлов |
| 120 - 600 | Лексические и семантические ошибки |
| 600 - … | Синтаксические ошибки |

* 1. **Контрольный пример**

Контрольный пример, написанный на языке BED-2020, представлен в приложении А.

**Глава 2. Структура транслятора**

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

Исходный код, написанный на языке программирования BED-2020, является для транслятора входными данными.

Как выходные данные используется объектный код и протоколы работы транслятора, описанные в пункте 2.3.

Компоненты транслятора приведены на рисунке 2.1.



Рис 2.1 - Структура транслятора BED-2020

Первоначально на вход лексического анализатора передается исходный код. Анализатором проверяется исходный текст на недопустимые символы, выделяет литералы, идентификаторы и ключевые слова, а также формирует таблицы лексем и идентификаторов.

Далее наступает черед синтаксического анализатора, к нему на вход поступа-ет таблица лексем, полученная на этапе лексического анализа. Если программа по-строена синтаксически правильно, то осуществляется переход к этапу семантического анализа, стоящему далее, при ином раскладе работа транслятора останавливается.

Семантический анализ предполагает проверку исходного кода на семантическую согласованность конструкций языка, проверяет правильность текста исходной программы с точки зрения семантики, после чего, если код верный с точки зрения семантики, то следующий этап – это генерация кода.

Генерация кода реализуется посредством чистой интерпретации, без создания промежуточного представления кода. В финале происходит генерация кода, во время исполнения которого формируется объектный код.

**2.2 Перечень входных параметров транслятора**

В таблице 2.1 представлены входные параметры, которые могут использоваться для представления работы транслятора.

Таблица 2.1 -Входные параметры транслятора

|  |  |  |
| --- | --- | --- |
| Параметр | Назначение | Тип |
| -in: | Указывает на файл с исходным кодом. Исходный код содержится в файле с расширением \*.txt | Обязательный |
| -out: | Файл для записи результата работы транслятора. | Не обязательный |
| -log: | Файл для записи результата работы программы. | Не обязательный |

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

По итогам своей работы транслятор формирует протокол, согласно заданным входным параметрам. –log: <путь к файлу> - в этом файле находится информация о входных параметрах приложения, этапе проверки символов на допустимость, результат работы лексического и семантического анализаторов. –out: <путь к файлу> - в этом файле содержится сгенерированный код на языке Ассемблера.

# Глава 3. Разработка лексического анализатора

**3.1 Структура лексического анализатора**

Структура лексического анализатора представлена на рисунке 3.1.



Рис 3.1 - Структура лексического анализатора

Лексический анализатор читает поток символов, составляющих исходную программу, и группирует эти символы в значащие последовательности, называющиеся лексемами. Лексема – это структурная единица языка, которая состоит из элементарных символов языка и не содержит в своём составе других структурных единиц языка. Лексемами языков программирования являются идентификаторы, константы, ключевые слова языка, знаки операций и т.п. На вход лексического анализатора поступает текст исходной программы, а выходная информация передаётся для дальнейшей обработки синтаксическому анализатору.

Исходный код на языке BED-2020 является входными данными.

Таблицы лексем и идентификаторов являются выходными данными.

**3.2 Контроль входных символов**

Таблица допустимости представлена на рисунке 3.2.

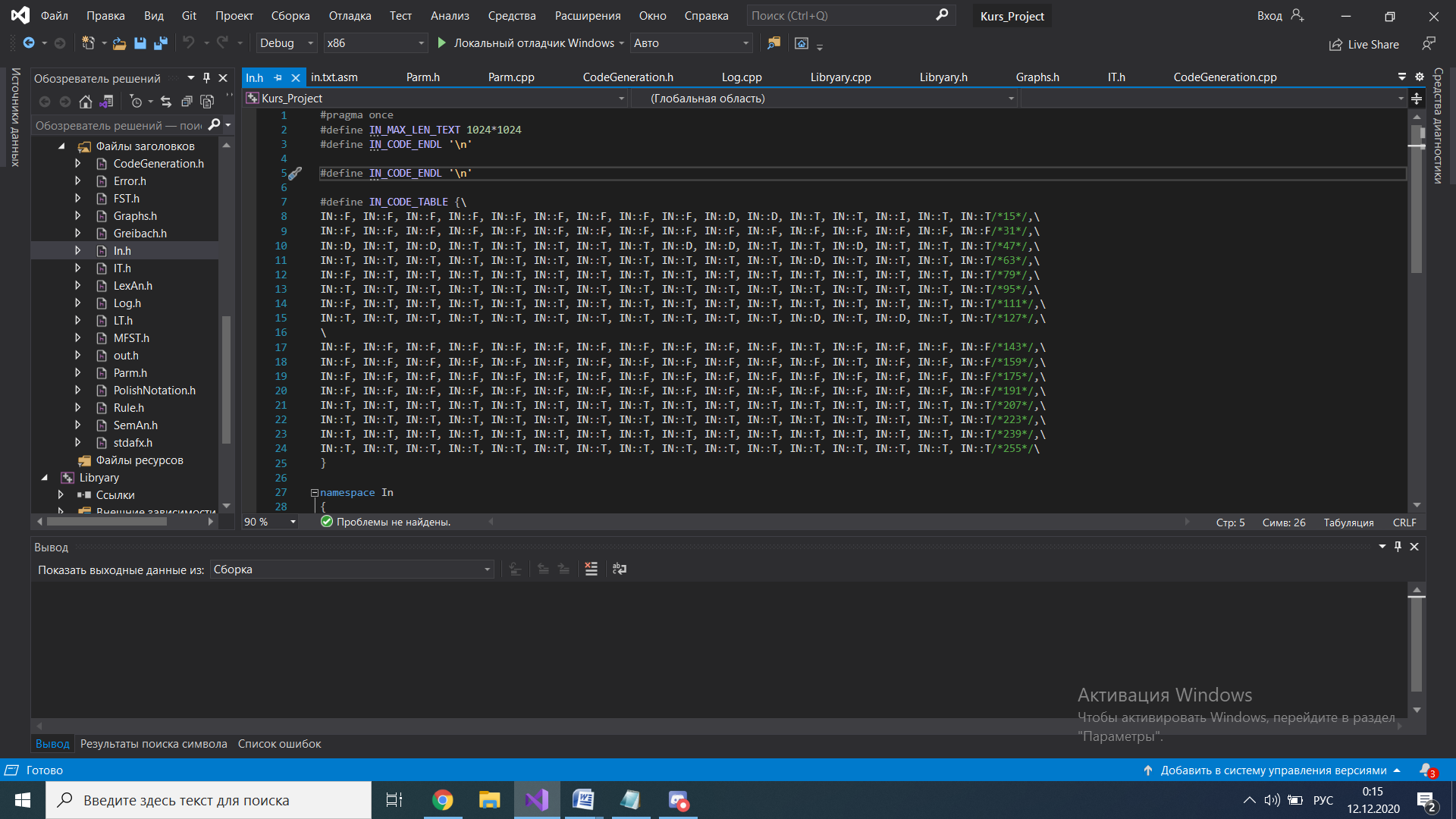


Рисунок 3.2 -Таблица допустимости входных символов

Таблица допустимости была сформирована на основе кодировки windows-1251. Таблица необходима для проверки входных символов на допустимость. Символы могут быть разрешенными, запрещенными, игнорируемыми и др. Символы представлены в шестнадцатеричной системе счисления. В таблице записаны различные числовые значения соответствующие символам в данной таблице:

«T» - разрешенные алфавитом символы;

«F» - запрещенные алфавитом символы;

«I» - символы, которые игнорируются;

«D» - символы, являющиеся сепараторами;

**3.3 Удаление избыточных символов**

Избыточными символами представлены пробелы и символы табуляции.

Алгоритм удаления избыточных символов:

Описание алгоритма удаления избыточных символов:

* посимвольно считываем файл с исходным кодом программы;
* если находим лишние пробелы или знак табуляции, то переходим к следующему символу, а если находим символ перехода на новую строку, то добавляем специальный символ-сепаратор “|”;
* продолжаем считывание файла по данному алгоритму;

## 3.4. Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем

Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем представлен в таблице 3.1.

Таблица 3.1 - Перечень ключевых слов

|  |  |
| --- | --- |
| Цепочка | Лексема |
| integer | t |
| string | t |
| unsigned integer | t |
| function | f |
| date | i |
| time | i |
| return | r |
| main | m |
| declare | d |
| call | c |
| + | v |
| - | v |
| \* | v |
| / | v |
| = | = |
| ( | ( |
| ) | ) |
| { | { |
| } | } |
| , | , |
| ; | ; |
| >= | ] |
| <= | [ |
| > | > |
| < | < |
| == | ~ |
| != | ! |
| идентификатор | i |
| строковый литерал | l |

## 3.5 Основные структуры данных

В приложении Б представлены основные структуры данных на этапе синтаксического анализа.

В таблице лексем содержатся лексемы, а также номер строки, в которой она находится. Таблица идентификаторов содержит сам идентификатор, тип данных, тип идентификатора, индекс в таблице лексем и значение.

## 3.6 Принцип обработки ошибок

В случае обнаружения ошибки, транслятор прекращает свою работу и в log-файл записывается соответствующее сообщение об ошибке с указанием строки и позиции.

## 3.7 Структура и перечень сообщений лексического анализатора

Перечень сообщений, генерируемых на этапе лексического анализа, представлен в таблице 3.2.

Таблица 3.2- Сообщения лексического анализатора

|  |  |
| --- | --- |
| Код | Сообщение |
| 2 | У вас нет main |
| 3 | в main нет return |
| 123 | Нераспознанная лексема |
| 125 | Перезапись идентификатора |
| 126 | Неизвестная переменная |
| 133 | Найдено ключевое слово call без соответствующих функций |
| 134 | Не найдено ключевого слова call при вызове функций даты/времени |

## 3.8 Параметры лексического анализатора и режим его работы

Входным параметром лексического анализатора является исходный текст программы, написанный на языке BED-2020

## 3.9 Алгоритм лексического анализа

Последовательность выполнения алгоритма работы лексического анализатора представлена ниже.

1. Считывание текста из исходного файла в буфер;
2. Далее – посимвольное считывание из буфера. Если текущий символ - конец строки – пункт 8;
3. Проверка символа на допустимость;
4. Если текущий символ сепаратор – запись в таблицу лексем. Если нет – дозапись в строку-буфер до тех пор, пока сепаратор не встретится;
5. Проверка строки-буфера на одно из зарезервированных слов. Если такое есть – запись в таблицу лексем и переход к пункту 2;
6. Проверка строки-буфера на соответствие одному из конечных автоматов. Если нет – пункт 8, выход из программы с ошибкой;
7. Запись в таблицу лексем соответствующего значения. Запись в таблицу идентификаторов того или иного идентификатора или литерала. Переход к пункту 2;
8. Конец работы анализатора.

## 3.10 Контрольный пример

На вход лексического анализатора подается программа на языке BED-2020, описанная в пункте 1.25. Результат работы лексического анализатора представлен в приложении В.

# Глава 4. Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора.

Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 - Структура синтаксического анализатора

Таблицы лексем и идентификаторов являются входными данными.

Дерево разбора является выходными данными.

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

Грамматика, описывающая язык BED-2020 представлена в таблице 4.1.

Таблица 4.1 - грамматика языка BED-2020

|  |  |
| --- | --- |
| Нетереминалы | Описание |
|  | Правила, описывающие общую структуру программы |
|  | Порождает правила, описывающие инструкции языка |
|  | Порождаетправила, описывающиевыражения |
|  | Порождает правила, описывающие арифметические действия |
|  | Порождает правила, описывающие формальные параметры функции |
|  | Порождает правила, описывающие фактические параметры функции |
|  | Порождает правила, описывающие условный оператор |

## 4.3 Построение конеченого магазинного автомата

Принцип действия конечного магазинного автомата представлен на рисунке 4.2.



Рис. 4.2 - МП-автомат

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.

Конфигурация (текущее состояние автомата) описывается тройкой , где - текущее состояние автомата, - остаток цепочки, - цепочка-содержимое магазина.

Начальное состояние , - начальное состояние автомата, - входная цепочка, - маркер дна магазина.

Цепочка  является допустимой (распознается) автоматом , если  и .

Работа автомата 

1. состояние автомата 
2. читает символ находящийся под головкой (сдвигает ленту);
3. не читает ничего (читает , не сдвигает ленту);
4. из  определяет новое состояние , если  или .
5. читает верхний (в стеке) символ  и записывает цепочку  т.к. , при этом, если , то верхний символ магазина просто удаляется.

работа автомата заканчивается 

## 4.4 Основные структуры данных

В приложении Г представлены основные структуры данных и правила пере-хода, используемые на этапе синтаксического анализа.

## 4.5 Описание алгоритма синтаксического разбора

Принцип работы конечного автомата с магазинной памятью представлен ниже.

1)В магазин записывается стартовый символ.

2)На основе полученной ранее таблицы лексем формируется входная лента.

3)Запускается автомат.

4)Выбирается цепочка по первому символу, соответствующая нетерминальному символу, записывается в магазин в обратном порядке.

5)Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и магазина. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другое правило нетерминала.

6)Если в правиле встретился нетерминал – пункт 4.

7)Если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

## 4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений, генерируемых на этапе синтаксического анализа, представлен в таблице 4.3.

Таблица 4.3 - Перечень сообщений

|  |  |
| --- | --- |
| Код | Сообщение |
| 600 | Неверная структура программы |
| 601 | Ошибочныйоператор |
| 602 | Ошибка в выражении |
| 603 | Ошибка в параметрах функции |
| 604 | Ошибка в параметрах вызываемой функции |
| 605 | Ошибочное определение логического условия |

## 4.7 Параметры синтаксического анализатора и режимы его работы

Таблицы идентификаторов и лексем являются входными параметрами для синтаксического анализатора. Эти таблицы мы получаем в ходе лексического анализа. В конце после разбора формируется дерево разбора, которое выводится на консоль.

## 4.8 Принцип обработки ошибок

Обработка ошибок происходит следующим образом:

* синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем;
* если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка, которая записывается в протокол работы и программа останавливается.

## 4.9 Контрольный пример

Результатом работы синтаксического анализатора является дерево разбора. Результат работы синтаксического анализатора представлен в приложении Д.

# Глава 5. Разработка семантического анализатора

1. 1. **Структура семантического анализатора**

Семантический анализатор состоит из набора функций для проверки правильности исходной программы. Функции анализатора выполняются на различных этапах работы транслятора. Структура семантического анализатора представлена на рисунке 5.1.

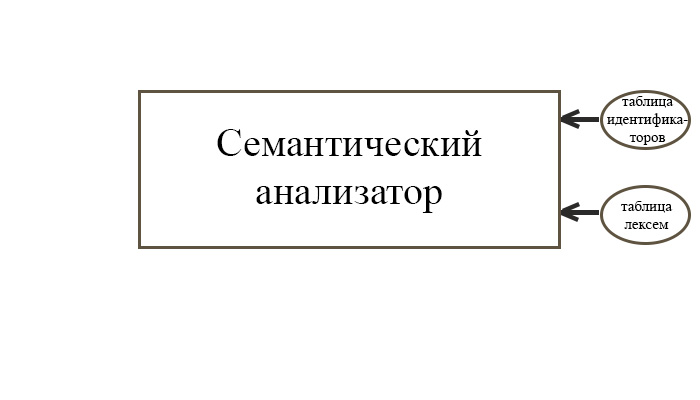


Рисунок 5.1 - Структура семантического анализатора

* 1. **Функции семантического анализатора**

Семантический анализатор выполняет проверку на основные правила языка (семантики языка), которые описаны в разделе 1.16.

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

Сообщения, генерируемые при выполнении семантических проверок, представлены в таблице 5.2.

Таблица 5.2 - Перечень сообщений

|  |  |
| --- | --- |
| Код | Сообщение |
| 132 | Имя идентификатора не может быть ключевым словом |
| 134 | Не найдено ключевого слова call при вызове функций даты/времени |
| 136 | Нельзя присваивать значение функции |
| 137 | Неверное количество параметров data/time |
| 138 | Неверное количество параметров вызываемой функции или неверные параметры |
| 139 | Тип функции и тип возвращаемого значения отличаются |
| 140 | Несоответствие присваемого типа типу переменной |
| 141 | Функция не возвращает значение |
| 142 | Нарушена структура программного блока |
| 146 | Неверный диапазон значений int |
| 147 | Неверный диапазон значений unsigned int |
| 149 | Нарушение формы логического выражения |

* 1. **Принцип обработки ошибок**

Все семантические ошибки являются критическими, из-за чего транслятор прекращает свою работу и в протокол работы транслятора выводится соответствующее сообщение об ошибке.

* 1. **Контрольный пример**

Результат работы семантических функций описан в главе 8.

**Глава 6. Преобразование выражений**

**6.1 Выражения, допускаемые языком**

В языке BED-2020 допускаются выражения с использованием числовых идентификаторов и литералов. Также предусмотрены следующие арифметические операции и операции сравнения:

* сложения: «+» ;
* вычитания: «-» ;
* умножения: «\*» ;
* деления: «/» ;
* больше: «>» ;
* меньше: «<» ;
* больше или равно: «>=» ;
* меньше или равно: «<=» ;
* равно: «==» ;
* не равно: «!=» ;

Также есть возможность изменять приоритет выполнения арифметических операций при помощи скобок:

* «(»;
* «)»;

За счет заключения операции в скобки происходит повышение его приоритета, а значит при вычислении всего выражения операция с более высоким приоритетом будет вычисляться раньше операции с меньшим приоритетом.

Приоритетность операций представлена в таблице 6.1. Чем выше число, тем выше и приоритет.

Таблица 6.1 - Приоритетность операций

|  |  |
| --- | --- |
| Операция | Приоритет |
| «(», «)» | 1 |
| «+», «-» | 2 |
| «\*», «/» | 3 |

**6.2 Польская запись и принцип ее построения**

Польская запись -форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Приоритетность операций приведена в таблице 6.1. Известен следующий принцип построения польской записи:

1. Посимвольно перебираем таблицу лексем;
2. Если текущий символ – какой-либо оператор или идентификатор, записываем его в финальную строку лексем и переходим к пункту 1;
3. Если идентификатор – функция, перед ним в финальную строку записывается токен “@”, означающий вызов функции;
4. Открывающаяся скобка автоматически заносится в стек операций;
5. Закрывающаяся скобка выталкивает все операции из стека в финальную строку и самоуничтожается вместе с открывающейся скобкой;
6. Операция выталкивает все операции с большим или равным приоритетом в финальную строку;
7. Когда встречается символ конца строки кода (“;”) – все операции выталкиваются из стека в финальную строку.
8. В конце операции преобразования к польской нотации после каждой лексемы функции добавляется лексема - число параметров, передаваемых в функцию.

**6.3 Программная реализация обработки выражений**

После этапов лексического и синтаксического анализа происходит преобразование в польскую запись, во время этапа генерации кода на язык ассемблера.

**6.4 Контрольный пример**

Контрольный пример разбора выражения содержится в таблице 6.2.

Таблица 6.2 - Разбор выражения

|  |  |  |
| --- | --- | --- |
| Выражение | Стек | Результат |
| (x + y)\*3 |  |  |
| x + y)\*3 | ( |  |
| + y)\*3 | ( | x |
| y)\*3 | ( + | x |
| )\*3 | ( + | x y |
| \*3 |  | x y+ |
| 3 | \* | x y + |
|  | \* | x y + 3 |
|  |  | x y + 3 \* |

# Глава 7. Генерация кода

## 7.1 Структура генератора кода

Заключительным этапом трансляции языка BED-2020 является генерация кода. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. Выходной файл на языке ассемблера, который будет являться результатом работы транслятора, в соответствие с таблицей лексем. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода BED-2020 представлена на рисунке 7.1.



Рисунок 7.1 – Структура генератора кода

## 7.2 Представление типов данных в памяти

Элементы таблицы идентификаторов расположены сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке BED-2020 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1–Соответствия типов идентификаторов языка MAA-2018и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке BED-2020 | Тип идентификатора на языке ассемблера | Пояснение |
| int | SDWORD | Хранит целочисленный тип данных со знаком. |
| uint | DWORD | Хранит целочисленный тип данных без знака. |
| string | BYTE – для строковых литералов  DWORD – для переменных | Литерал хранит последовательность байтов.  Переменные хранят указатель на начало строки литерала. |

## 7.3 Особенности алгоритма генерации кода

В языке BED-2020 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2

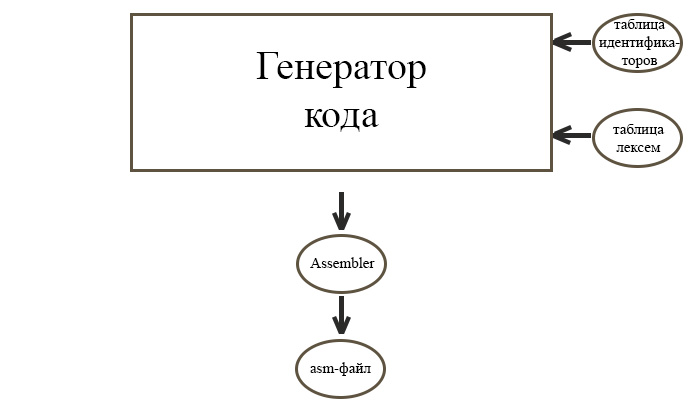


Рисунок 7.2 - Структура генератора кода

Сгенерированный код можно посмотреть в приложении Е.

# 8. Тестирование транслятора

На этапе трансляции могут возникнуть ошибки разного вида. Далее будет приведен перечень ошибок на разных этапах трансляции в качестве примера. Код будет с заранее допущенными ошибками.

Тестирование ошибок на этапе лексического анализатора представлено в листинге 9.1 и рисунке 9.1.

|  |
| --- |
| main  {  declare integer h;  h = 3;  declare integer h;  return h;  }; |

Листинг 9.1 – Код с лексической ошибкой

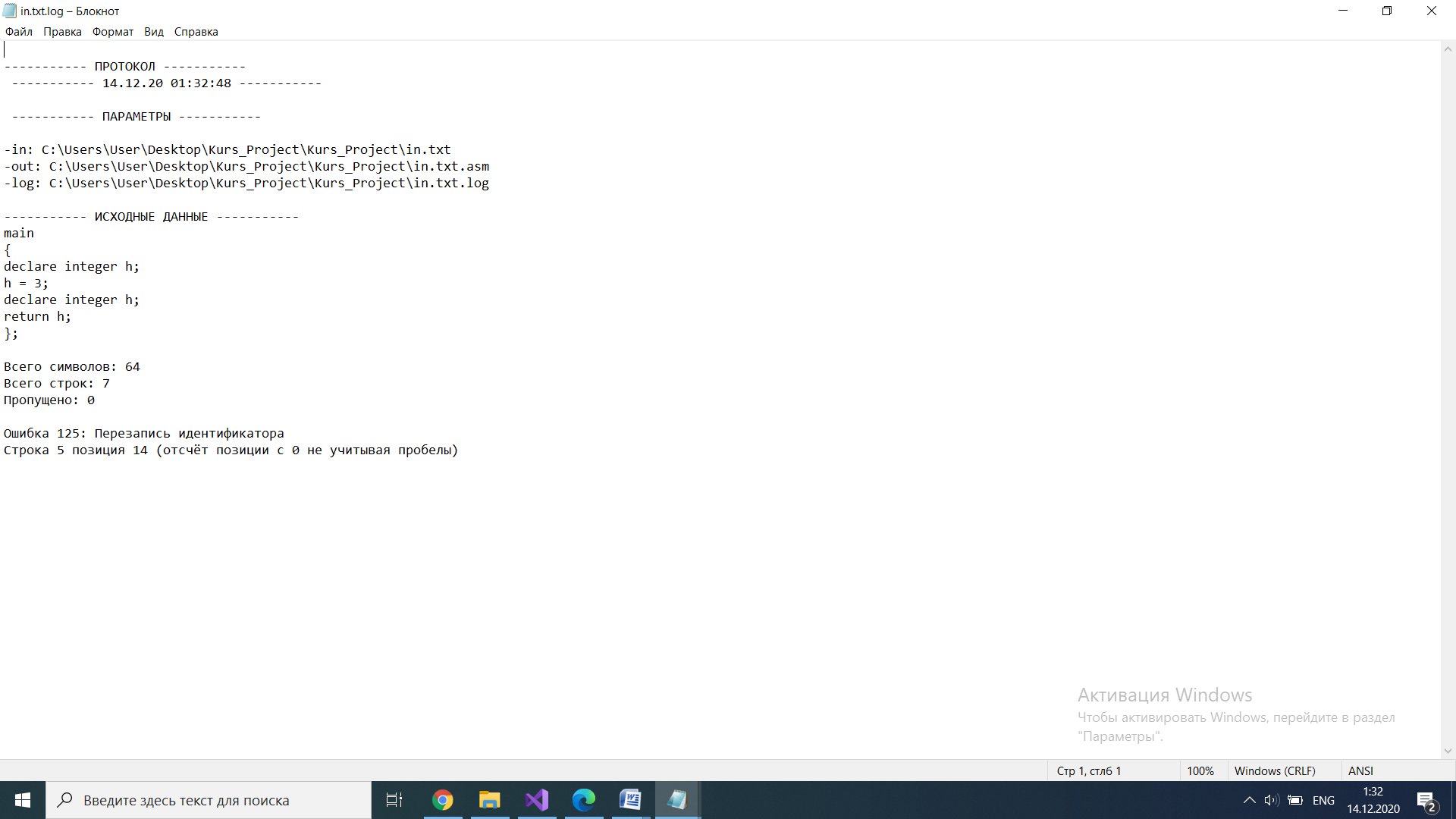


Рисунок 9.1 – Сообщение о допущенной лексической ошибке

Тестирование ошибок на этапе семантического анализатора представлено в листинге 9.2. и на рисунке 9.2.

|  |
| --- |
| main  {  declare integer x;  x = 'hello!';  return 0;  }; |

Листинг 9.2 – Код с семантической ошибкой

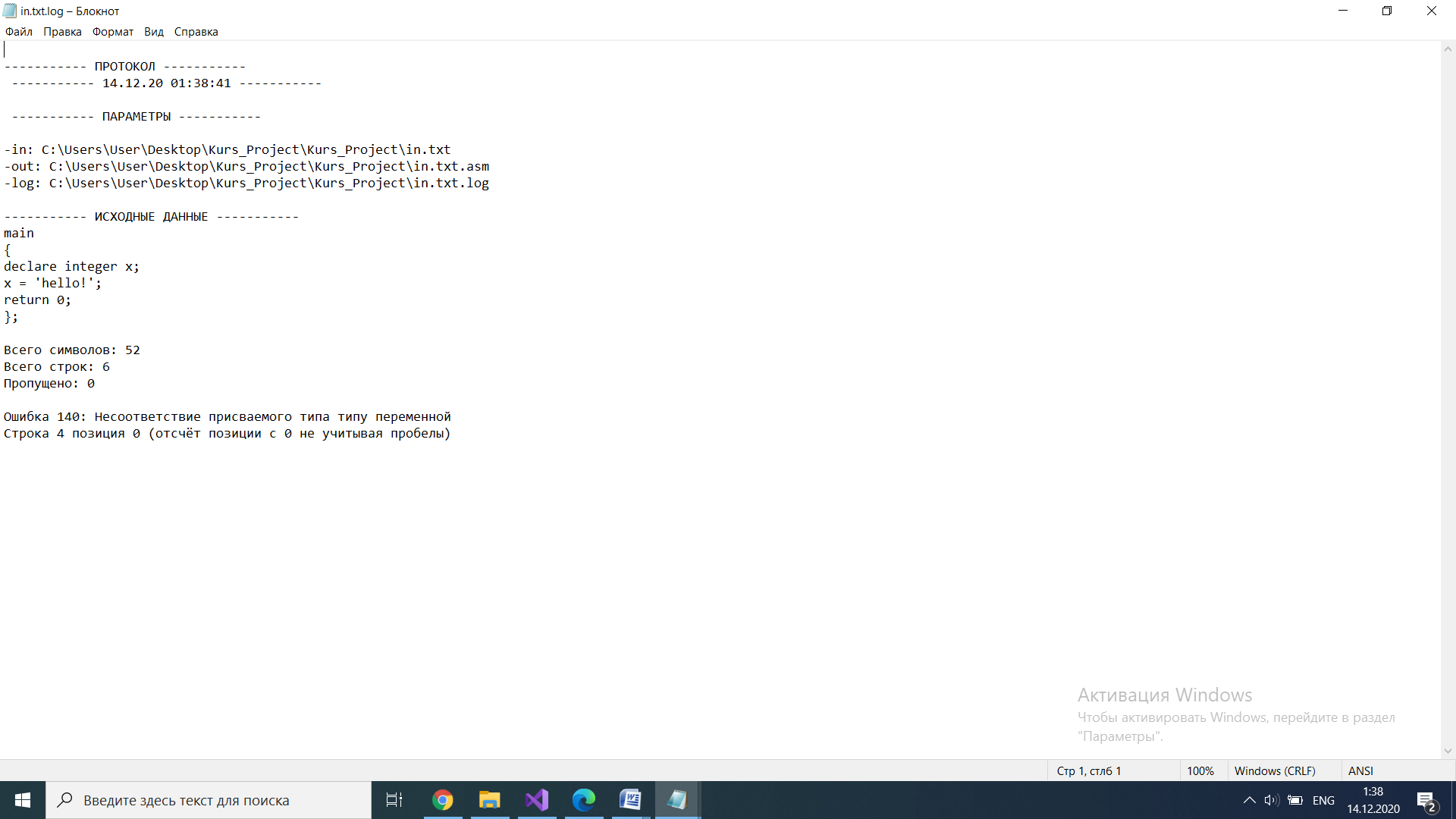


Рисунок 9.2 – Сообщение о допущенной семантической ошибке

Тестирование ошибок на этапе синтаксического анализатора представлено в листинге 9.3. и на рисунке 9.3.

|  |
| --- |
| declare integer function one(integer y)  {  return y;  };  main  {  declare integer x = 5;  declare integer z;  one(x) = 5;  return 0;  }; |

Листинг 9.3. – Код с синтаксической ошибкой

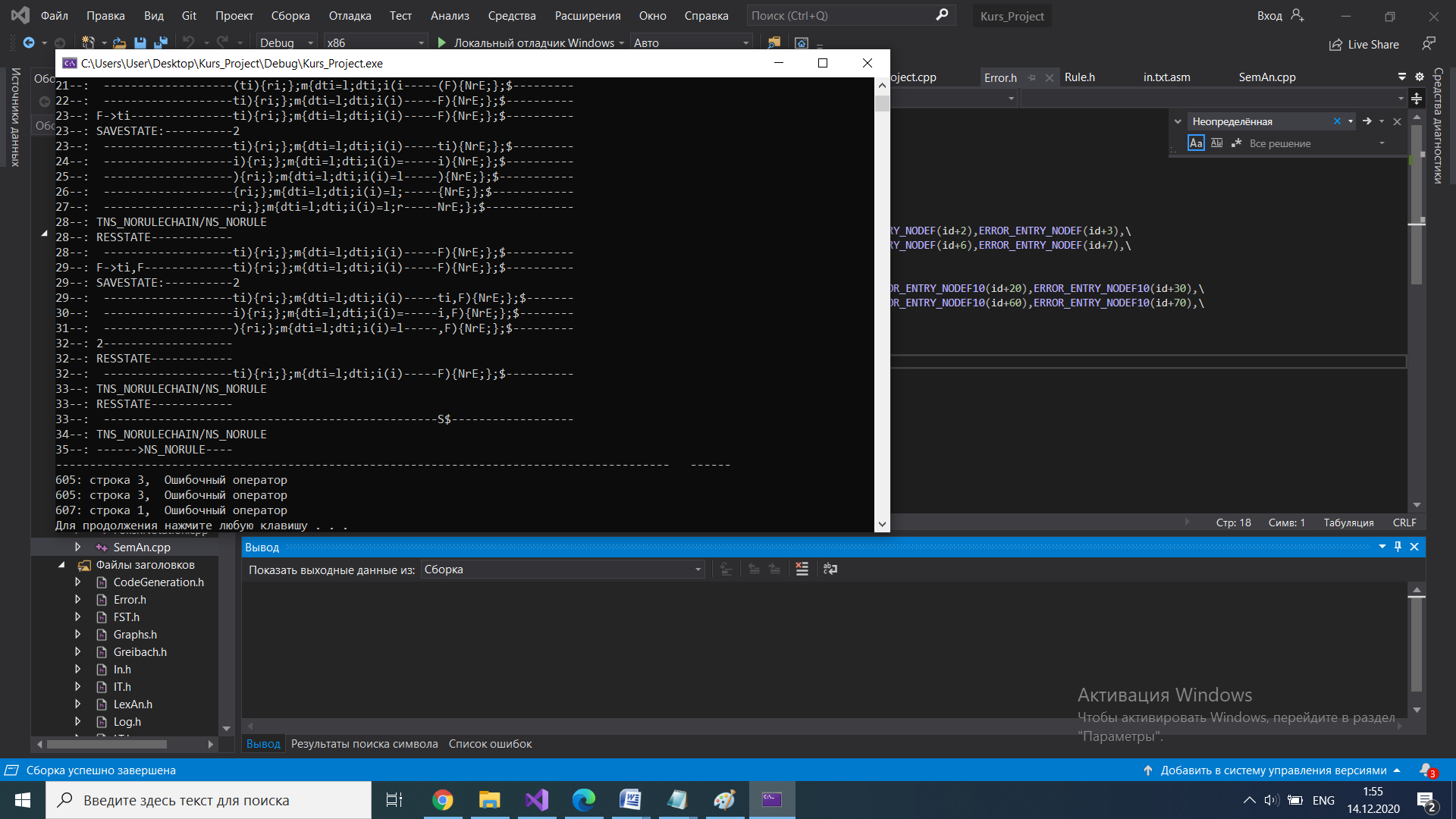
**

Рисунок 9.3. – Сообщение о допущенной синтаксической ошибке

## Заключение

Данный курсовой проект дал возможность на изучение языка программирования как самостоятельно, так и с помощью методических указаний и остального предоставленного материала. Реализован минимум для курсового проекта. Этот минимум дополнен некоторыми другими структурными элементами.

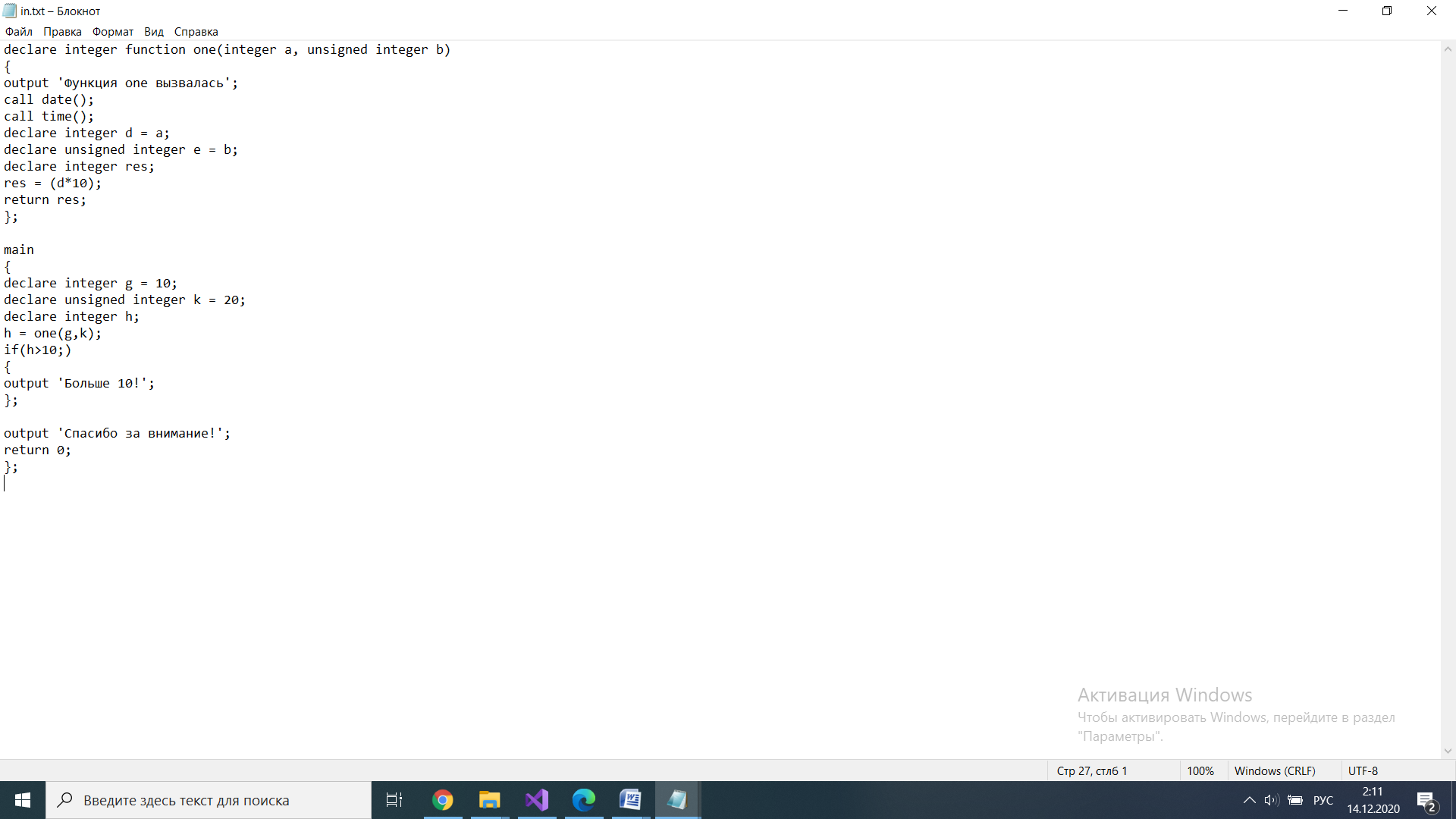
Окончательная версия языка BED-2020 включает:

* 3 типа данных;
* Реализация операции вывода output;
* Реализация функций стандартной библиотеки (date/time);
* Наличие 4 арифметических операторов для вычисления выражений;
* Реализация условного оператора;
* Реализована система обработки ошибок.
* Более 3000 строк кода
* Возможность инициализации при объявлении переменной
* Наличие 6 операторов сравнения

Полученные знания при выполнении курсового проекта будут способствовать последующему изучению новых технологий, так как изучение новых языков программирования уже будет проходить на уровне понимания работы языка, а не просто изучение синтаксиса.

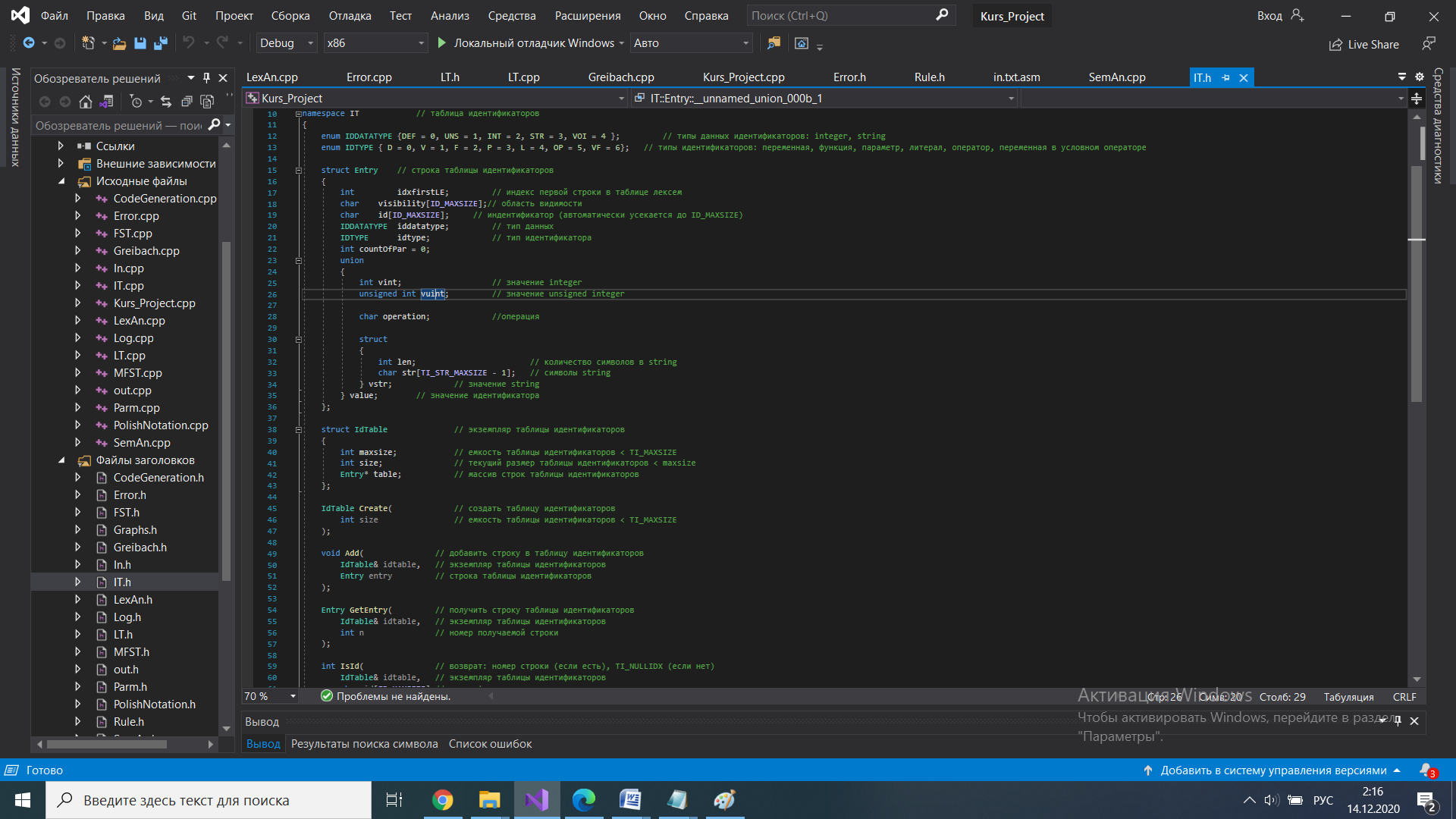
# Приложение А

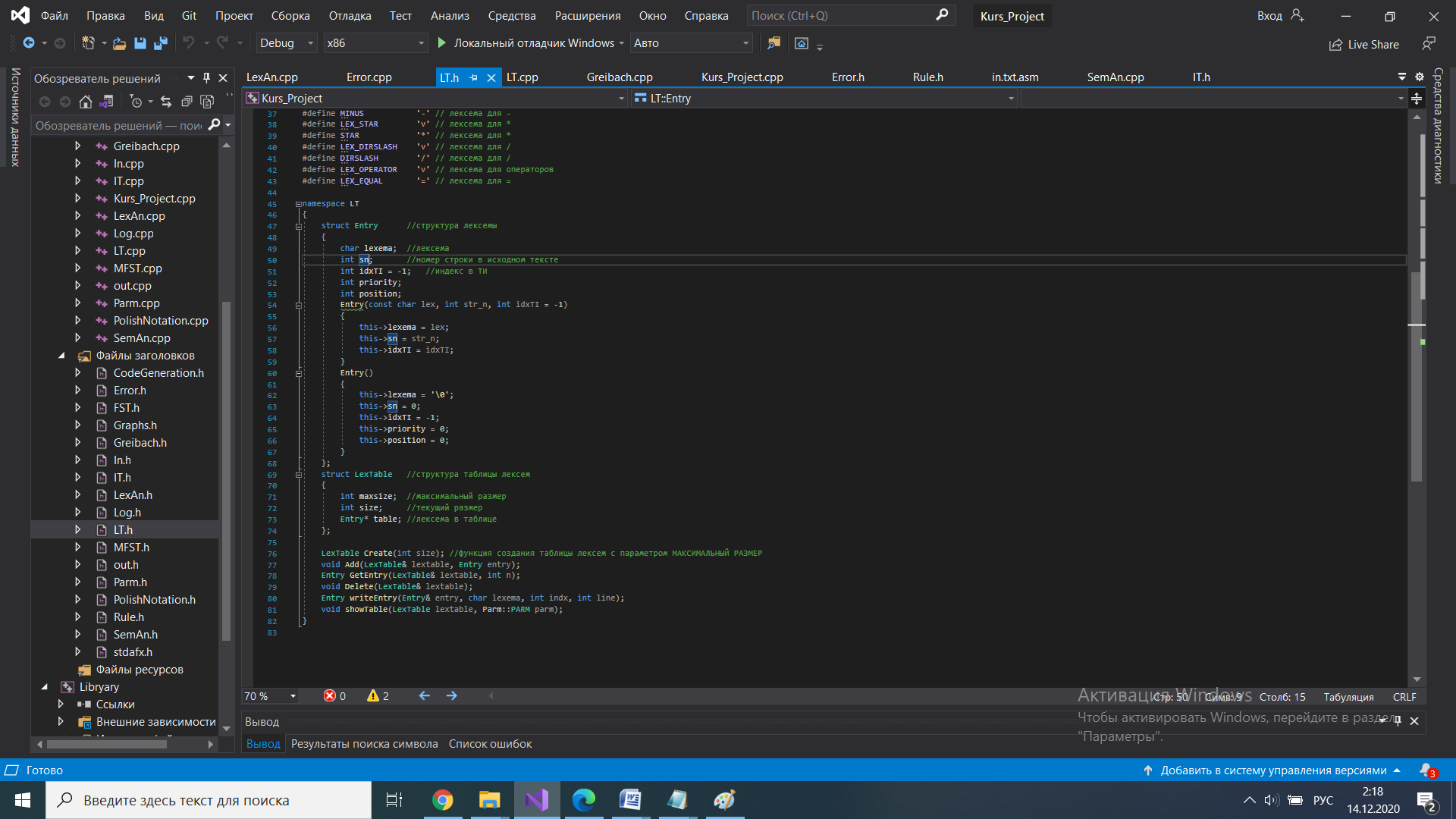
Исходный код:



# Приложение Б

Структуры данных, использованные на этапе лексического анализа





# Приложение В

Таблица лексем:

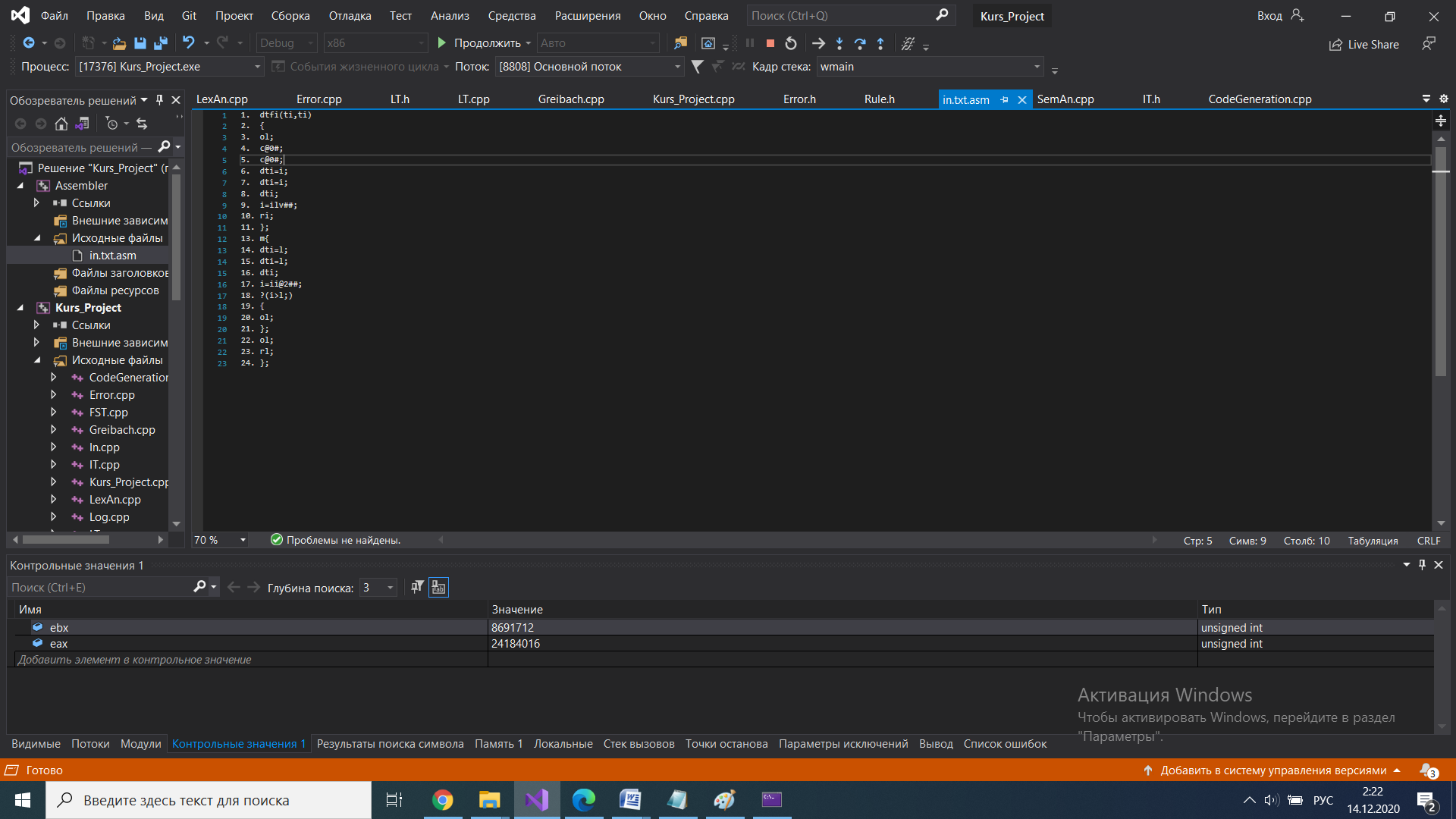
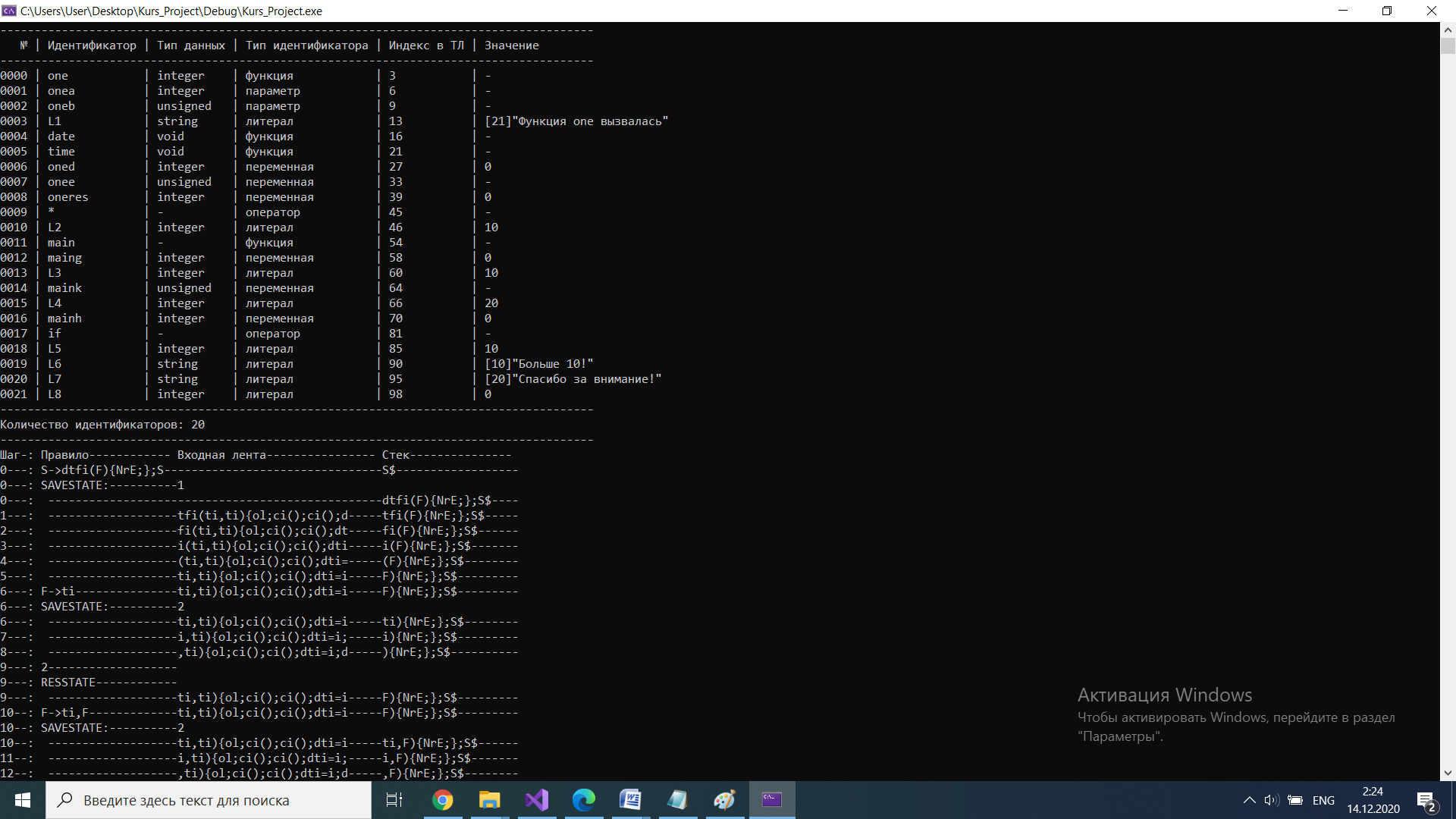
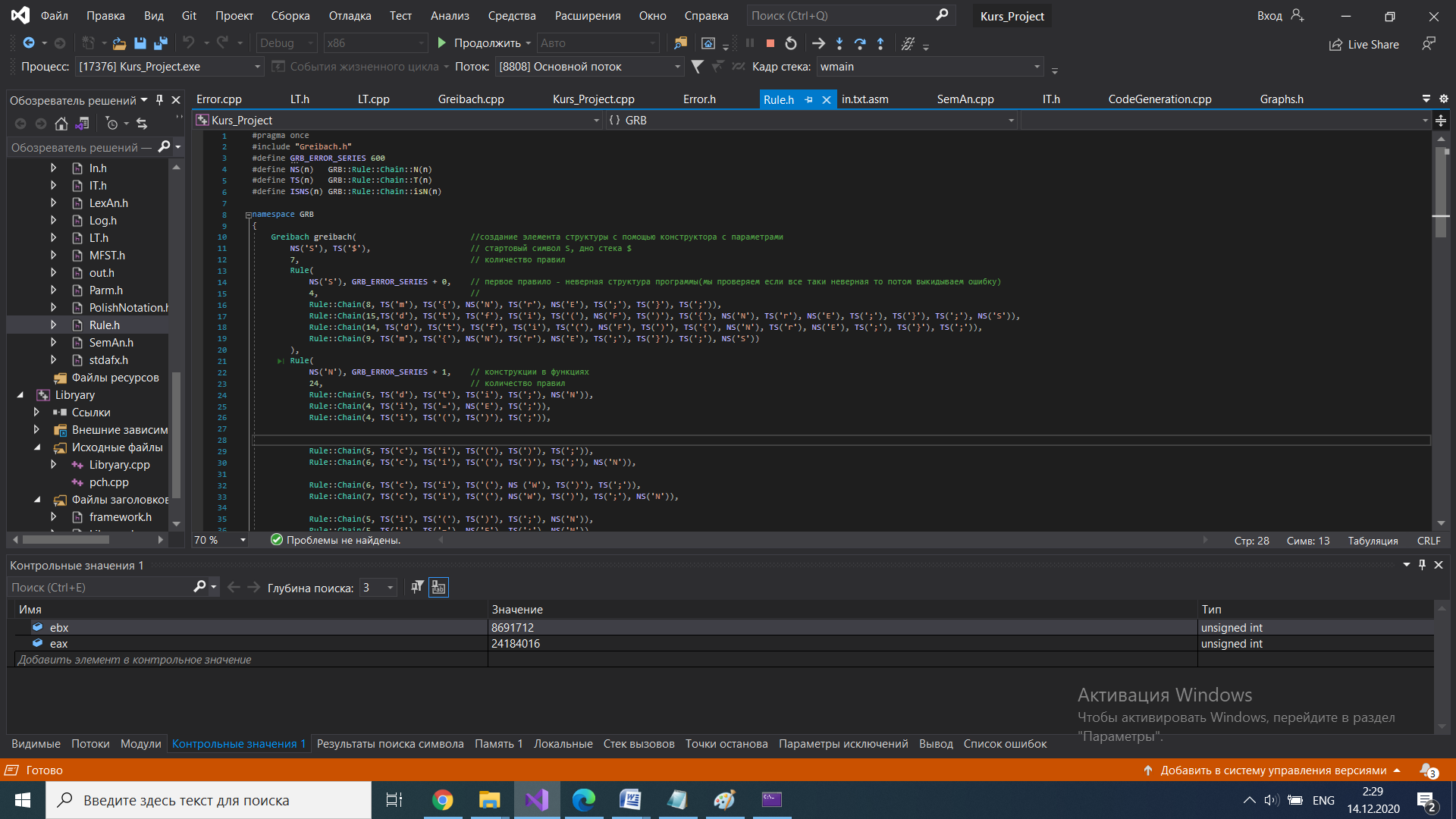


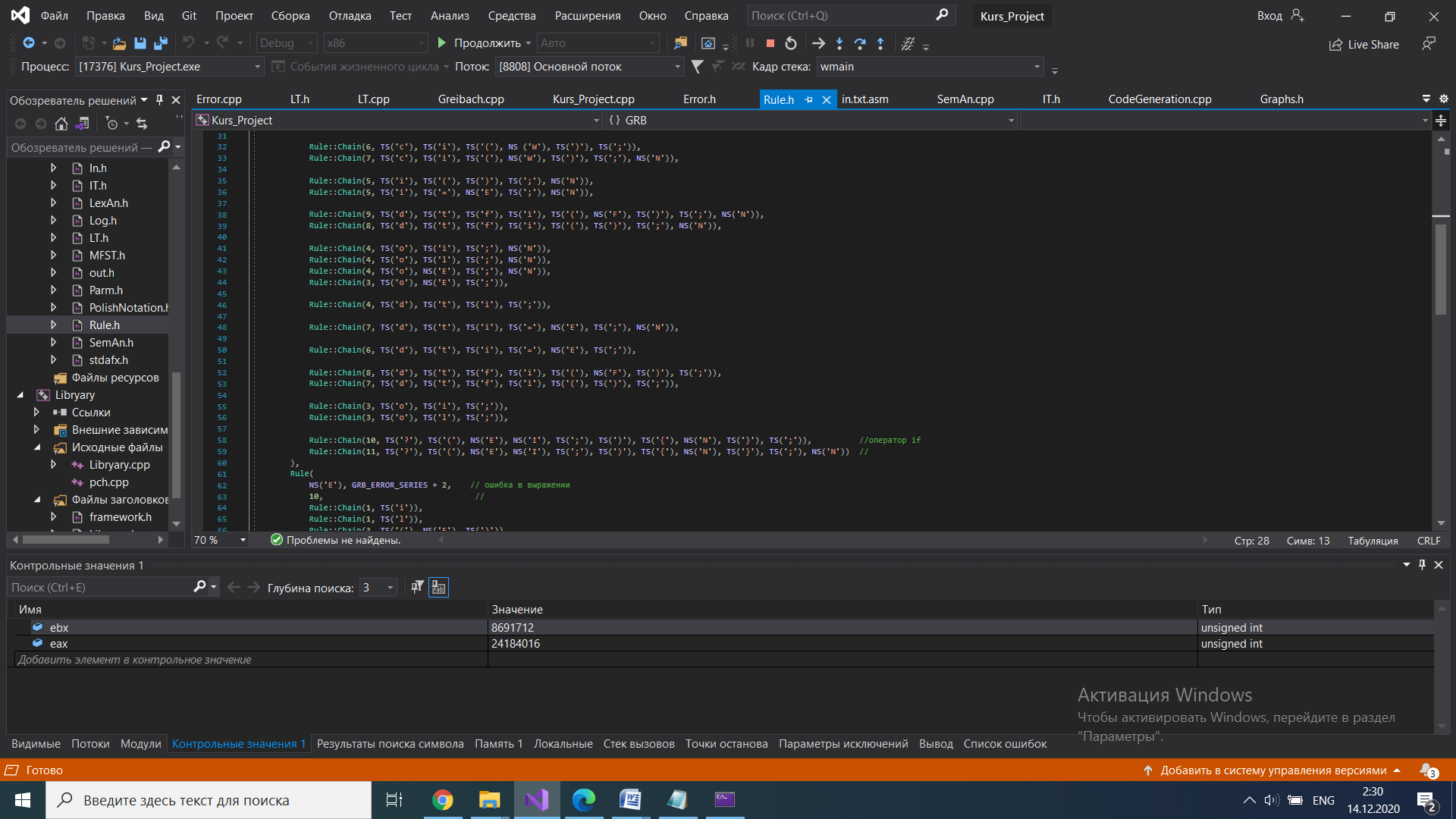
Таблица идентификаторов:

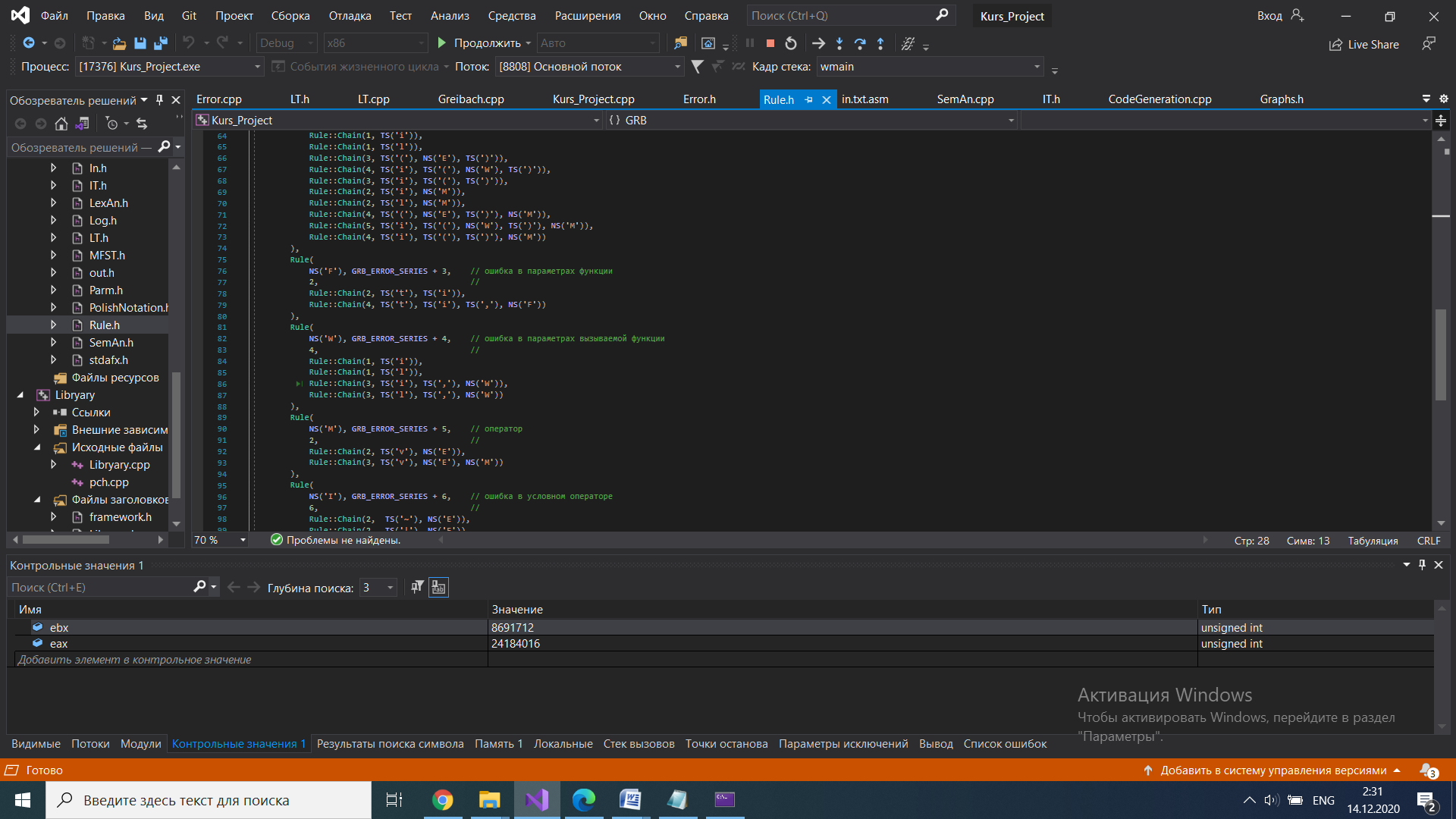


# Приложение Г

Структура данных грамматики Грейбах

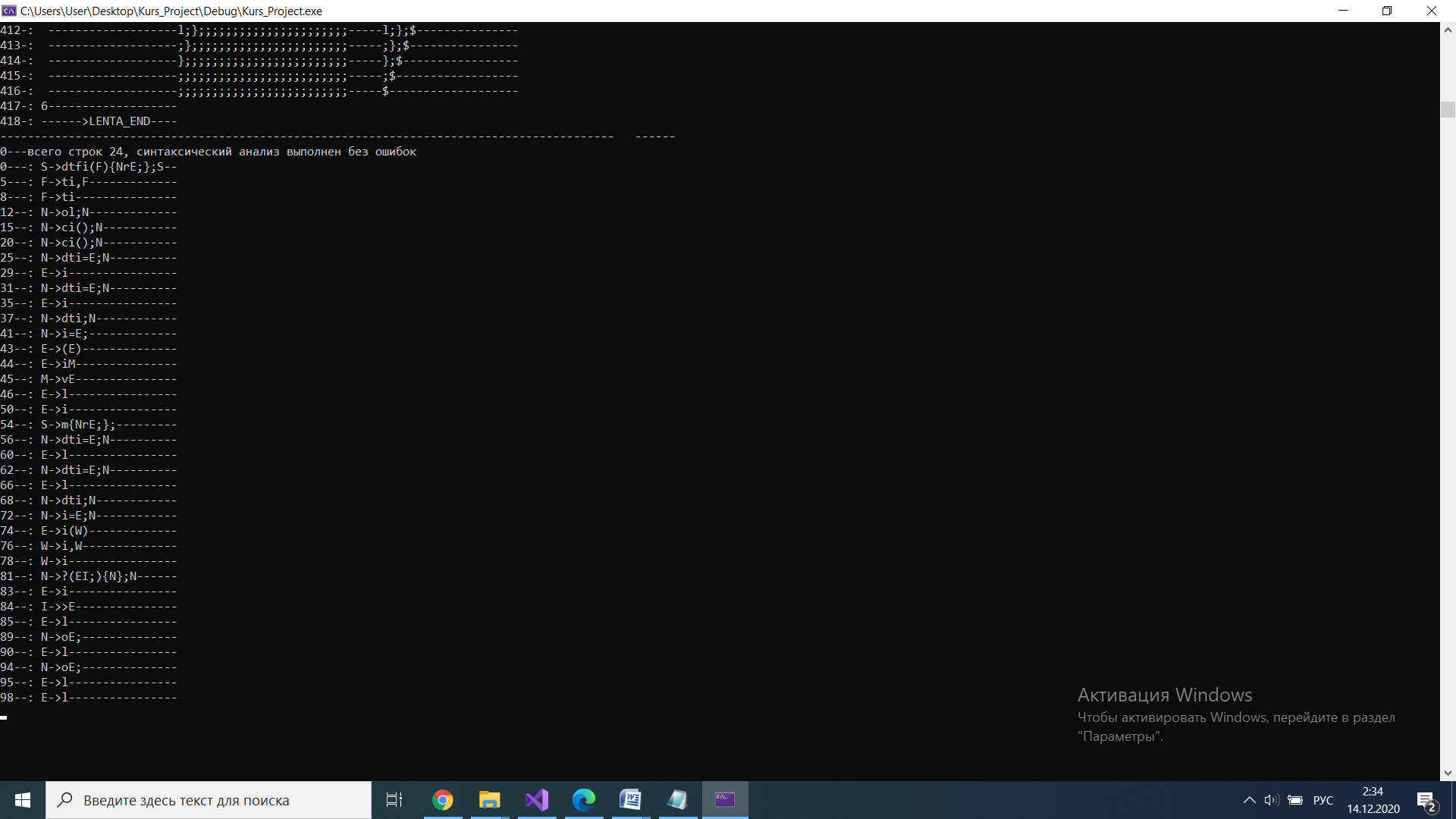




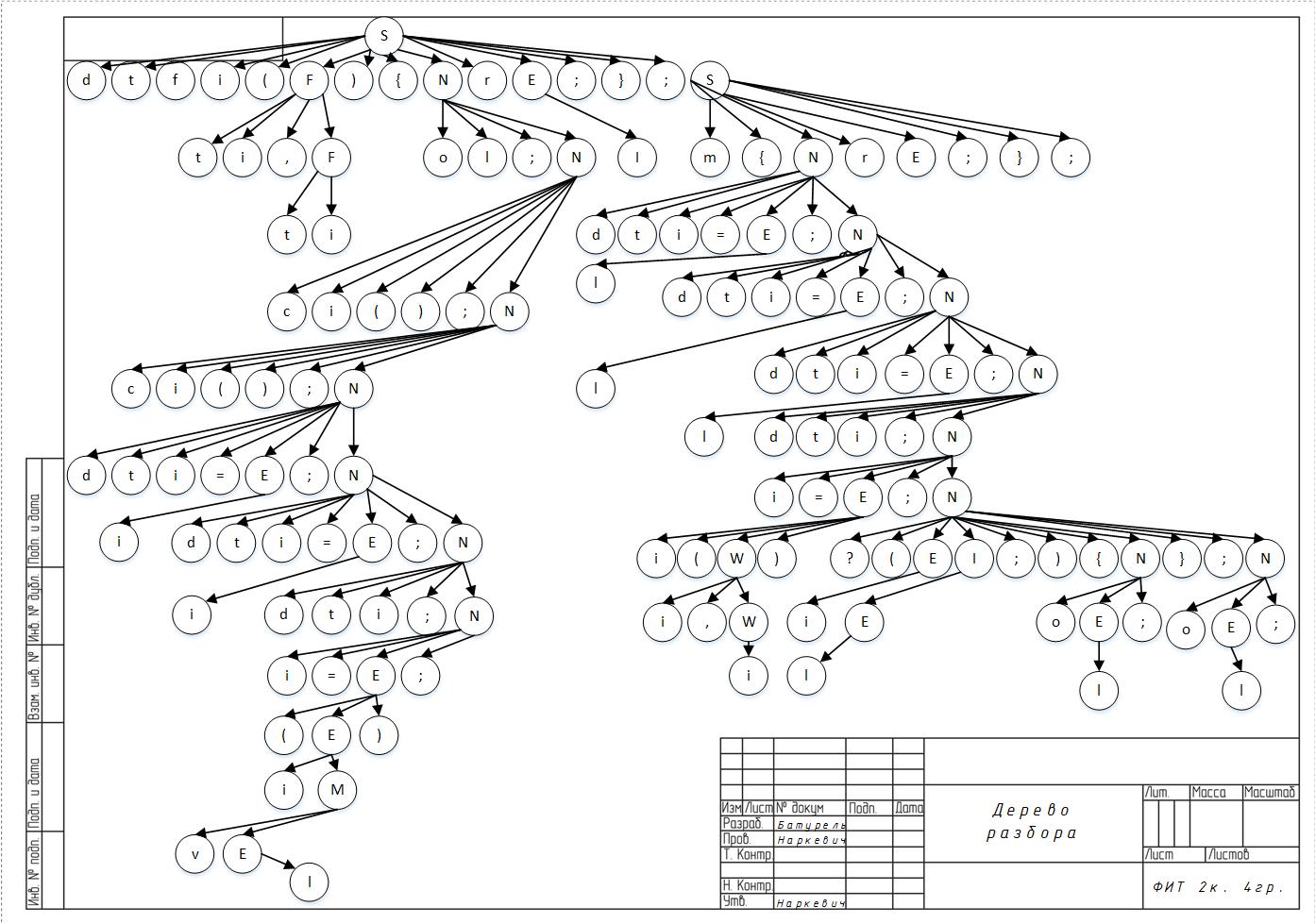


# Приложение Д

Дерево разбора

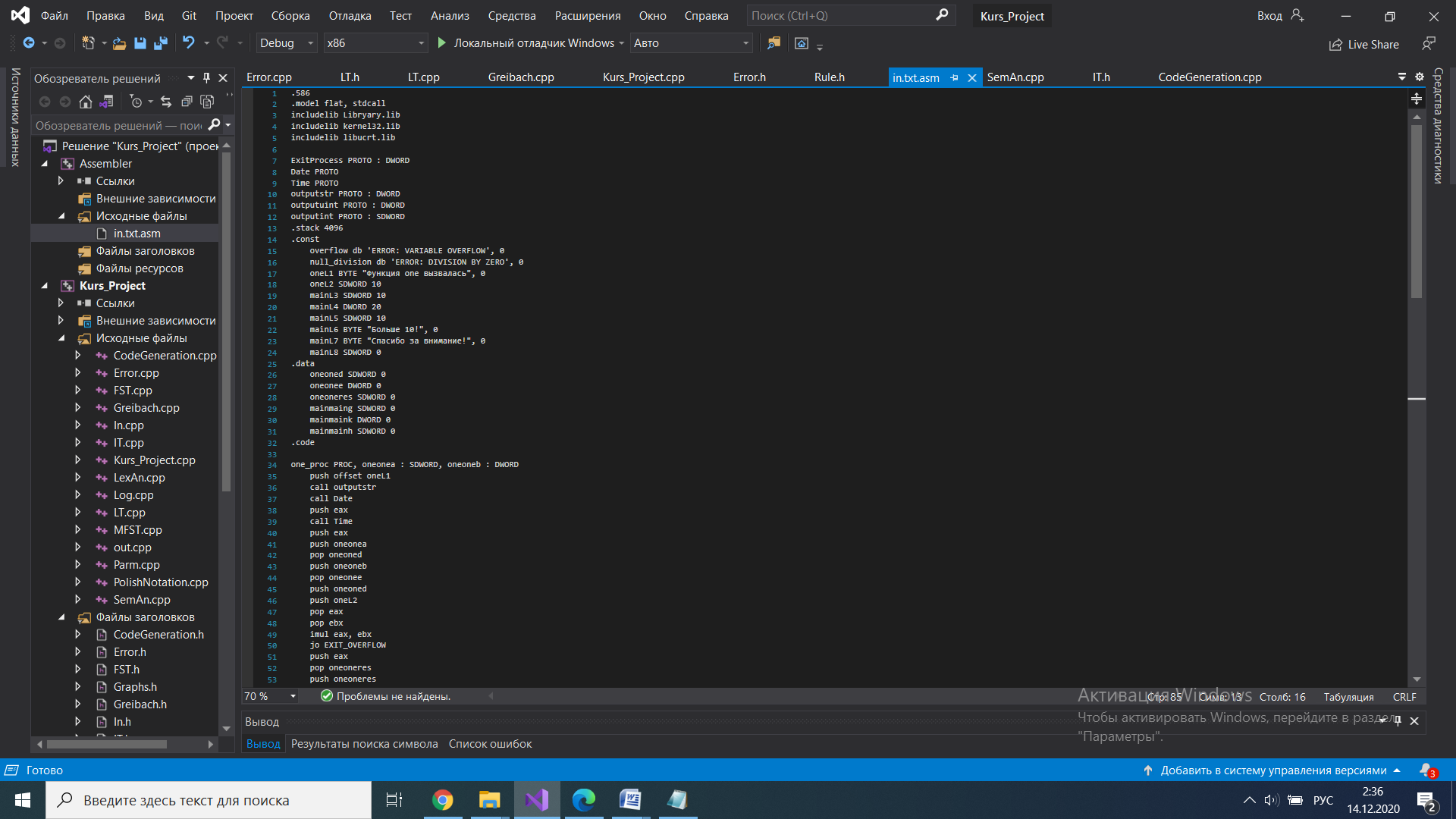


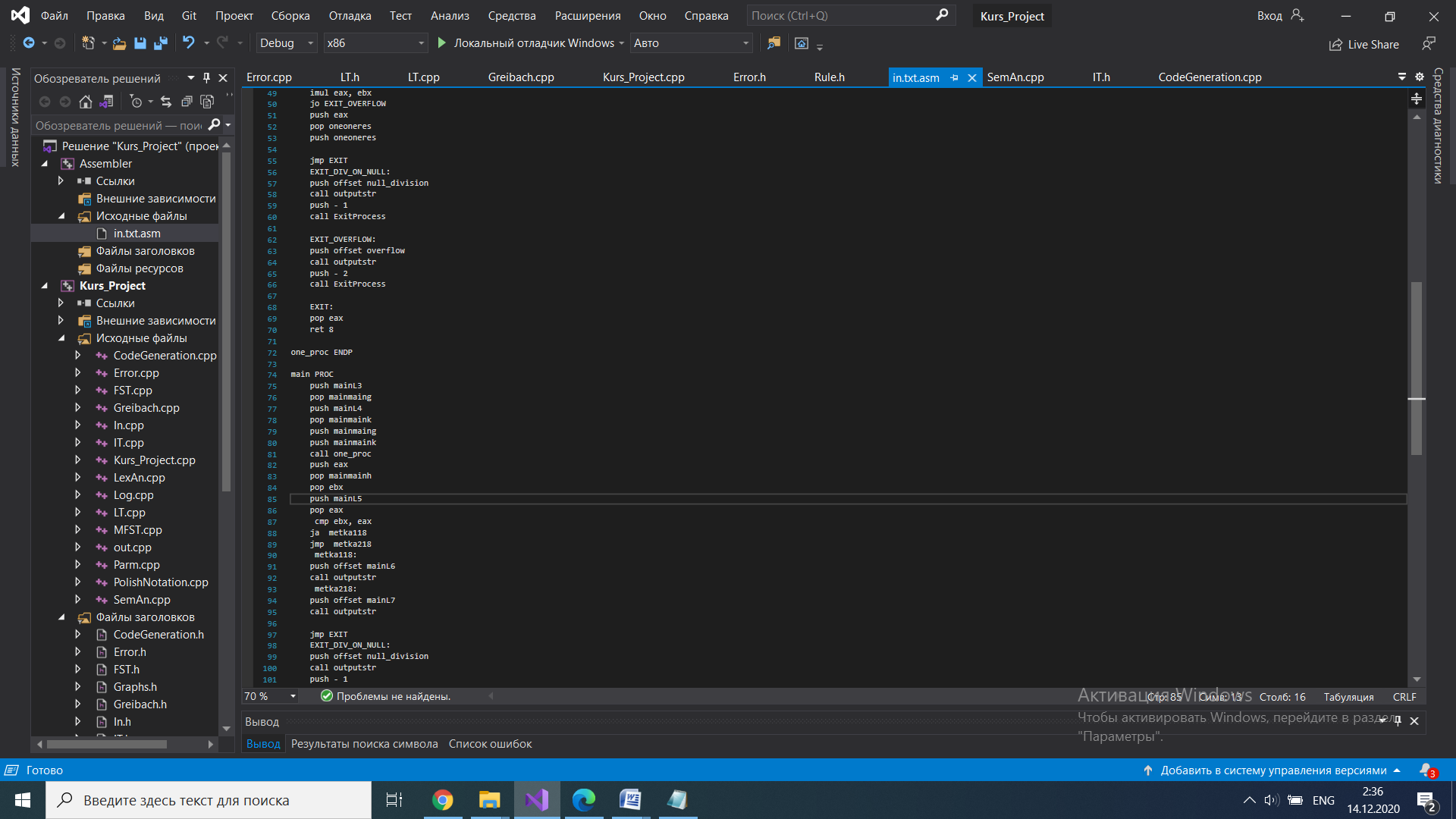
Граф дерева синтаксического разбора

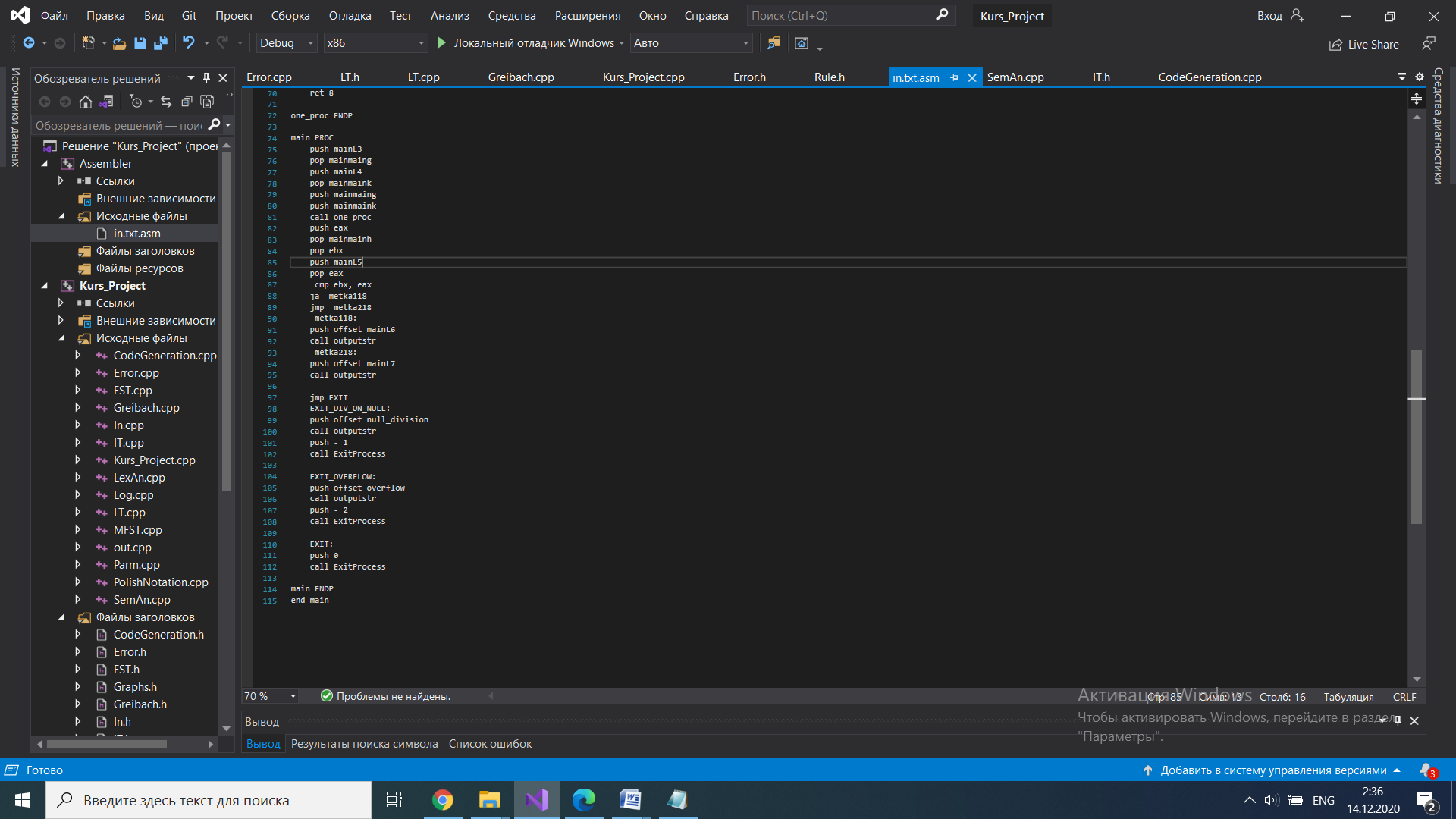


# Приложение Е

Результат генерации кода







**Литература**

1. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.  
2. Ахо, А. Теория синтаксического анализа, перевода и компиляции /А. Ахо, Дж. Ульман. – Москва : Мир, 1998. – Т. 2 : Компиляция. - 487 с.  
3. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.  
4. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.  
5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с