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 SII - 2020»

Выполнил студент Скородумов Иван Иванович

(Ф.И.О.)

Руководитель проекта пр.ст. Пахолко Алёна Степановна

(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 пр.ст. Пахолко Алёна Степановна

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

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

Нормоконтролер пр.ст. Пахолко Алёна Степановна

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

Курсовой проект защищен с оценкой

Минск 2020

**Содержание**

[Глава 1. Спецификация языка программирования 5](#_Toc501385916)

[1.1. Характеристика языка программирования 5](#_Toc501385917)

[1.2. Алфавит языка 5](#_Toc501385918)

[1.3. Символы сепараторы 6](#_Toc501385919)

[1.4. Применяемые кодировки 6](#_Toc501385920)

[1.5. Типы данных 6](#_Toc501385921)

[1.6. Преобразование типов данных 7](#_Toc501385922)

[1.7. Идентификаторы 7](#_Toc501385923)

[1.8. Литералы 7](#_Toc501385924)

[1.9. Область видимости идентификаторов 8](#_Toc501385925)

[1.10. Инициализация данных 8](#_Toc501385926)

[1.11. Инструкции языка 9](#_Toc501385927)

[1.12. Операции языка 9](#_Toc501385928)

[1.13. Выражения и их вычисления 10](#_Toc501385929)

[1.14. Программные конструкции языка 10](#_Toc501385930)

[1.15. Область видимости 11](#_Toc501385931)

[1.16. Семантические проверки 11](#_Toc501385932)

[1.17. Распределение оперативной памяти на этапе выполнения 11](#_Toc501385933)

[1.18. Стандартная библиотека и её состав 11](#_Toc501385934)

[1.19. Ввод и вывод данных 12](#_Toc501385935)

[1.20. Точка входа 12](#_Toc501385936)

[1.21. Препроцессор 13](#_Toc501385937)

[1.22. Соглашения о вызовах 13](#_Toc501385938)

[1.23. Объектный код 13](#_Toc501385939)

[1.24. Классификация сообщений транслятора 13](#_Toc501385940)

[1.25. Контрольный пример 13](#_Toc501385941)

[Глава 2. Структура транслятора 13](#_Toc501385942)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 14](#_Toc501385943)

[2.2 Перечень входных параметров транслятора 15](#_Toc501385944)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 15](#_Toc501385945)

[Глава 3. Разработка лексического анализатора 16](#_Toc501385946)

[3.1 Структура лексического анализатора 16](#_Toc501385947)

[3.2 Контроль входных символов 16](#_Toc501385948)

[3.3 Удаление избыточных символов 17](#_Toc501385949)

[3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов 17](#_Toc501385950)

[3.5 Основные структуры данных 18](#_Toc501385951)

[3.6 Принцип обработки ошибок 18](#_Toc501385952)

[3.7 Структура и перечень сообщений лексического анализатора 19](#_Toc501385953)

[3.8 Параметры лексического анализатора и режимы его работы 19](#_Toc501385954)

[3.9 Алгоритм лексического анализа 19](#_Toc501385955)

[3.10 Контрольный пример 20](#_Toc501385956)

[Глава 4. Разработка синтаксического анализатора 21](#_Toc501385957)

[4.1 Структура синтаксического анализатора 21](#_Toc501385958)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 21](#_Toc501385959)

[4.3 Построение конечного магазинного автомата 23](#_Toc501385960)

[4.4 Основные структуры данных 24](#_Toc501385961)

[4.5 Описание алгоритма синтаксического разбора 24](#_Toc501385962)

[4.6 Структура и перечень сообщений синтаксического анализатора 24](#_Toc501385963)

[4.7 Параметры синтаксического анализатора и режимы его работы 25](#_Toc501385964)

[4.8 Принцип обработки ошибок 25](#_Toc501385965)

[4.9 Контрольный пример 25](#_Toc501385966)

[Глава 5. Разработка семантического анализатора 26](#_Toc501385967)

[5.1 Структура семантического анализатора 26](#_Toc501385968)

[5.2 Функции семантического анализатора 26](#_Toc501385969)

[5.3 Структура и перечень сообщений семантического анализатора 26](#_Toc501385970)

[5.4 Принцип обработки ошибок 27](#_Toc501385971)

[5.5 Контрольный пример 27](#_Toc501385972)

[Глава 6. Преобразование выражений 27](#_Toc501385973)

[6.1 Выражения, допускаемые языком 28](#_Toc501385974)

[6.2 Польская запись 28](#_Toc501385975)

[6.3 Программная реализация обработки выражений 29](#_Toc501385976)

[6.4 Контрольный пример 29](#_Toc501385977)

[Глава 7. Генерация кода 30](#_Toc501385978)

[7.1 Структура генератора кода 30](#_Toc501385979)

[7.2 Представление типов данных в оперативной памяти 30](#_Toc501385980)

[7.3 Алгоритм работы генератора кода 31](#_Toc501385981)

[Глава 8. Тестирование транслятора 33](#_Toc501385982)

[8.1 Тестирование фазы проверки на допустимость символов 33](#_Toc501385983)

[8.2 Тестирование лексического анализатора 33](#_Toc501385984)

[8.3 Тестирование синтаксического анализатора 34](#_Toc501385985)

[8.4 Тестирование семантического анализатора 34](#_Toc501385986)

[Приложения 38](#_Toc501385987)

[Контрольный пример 38](#_Toc501385988)

[Приложение А 39](#_Toc501385989)

[Приложение В 49](#_Toc501385990)

[Приложение Г 51](#_Toc501385991)

[Приложение Е 55](#_Toc501385992)

[Литература 37](#_Toc501385993)

**Введение**

Целью курсового проекта поставлена задача разработки компилятора для моего языка программирования SII-2020. Этот язык программирования предназначен для выполнения простейших операций и арифметических действий над числами.

Компилятор SII-2020 – это программа, задачей которого является перевод программы, написанной на языке программирования SII-2020 в программу на язык ассемблера.

Транслятор SII-2020 состоит из следующих частей:

– семантический анализатор;

– синтаксический анализатор;

– логический анализатор;

– генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разбратка структуры транслятора;

– разработка лексического и семантического анализаторов;

– разработка синтаксического анализатора;

– преобразование выражений;

– генерация кода на язык ассемблера;

– тестирование транслятора.

Решения каждой из поставленных задач будут приведены в соответствующих главах курсового проекта.

# **Глава 1. Спецификация языка программирования**

## **Характеристика языка программирования**

Язык программирования SII-2020 классифицируется как процедурный, универсальный, строготипизированный, компилируемый и не объектно-ориентированный язык.

## **Алфавит языка**

Алфавит языка SII-2020 основан на кодировке Windows-1251, представленной на рисунке 1.1.



Рисунок 1.1 – Алфавит входных символов

На этапе выполнения могут использоваться символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, спецсимволы, а также непечатные символы пробела, табуляции и перевода строки. Русские символы разрешены только в строковых литералах.

## **Символы сепараторы**

Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |  |
| --- | --- | --- |
| Сепаратор | Название | Область применения |
| ‘ ‘ | Пробел | Допускается везде, кроме идентификаторов и ключевых слов |
| ; | Точка с запятой | Разделение конструкций |
| {…} | Фигурные скобки | Заключение программного блока |
| […] | Квадратные кавычки | Блок кода |
| (…) | Круглые скобки | Приоритет операций, параметры функции |
| “…” | Двойные кавычки | Строковый литерал |
| ‘…’ | Одинарные кавычки | Допускается везде, кроме идентификаторов и ключевых слов |
| = | Знак «равно» | Присваивание значения |
| , | Запятая | Разделение параметров |
| /  \  +  -  \*  : | Знаки «косая черта», «обратная косая черта», «плюс», «астерикс», «тильда», | Выражения |
| &  ^  <  > | «амперсанд», «циркумфлекс», знаки «больше» и «меньше» | Выражения в операторе цикла |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования SII-2020 используется кодировка Windows-1251.

## **Типы данных**

В языке SII-2020 реализованы три типа данных: целочисленный беззнаковый, символьный, строковый. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка SII-2020

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Символьный тип данных char | Фундаментальный тип данных. Используется для работы с символом, который в памяти занимает 1 байт. Инициализация по умолчанию: символ конца строки “\0”. |

Окончание таблицы 1.2

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Целочисленный тип данных uint | Фундаментальный тип данных. Используется для работы с целочисленными положительными значениями. В памяти занимает 4 байта.  Максимальное значение: 4294967295. Минимальное значение: 0.  Инициализация по умолчанию: значение 0.  При попытке инициализации значением больше максимального, инициализируется максимальным. |
| Строковый тип данных str | Фундаментальный тип данных. Используется для работы с набором символов, каждый символ в памяти занимает 1 байт. Инициализация по умолчанию: символ конца строки “\0”. Для типа str определён оператор + – конкатенация строк. |

У каждого типа данных есть ограниченный набор доступных операторов.

## **Преобразование типов данных**

Преобразование типов данных не поддерживается, т.е. язык является строготипизированным. Но в стандартной библиотеке есть функции преобразования типа uint в тип char и наоборот.

## **Идентификаторы**

В имени идентификатора допускаются только символы латинского алфавита нижнего регистра. Максимальная длина имени идентификатора – 8 символов. Максимальная длина имени идентификатора функции – 8 символов. При вводе идентификатора длиной более разрешенного количества символов, он будет усекаться. Имя идентификатора не может совпадать с ключевыми словами, уже содержащаяся в стандартной библиотеке.

## **Литералы**

С помощью литералов осуществляется инициализация переменных. В языке существует два типа литералов. Краткое описание литералов языка SII-2020 представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |  |  |
| --- | --- | --- | --- |
| Тип литерала | Регулярное выражение | Описание | Пример |
| Строковый литерал | [a-z|A-Z|А-Я|а-я|0-9|!-/]\* | Строковые литералы, максимальная длина строки 255 символов | output “aaa” |

Окончание таблицы 1.3

|  |  |  |  |
| --- | --- | --- | --- |
| Тип литерала | Регулярное выражение | Описание | Пример |
| Целочисленный литерал | [1-9]+[0-9]\* | Целочисленные неотрицательные литералы, по умолчанию инициализируются 0. Литералы могут быть только rvalue. | bind uint sum = 15;  15 – целочисленный литерал. |
| Символьный литерал | [a-z|A-Z|А-Я|а-я|0-9|!-/] | Символ, заключённый в ‘’ (одинарные кавычки), по умолчанию инициализируются пустой строкой. Литералы могут быть только rvalue. | bind char symbol = ‘T’;  T – символьный литерал. |

Литералы являются константами и при генерации кода объявляются один раз.

## **Область видимости идентификаторов**

Область видимости «сверху вниз» (по принципу С++). В языке SII-2020 требуется обязательное объявление переменной перед её инициализацией и последующим использованием. Все переменные должны находиться внутри программного блока. Имеется возможность объявления одинаковых переменных в разных блоках, т. к. переменные, объявленные в одной функции, недоступны в другой. Каждая переменная получает префикс – название функции, в которой она объявлена. Объявление функций стандартной библиотеки можно производить в любом месте кода.

## **Инициализация данных**

При объявлении переменной не допускается инициализация данных. Краткое описание способов инициализации переменных языка SII-2020 представлено в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

|  |  |  |
| --- | --- | --- |
| Конструкция | Описание | Пример |
| bind <тип данных> <идентификатор>; | Автоматическая инициализация: переменные типа uint инициализируются нулём, переменные типа char – пустым символом. | bind uint sum;  bind symbol chr; |
| <идентификатор> = <значение>; | Присваивание переменной значения. | sum = 15;  chr = ‘S’; |

Соответствие типов проверяется на синтаксическом анализе.

## **Инструкции языка**

Все возможные инструкции языка программирования SII-2020 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования SII-2020

|  |  |
| --- | --- |
| Инструкция | Запись на языке SII-2020 |
| Объявление переменной | bind <тип данных> <идентификатор>; |
| Объявление функции | bind <тип данных> func <идентификатор> (<тип данных> <идентификатор>, …) {<блок кода>}; |
| Присваивание | <идентификатор> = <значение>/<идентификатор>; |
| Объявление внешней функции | external <тип данных> func <идентификатор> (<тип данных> <идентификатор>, …); |
| Блок инструкций | go  {  …  } |
| Возврат из подпрограммы | stop <идентификатор> / <литерал>; |
| Условная инструкция | while(<условие>)[<блок кода>]; |
| Вывод данных | output <идентификатор> / <литерал>; |
| Однострочный комментарий до конца строки | #<любой текст> |

Все инструкции кроме функции входа в программу требуют закрывающую ;

* 1. **Операции языка**

Язык программирования SII-2020 может выполнять операции, представленные в таблице 1.6. Операция сдвига учитывает только первый младший бит оператора, т.к. сдвиг более чем на 255 любого числа кроме нуля вернет число большее, чем можно разместить в типе данных uint.

Таблица 1.6 – Операции языка программирования SII‑2020

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Примечание | Типы данных | Пример |
| = | Присваивание | (uint, uint)  (char, char)  (str, str) | sum = 15;  chr = ‘T’; |
| <,> | Знаки «больше», «меньше» для условной инструкции | (uint, uint)  (char, char) | while(sum < diff) […]; |
| & | Оператор эквивалентности | (uint, uint)  (char, char) | while(sum & diff) […]; |
| ^ | Оператор неравенства | (uint, uint)  (char, char) | while(sum ^ diff) […]; |

Окончание таблицы 1.6

|  |  |  |  |
| --- | --- | --- | --- |
| ( | Приоритет операций | - | sum = (a + b) \* c; |
| ) |
| + | Суммирование | (uint, uint)  (char, char) | sum = a + b; |
| - | Вычитание | (uint, uint) | diff = a – b; |
| \* | Умножение | (uint, uint) | mul = a\*b; |
| : | Деление | (uint, uint) | div = a:b; |
| % | Остаток от деления | (uint, uint) | mod = a%b; |
| / | Сдвиг влево | (uint, uint) | pr = a / b; |
| \ | Сдвиг вправо | (uint, uint) | pr = a \ b; |

Т.к. отрицательные числа не поддерживаются, если результат операции меньше нуля, он вычитается из максимального значения.

## **Выражения и их вычисления**

Круглые скобки в выражении используются для изменения приоритета операций. Также не допускается запись двух подряд идущих арифметических операций. Выражение может содержать вызов функции, если эта функция уже содержится в стандартной библиотеке. Выражения вычисляются только после оператора присваивания.

## **Программные конструкции языка**

Ключевые программные конструкции языка программирования SII-2020 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка SII-2020

|  |  |
| --- | --- |
| Конструкция | Запись на языке SII-2020 |
| Главная функция (точка входа) | go  {  …  stop <идентификатор> / <литерал>;  } |
| Функция | bind <тип данных> func <идентификатор> (<тип> <идентификатор>, …)  {… stop <идентификатор> / <литерал>; }; |
| Цикл | while(a^8)[ …]; |

Принцип действия while схож с одноименным циклом в C++;

## **Область видимости**

В языке SII-2020 все переменные являются локальными, т.е. имеют функциональную область видимости. Они обязаны находится внутри программного блока функций (по принципу С++). Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

## **Семантические проверки**

Таблица с перечнем семантических проверок, предусмотренных языком, приведена в таблице 1.8.

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы не должны повторно объявляться в пределах одной функции. |
| 2 | Тип возвращаемого значения должен совпадать с типом функции при её объявлении или подключении |
| 3 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении или подключении |
| 4 | В функцию должно быть передано то число параметров, сколько ожидается |
| 5 | Тип данных результата выражения должен совпадать с типом данных идентификатора, которому оно присваивается |
| 6 | Типы данных операндов выражения должны быть одинаковыми |
| 7 | Для типа данных str доступна только операция сложения (конкатенация срок) |
| 8 | Тип данных str не может быть аргументом условной конструкции |
| 9 | Для типа char определены только операции + и - |
| 10 | Функции не должны подключаться дважды в пределах одной программы |

Если семантическая проверка не проходит, то в лог журнал записывается соответствующая ошибка.

## **Распределение оперативной памяти на этапе выполнения**

Все переменные размещаются в куче.

## **Стандартная библиотека и её состав**

Стандартная библиотека SII-2020 написана на языке программирования C++.

Для использования функций стандартной библиотеки, нужно явно подключить необходимую функцию с помощью ключевого слова extern, далее работа с ними производится как с пользовательскими функциями. Функции стандартной библиотеки с описанием представлены в таблице 1.9.

Так же в библиотеке есть приватные функции. Представлены они в таблице 1.10

Таблица 1.9 – Состав стандартной библиотеки

|  |  |  |
| --- | --- | --- |
| Функция(C++) | Возвращаемое значение | Описание |
| int random(int max) | uint | Возвращает случайное число в диапазоне от 0 до max |
| int ord(char) | char | Возвращает код символа |
| char chr(int) | char | Возвращает символ с заданным кодом |

Таблица 1.10 – Приватные функции стандартной библиотеки

|  |  |  |
| --- | --- | --- |
| Функция(C++) | Возвращаемое значение | Описание |
| char\* strConcat(void\* a, void\* b) | str | Конкатенирует строки. Вызывается оператором ‘+’ применённым на идентификаторах типа str |
| void outputuint (unsigned int a) | – | Выводит число на экран Вызывается оператором output |
| void outputchar (char a) | – | Выводит символ на экран Вызывается оператором output |
| void outputstr (void\* in) | – | Выводит строку на экран Вызывается оператором output |

Приватные функции не могут быть вызваны явно и не требуют предварительного пользовательского подключения. Они вызываются специальными операторами языка.

## **Ввод и вывод данных**

В языке SII-2020 не реализованы средства ввода данных.

Для вывода данных в стандартный поток вывода предусмотрен оператор output, который базируется на приватных функциях стандартной библиотеки.

## **Точка входа**

В языке SII-2020 каждая программа должна содержать главную функцию go, т. е. точку входа, с которой начнется последовательное выполнение программы.

## **Препроцессор**

Препроцессор в языке программирования SII-2020 не предусмотрен.

## **Соглашения о вызовах**

В языке вызов функций происходит по соглашению о вызовах stdcall. Особенности stdcall:

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## **Объектный код**

SII-2020 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке SII-2020 и выявления её транслятором в текущий файл протокола выводится сообщение. Классификация сообщений приведена в таблице 1.10.

Таблица 1.10. Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-99 | Системные ошибки |
| 100-109 | Ошибки параметров |
| 110-119 | Ошибки открытия и чтения файлов |
| 120-129 | Ошибки лексического анализа |
| 130-139 | Ошибки таблиц лексем и таблиц идентификаторов |
| 600-699 | Ошибки синтаксического анализа |
| 700-900 | Ошибки семантического анализа |

Компилятор может обрабатывать до 1000 различных ошибок.

## **Контрольный пример**

Контрольный пример представлен в главе Приложения.

# **Глава 2. Структура транслятора**

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

Транслятор преобразует программу, написанную на языке SII-2020 в программу на языке ассемблера.[1] Для указания выходных файлов используются входные параметры транслятора, которые описаны в пункте 2.2. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.



Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. Назначением лексического анализатора является нахождение ошибок лексики языка SII-2020 и формирование таблицы лексем и таблицы идентификаторов.

Семантический анализ в свою очередь является проверкой исходной программы SII-2020 на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода SII-2020. Для этого используются таблица лексем и идентификаторов. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке SII-2020, прошедший успешно все предыдущие этапы, в код на языке Ассемблера.

## **2.2 Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 — Входные параметры транслятора языка SII-2020

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с любым расширением , в котором содержится исходный код на языке SII-2020. Данный параметр должен быть указан обязательно. В случае если он не будет задан, то выполнение этапа трансляции не начнётся. | Не предусмотрено |
| -log:<имя\_файла> | Файл содержит в себе краткую информацию об исходном коде на языке SII-2020. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора. | <имя\_файла>.log |
| -out:<имя\_файла> | В этот файл будет записан результат трансляции кода на язык assembler | <имя\_файла>.asm |
| m | Вывод дерева разбора синтаксического анализатора. | – |
| l | Вывод таблицы лексем | – |
| i | Вывод таблицы идентификаторов | – |

Таблицы лексем и дерево разбора синтаксического анализатора выводятся в лог журнал.

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

Таблица с перечнем протоколов, формируемых транслятором языка SII-2020 и их назначением представлена в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка SII-2020

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, “\*.log ” | Файл содержит в себе краткую информацию об исходном коде на языке SII-2020. В этот файл выводится протокол работы анализаторов, а так же различные ошибки |
| “SII-2020ASM.asm” | Содержит сгенерированный код на языке Ассемблера. |

В log файл выводятся все ошибки, за исключением тех, что связаны с открытием файла log или считывания параметров.

# **Глава 3. Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке SII-2020. Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 — Структура лексического анализатора SII-2020

Результатом работы лексического анализатора являются заполненные таблица лексем и таблица идентификаторов.

## **3.2 Контроль входных символов**

Исходный код на языке программирования SII-2020 прежде чем транслироваться проверяется на допустимость символов. То есть изначально из входного файла считывается по одному символу и проверяется является ли он разрешённым.

Таблица для контроля входных символов представлена в приложении Ж.

Принцип работы таблицы заключается в соответствии значения каждому элементу в шестнадцатеричной системе счисления значению в таблице ASCII.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, S – пробельный символ, С – символ одинарной кавычки, L – символ-разделитель, D – символ двойной кавычки, O – символ начала комментария, N – символ новой строки.

## **3.3 Удаление избыточных символов**

Удаление избыточных символов не предусмотрено, так как после проверки на допустимость символов исходный код на языке программирования SII-2020 разбивается на токены, которые записываются в очередь.

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

Лексемы – это символы, соответствующие ключевым словам, символам операций и сепараторам, необходимые для упрощения дальнейшей обработки исходного кода программы. Данное соответствие описано в таблице 3.1.

Таблица 3.1 Соответствие ключевых слов, символов операций и сепараторов с лексемами

|  |  |  |  |
| --- | --- | --- | --- |
| Тип цепочки | Примечание | Цепочка | Лексема |
| Тип данных | Целочисленный тип данных | uint | i |
| Строковый тип данных | str | i |
| Символьный тип данных | char | i |
| Лексема | Объявление переменной | bind | v |
| Подключение функции библиотеки | extern | e |
| Оператор вывода | output | p |
| Объявление функции | func | f |
| Возврат значения из функции | stop | r |
| Инструкция цикла | while | u |
| Блок инструкции цикла | [ | [ |
| ] | ] |
| Блок функции | { | { |
| } | } |
| Изменение приоритетности в выражении и отделение параметров функций | ( | ( |
| ) | ) |
| Сепараторы | ; | ; |
| , | , |
| Оператор присваивания | = | v |
| Условный оператор | < | b |
| > | b |
| & | b |
| ^ | b |

Окончание таблицы 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Оператор | Знаки арифметических операций | + | v |
| - | v |
| \* | v |
| / | v |
| \ | v |
| : | v |
| % | v |
| Идентификатор |  | [a-z|A-Z]+  [a-z|A-Z|0-9]\* | i |
| Литерал | Целочисленный литерал | [1-9]+[0-9]\* | l |
|  | Символьный литерал | [a-z|A-Z|0-9]\*  кроме ‘ | l |
|  | Строковый литерал | [a-z|A-Z|0-9]\* кроме ” | l |
| Точка входа |  | go | m |

Каждому выражению соответствует детерминированный конечный автомат, то есть автомат с конечным состоянием, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся фраза и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Пример реализации таблицы лексем представлен в приложении А.

Также в приложении А находятся конечные автоматы, соответствующие лексемам языка SII-2020.

## **3.5 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных языка SII-2020, используемых для хранения, представлены в приложении А. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде, номер столбца в исходном коде, индекс таблицы идентификаторов (если нет соответствующего идентификатора, то индекс равен -1), а также специальное поле, в котором хранится значение лексемы. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора, его значение, а также бинарное поле для определения внешний ли идентификатор.

## **3.6 Принцип обработки ошибок**

Все ошибки являются критическими и приводят к прекращению работы транслятора и выводу диагностического сообщения в лог файл.

## **3.7 Структура и перечень сообщений лексического анализатора**

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 120-125. Также сам текст ошибки содержит в себе префикс [LA]. Перечень сообщений лексического анализатора представлен на рисунке 3.3.

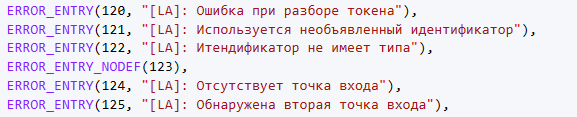


Рисунок 3.2 – Перечень ошибок лексического анализатора

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

Входные параметры используются для вывода результата работы лексического анализатора. Они передаются аргументами через командную строку и рассмотрены в таблице 2.1

## **3.9 Алгоритм лексического анализа**

Алгоритм работы лексического анализа заключается в последовательном распознавании и разборе цепочек исходного кода и заполнение таблиц идентификаторов и лексем. Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов. В случае, если подходящий автомат не был обнаружен, запоминается номер строки, в которой находился этот токен и выводится сообщение об ошибке. Если токен разобран, то дальнейшие действия, которые будут с ним производиться, будут зависеть от того, чем он является. Регулярные выражения — аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа.

В случае, если токен является идентификатором, перед его именем записывается название функции в которой он объявлен и после этого он заносится в таблицу идентификаторов.

В случае, если токен является идентификатором функции, название функции в которой он объявлен не записывается.

В случае, если токен является литералом, то он заносится в таблицу идентификаторов в виде abi, где a – имя функции, где объявлен литерал, b – “$LEX”, c –количество определённых литералов+1.

Когда встречаем токен, являющийся ключевым словом, которое отвечает за тип данных или вид идентификатора, заносим лексему, соответствующую ему, в таблицу лексем и запоминаем тип данных или вид идентификатора, которому он соответствует. В последствии, когда встречаем идентификатор, заносим его в таблицу идентификаторов с соответствующим ему типом данных и видом идентификатора, и именем вида “ab”, где a – имя функции, где объявлен идентификатор, b – имя идентификатора.

Пример. Регулярное выражение для ключевого слова bind: «bind».

Граф конечного автомата для этой лексемы представлен на рисунке 3.4. S0 – начальное состояние, S4 – конечное состояние автомата.

b

n

d

i

Рисунок 3.3 – Граф переходов для цепочки «bind»

## **3.10 Контрольный пример**

Результат работы лексического анализатора – таблицы лексем и идентификаторов – представлен в приложении А.

# **Глава 4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций SII-2020. Входом для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные после фазы лексического анализа. Выходом – дерево разбора. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора SII-2020

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

В синтаксическом анализаторе транслятора языка SII-2020 используется контекстно-свободная грамматика , где

T – множество терминальных символов (было описано в разделе 1.2 данной пояснительной записки),

N – множество нетерминальных символов (первый столбец таблицы 4.1),

P – множество правил языка (второй столбец таблицы 4.1),

S – начальный символ грамматики, являющийся нетерминалом.

Эта грамматика имеет нормальную форму Грейбах, т.к. она не леворекурсивная (не содержит леворекурсивных правил) и правила  имеют вид:

1. , где ; (или , или )
2. , где — начальный символ, при этом если такое правило существует, то нетерминал  не встречается в правой части правил.

Грамматика языка SII-2020 представлена в приложении Б.

TS – терминальные символы, которыми являются сепараторы, знаки арифметических операций и некоторые строчные буквы.

NS – нетерминальные символы, представленные несколькими заглавными буквами латинского алфавита.

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов SII-2020

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | m{NrE;};  m{rE;};  dtfi(F){NrE;};S  dtfi(F){rE;};S  dtfi(){NrE;};S  dtfi(){rE;};S  m{rE;};S | Порождает правила, описывающее общую структуру программы |
| N | dti;  rE;  ivE;  etfi(F);  dti;N  ivE;N  etfi(F);N  pi;  pl;  pi;N  pl;N  u(B)[N];  u(B)[N];N  etfi();N  etfi(); | Порождает правила, описывающие конструкции языка |
| E | i  l  (E)  i(W)  iM  lM  (E)  i(W)  i()  i(M)  l(M)  (E)M  i(W)M  i(W) | Порождает правила, описывающие выражения |

Окончание таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| F | ti  ti,f | Порождает правила, описывающие параметры локальной функции при её объявлении |
| W | i  l  i,W  l,W | Порождает правила, описывающие принимаемые параметры функции |
| B | ibi  ibl  lbi  lbl | Порождает правила, описывающие условное выражение в операторе цикла |
| M | vE  v(E)  v(E)M  vEM | Порождает правила, описывающие знаки арифметических операций |

Протокол и ошибки работы синтаксического анализатора выводятся в лог журнал.

## **4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку, описание которой представлено в таблице 4.2.[1] Структура данного автомата показана в приложении В.

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |

Окончание таблицы 4.2

|  |  |  |
| --- | --- | --- |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ А) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

Для вывода результата работы синтаксического анализатора нужно использовать флаг m.

## **4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного конечного автомата и структуру грамматики Грейбах, описывающей правила языка SII-2020. Данные структуры представлены в приложении В.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

1. в магазин записывается стартовый символ;
2. на основе полученных ранее таблиц формируется входная лента;
3. запускается автомат;
4. выбирается цепочка, 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 стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала;
6. если в магазине встретился нетерминал, переходим к пункту 4;
7. если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно и формируется дерево разбора. Иначе генерируется исключение.

## **4.6 Структура и перечень сообщений синтаксического анализатора**

Индексы ошибок, обнаруживаемых синтаксическим анализатором, находятся в диапазоне 600-609. Перечень сообщений синтаксического анализатора представлен на рисунке 4.1.

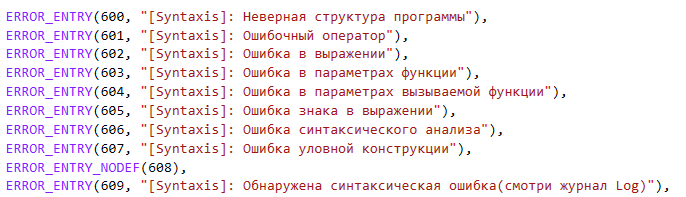


Рисунок 4.1 – Перечень сообщений синтаксического анализатора

Текст синтаксической ошибки содержит в себе префикс [Syntaxis].

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

Для вывода результата работы синтаксического анализатора используются входные параметры, описанные в пункте 2.2Перечень входных параметров транслятора в таблице 2.1.

## **4.8 Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

1. Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
2. Если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка.
3. В случае ошибки выводится соответствующее сообщение в журнал лога и компилятор прекращает работу.

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке SII-2020 представлен в приложении Г. Дерево разбора исходного кода также представлено в приложении Г.

# **Глава 5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ происходит при выполнении фазы лексического анализа и реализуется в виде отдельных проверок текущих ситуаций в конкретных случаях: установки флага или нахождении в особом месте программы (оператор выхода из функции, оператор ветвления, вызов функции стандартной библиотеки). Структура семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1 — структура семантического анализатор

Функции семантического анализатора частично реализованы в лексическом анализаторе.

## **5.2 Функции семантического анализатора**

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

## **5.3 Структура и перечень сообщений семантического анализатора**

Все ошибка семантического анализатора имеют идентификатор свыше 700. Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

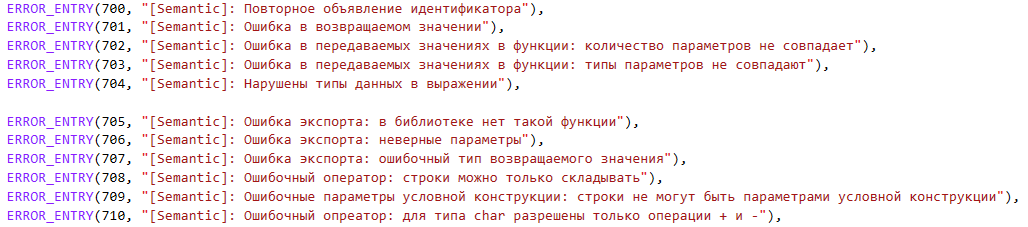


Рисунок 5.2 – Перечень сообщений семантического анализатора

Текст семантической ошибки содержит в себе префикс [Semantic].

## **5.4 Принцип обработки ошибок**

При обнаружении хотя бы одной ошибки транслятор завершит свою работу c

Записью информации об ошибке в лог файл.

## **5.5 Контрольный пример**

Результат работы контрольного примера расположен в приложении А, где показан результат лексического анализатора, т.к. представленные таблицы лексем и идентификаторов проходят лексическую и семантическую проверки одновременно.

**Глава 6. Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке SII-2020 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, ^, :, /, \ и (), и вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке SII-2020

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

Некоторые из операций таблицы 6.1 используются для типов, отличных от целочисленных.

## **6.2 Польская запись**

Выражения в языке SII-2020 преобразовываются к обратной польской записи.

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок.

Обратная польская запись – это форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. 

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;

– исходная строка просматривается слева направо;

– 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 «@».

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| x+y\*5/(z-2) |  |  |
| +y\*5/(z-2) | x |  |
| y\*5/(z-2) | x | + |
| \*5/(z-2) | xy | + |
| 5/(z-2) | xy | +\* |
| /(z-2) | xy5 | +\* |
| (z-2) | xy5\* | +/ |
| z-2) | xy5\* | +/( |
| -2) | xy5\*z | +/( |
| 2) | xy5\*z | +/(- |
| ) | xy5\*z2 | +/(- |
|  | xy5\*z2- | +/ |
|  | xy5\*z2-/ | + |
|  | xy5\*z2-/+ |  |

Как результат успешного разбора, мы получаем пустой стек и заполненную результирующую строку.

## **6.3 Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений к польской записи представлена в приложении Д.

## **6.4 Контрольный пример**

Пример преобразования выражения к польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления.

# **Глава 7. Генерация кода**

## **7.1 Структура генератора кода**

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы SII-2020 в цепочку символов выходного языка. На вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом.



Рисунок 7.1 — Структура генератора кода

## **7.2 Представление типов данных в оперативной памяти**

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. [2] Соответствия между типами данных идентификаторов на языке SII-2020 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка SII-2020 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке SII-2020 | Тип идентификатора на языке ассемблера | Пояснение |
| char | BYTE | Хранит символьный тип данных. |
| str | DWORD | Хранит указатель на начало строки. |
| uint | DWORD | Хранит целочисленный тип данных без знака. |
| Лексема | BYTE  DWORD  DWORD | Литералы: символьные,  целочисленные, строковые |

Идентификаторы языка SII-2020 размещены в сегменте данных (.data). Литералы – в сегменте констант (.const).

## **7.3 Алгоритм работы генератора кода**

Алгоритм генерации кода выглядит следующим образом:

1. Генерирует заголовочную информацию (Лист. 7.1): модель памяти, подключение библиотек, прототипы внешних функций, размер стека.

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib ../Debug/SII-2020LIB.lib  ExitProcess PROTO :DWORD  random PROTO :DWORD  outputuint PROTO :DWORD  outputchar PROTO :BYTE  outputstr PROTO :DWORD  strConcat PROTO :DWORD,:DWORD |

Листинг 7.1 –Заголовочная информация

1. Проходит полностью таблицу идентификаторов и заполняет поле .const литералами (Лист. 7.2).

|  |
| --- |
| .const  FindFact$LEX1 DWORD 1 ;INT  main$LEX4 BYTE "Нахождение факториала числа ", 0 ;STR  main$LEX5 DWORD 11 ;INT  main$LEX6 BYTE "Равен ", 0 ;STR  main$LEX7 BYTE "Демонстрация сдвига", 0 ;STR  main$LEX8 DWORD 32 ;INT  main$LEX9 DWORD 2 ;INT  main$LEX10 BYTE "demo<<2", 0 ;STR  main$LEX12 BYTE "demo>>2", 0 ;STR |

Листинг 7.2 – Пример заполнения поля .const

1. Проходим таблицу идентификаторов и объявляем переменные в поле .data. (Лист. 7.3).

|  |  |
| --- | --- |
| .data  FindFactanswer DWORD 0 ;INT  mainnumber DWORD 0 ;INT  maindemo DWORD 0 ;INT  maindemo1 DWORD 0 ;INT  maindemo2 DWORD 0 ;INT  maina DWORD 0 ;INT  mainqq DWORD 0 ;INT |  |

Листинг 7.3 – Пример заполнения поля .data

1. Генерируем сегмент данных .code (Лист. 7.4). Сперва проходим по таблице идентификаторов и ищем функции. Объявляем их и генерируем код, содержащийся в функциях. При генерации кода, при встрече оператора присваивания, описываем вычисление выражения. Описание алгоритма преобразования выражений представлено в пункте 7.3.

|  |
| --- |
| .code  FindFact PROC uses ebx ecx edi esi , FindFacta: DWORD  ; String #3 :ivl  push FindFactLEX1  pop FindFactanswer  While17Start:  mov eax, FindFacta  mov ebx, FindFactLEX1  cmp eax, ebx  jl While17End  ; String #4 :iviiv  push FindFactanswer |

Листинг 7.4 – Пример заполнения поля .code

После генерации всех пользовательских функций, генерируется функция начала программы go в функции main по такому же принципу.

## **Контрольный пример**

Генерируемый код записывается в файл заданный параметром “-out”. Сгенерированный код можно посмотреть в приложении Е.

# **Глава 8. Тестирование транслятора**

## **8.1 Тестирование фазы проверки на допустимость символов**

В языке SII-2020 не разрешается использовать запрещённые входным алфавитом символы где-либо кроме строковых или символьных переменных. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| go{в} | Ошибка 111: [IN]: Недопустимый символ в исходном файле (-in), строка 1, столбец 4 |

Запрещённые символы перечислены в Приложении Ж.

## **Тестирование лексического анализатора**

На этапе лексического анализа могут возникнуть ошибки, описанные в пункте 3.7. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| go{  bind uint 1s;  } | Ошибка 120: [LA]: Ошибка при разборе токена, строка 2, столбец 11 |
| go{  s = 5;  } | Ошибка 121: [LA]: Используется необъявленный идентификатор, строка 2, столбец 1 |
| bind uint func a(){  bind q;  } | Ошибка 122: [LA]: Идентификатор не имеет типа, строка 2, столбец 6 |
| bind uint func a(){  bind uint q;  } | Ошибка 124: [LA]: Отсутствует точка входа |
| go(){  bind char a;  }  go(){  bind char z;  } | Ошибка 125: [LA]: Обнаружена вторая точка входа, строка 4, столбец 1 |

Ошибка лексического анализатора приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

## **8.3 Тестирование синтаксического анализатора**

На этапе синтаксического анализа могут возникнуть ошибки, описанные в пункте 4.6. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| go{  bind char a  } | Ошибка 609: [Syntaxis]: Обнаружена синтаксическая ошибка(смотри журнал Log) |

Ошибка синтаксического анализатора также приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

## **8.4 Тестирование семантического анализатора**

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| go{  bind char a;  bind str a;  } | Ошибка 700: [Semantic]: Повторное объявление идентификатора, строка 3, столбец 10 |
| bind uint func f(uint q){  bind uint a;  stop a;  };  go{  bind str a;  a = f(3);  stop 0;  }; | Ошибка 704: [Semantic]: Нарушены типы данных в выражении, строка 7, столбец 5 |
| bind uint func f(uint q){  bind str a;  stop a;  };  go{  bind uint a;  a = f(3);  stop 0;  }; | Ошибка 701: [Semantic]: Ошибка в возвращаемом значении, строка 3, столбец 6 |

Окончание таблицы 8.4

|  |  |
| --- | --- |
| bind uint func f(str a){  stop 5;  };  go{  bind uint a;  a = f(a);  stop a;  }; | Ошибка 703: [Semantic]: Ошибка в передаваемых значениях в функции: типы параметров не совпадают, строка 6, столбец 7 |
| bind uint func f(uint a, uint b){  stop 5;  };  go{  bind uint a;  a = f(a);  stop a;  }; | Ошибка 702: [Semantic]: Ошибка в передаваемых значениях в функции: количество параметров не совпадает, строка 6, столбец 8 |
| go{  extern uint func rand(uint max);  bind uint a;  a = f(a);  stop a;  }; | Ошибка 705: [Semantic]: Ошибка экспорта: в библиотеке нет такой функции, строка 2, столбец 18 |

Ошибка семантического анализатора также приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

# **Заключение**

По окончании выполнения всех пунктов, изложенных ранее, получили рабочий транслятор языка программирования SII-2020 на язык ассемблера.

Язык SII-2020 поддерживает 3 типа данных: беззнаковый целочисленный (uint), строковый (str), символьный(char).

Для целочисленного типа реализована обработка 7 арифметических действий, скобок, обозначающих приоритет операций.

Для символьного типа реализована обработка 2 арифметических действий, скобок, обозначающих приоритет операций.

Для строкового типа данных определена операция конкатенации.

На этапе семантического анализа производится проверка соответствия исходного кода спецификации по 10 правилам.

Реализованы 3 публичные и 4 приватные функции стандартной библиотеки.

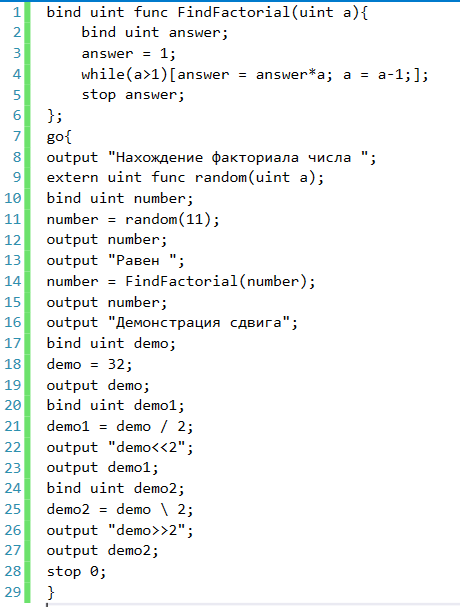
Итоговое суммарное количество строк исходного кода ~4761.

# **Литература**

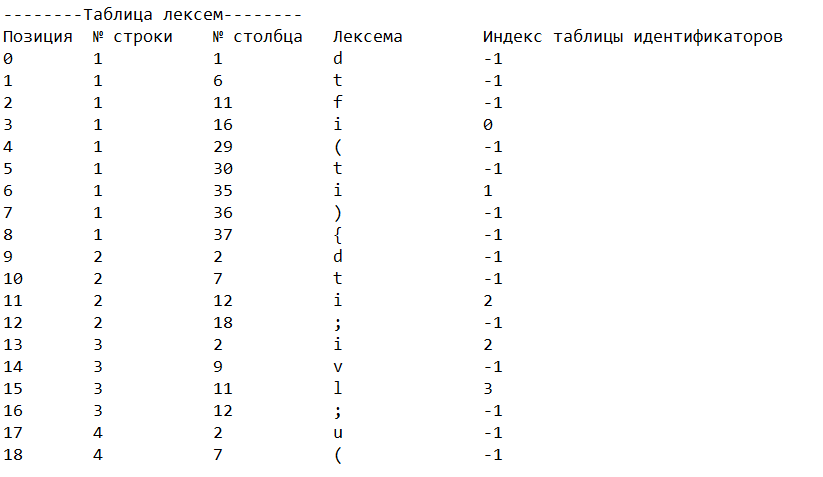
1. Карпов Ю. Теория и технология программирования. Основы построения трансляторов, 2005. – 272с.
2. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.

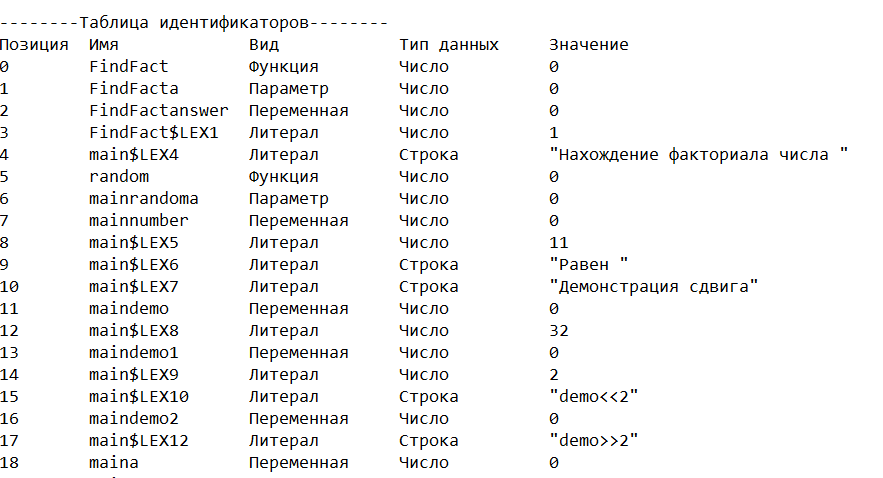
# **Приложения**

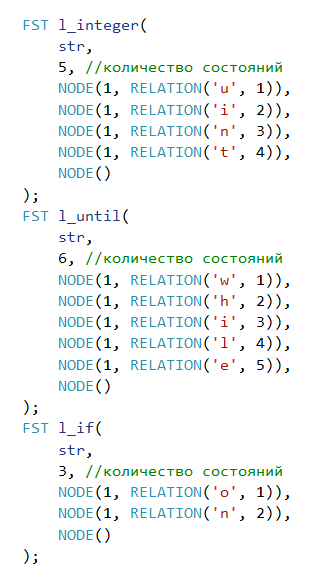
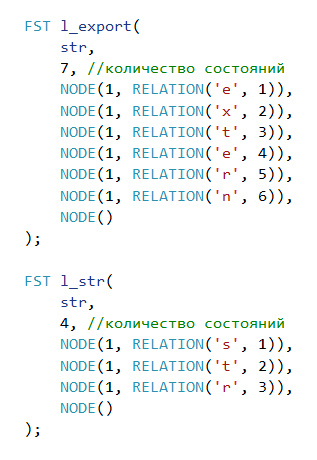
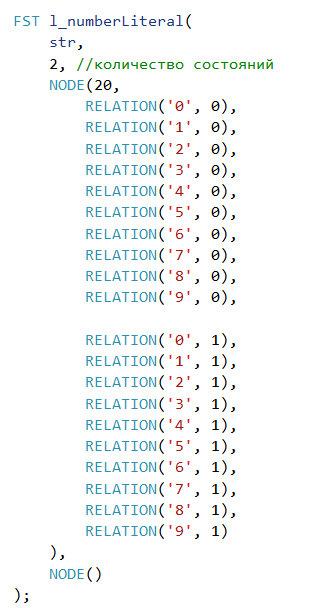
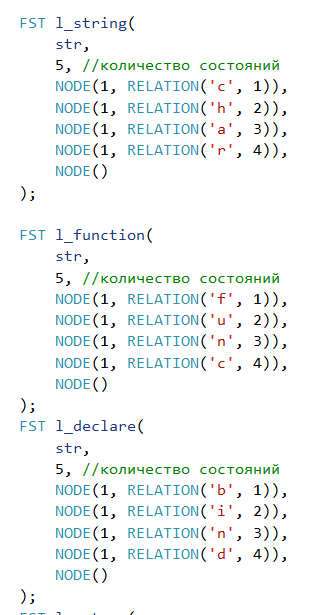
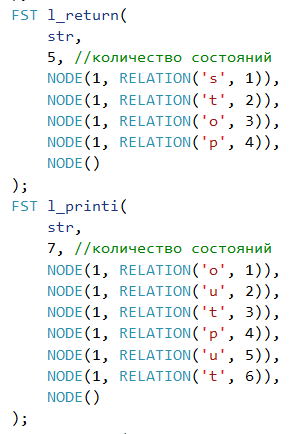
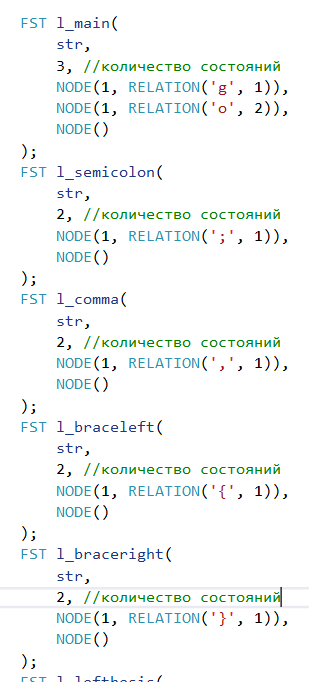
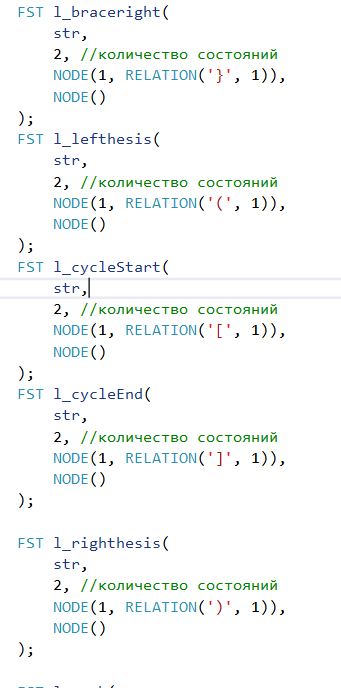
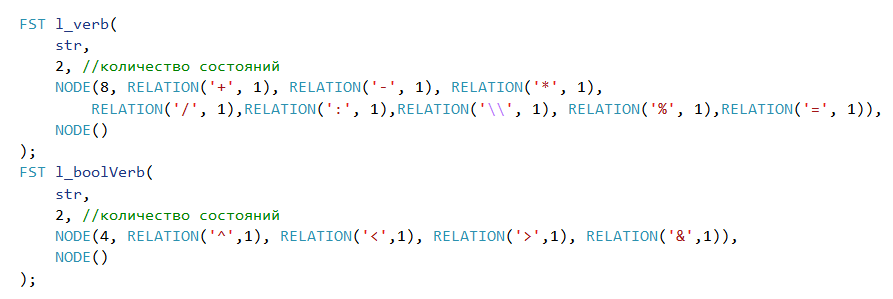
## **Контрольный пример**

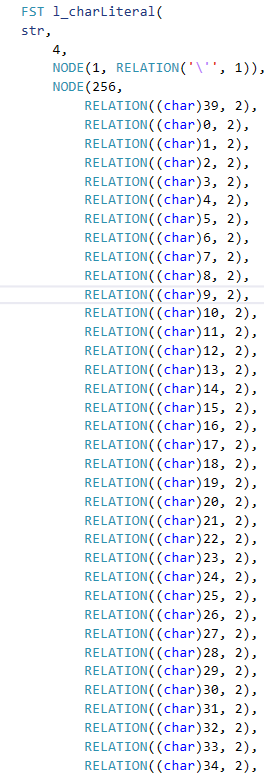
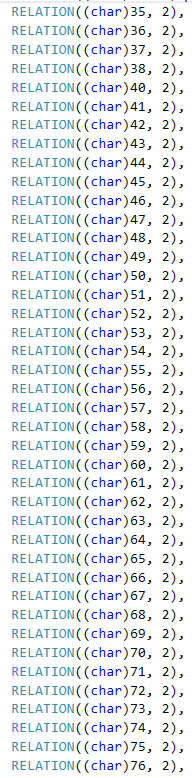
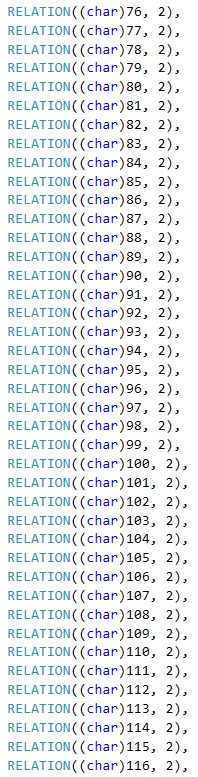
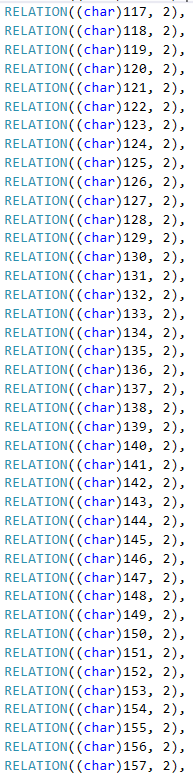
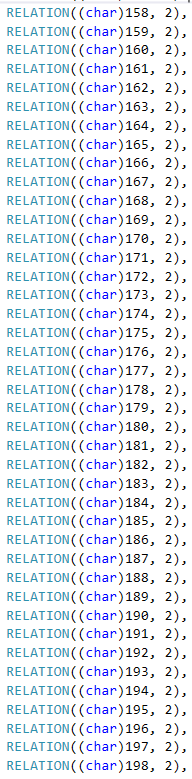
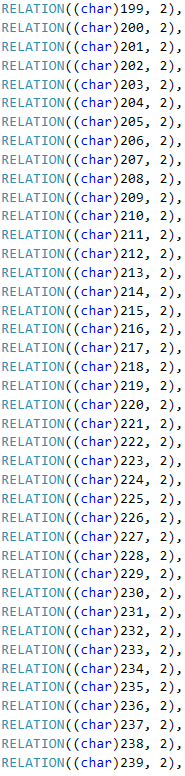
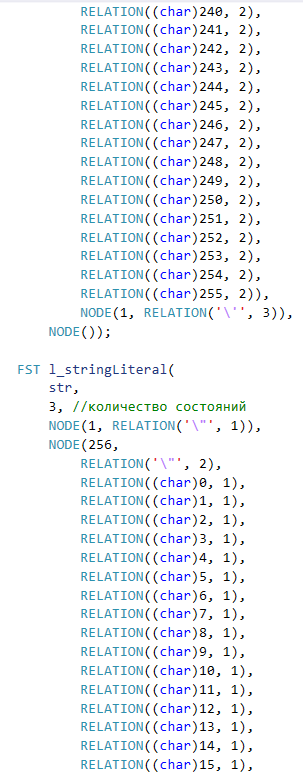


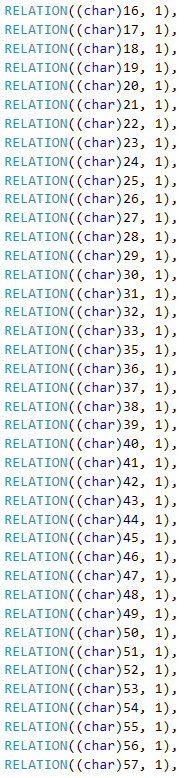
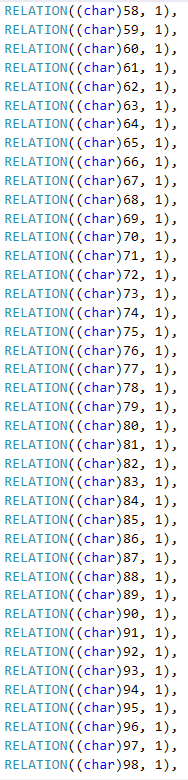
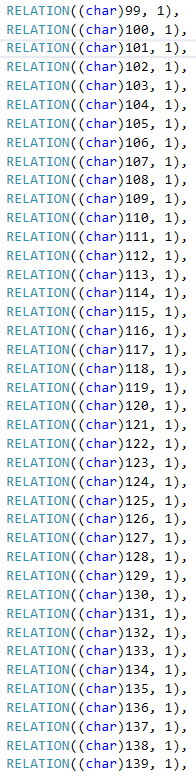
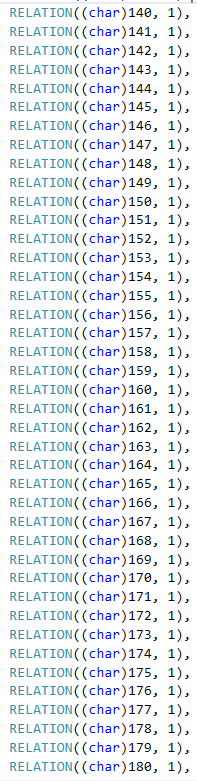
## **Приложение А**

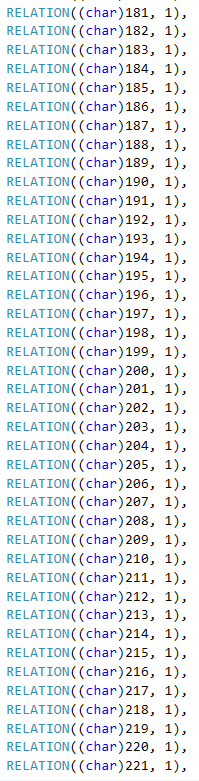
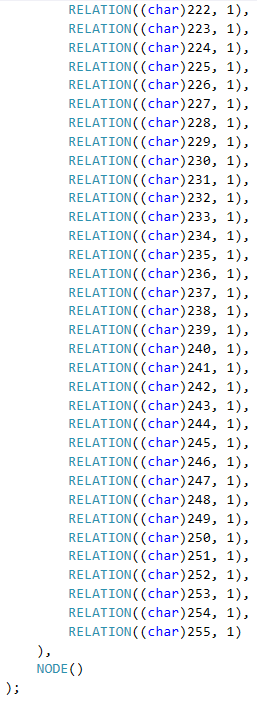
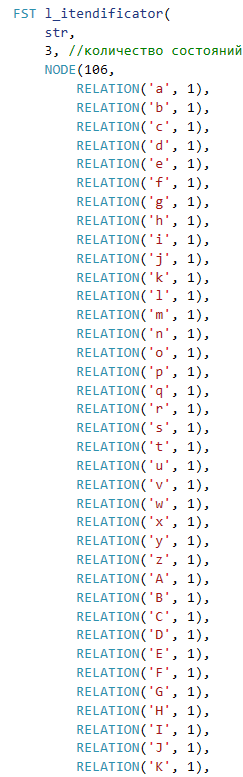


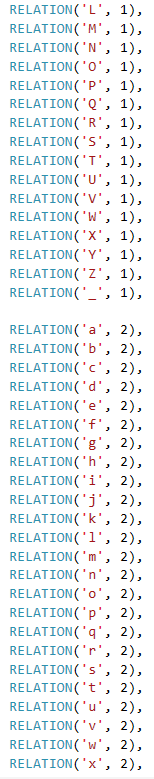
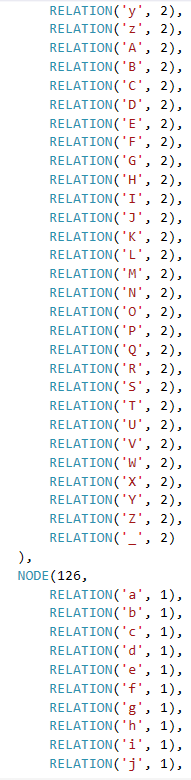
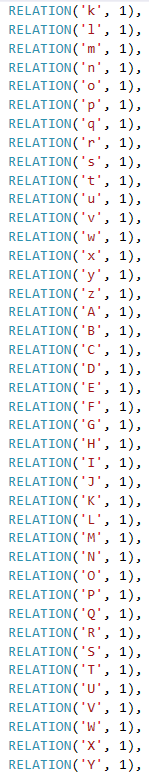
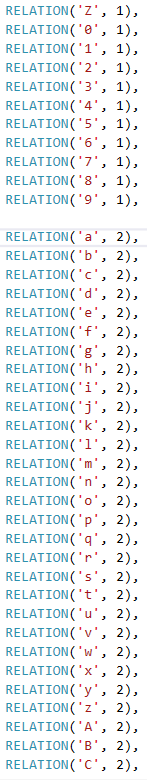
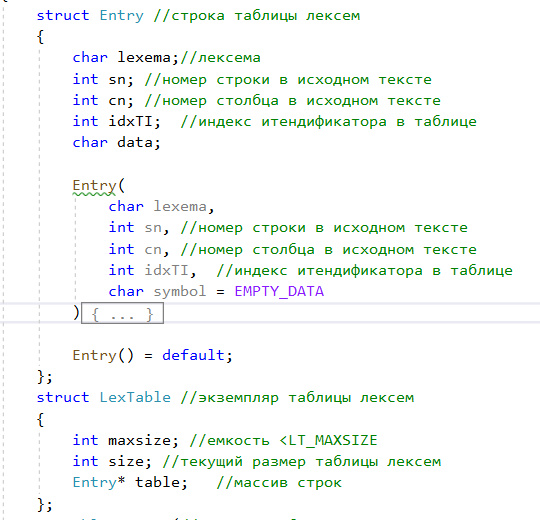
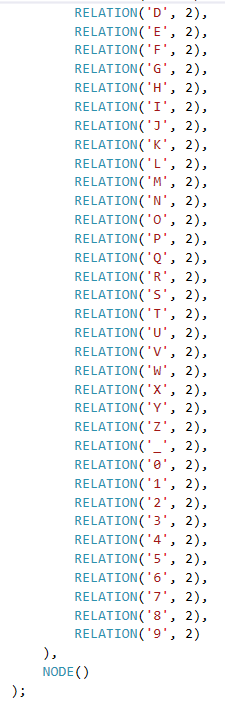


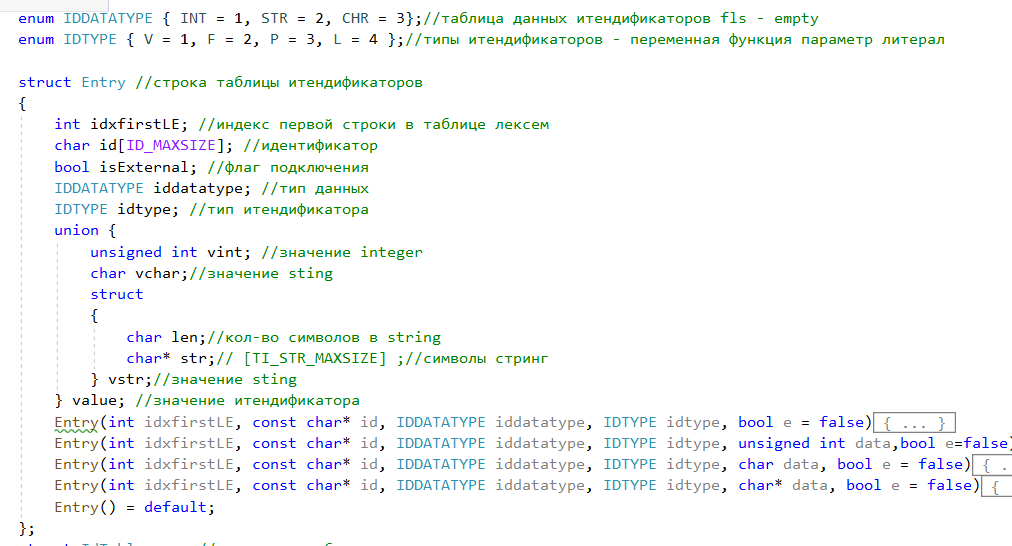
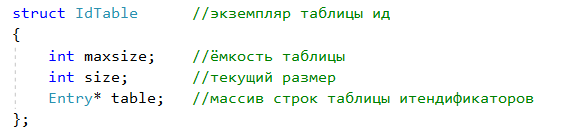
       

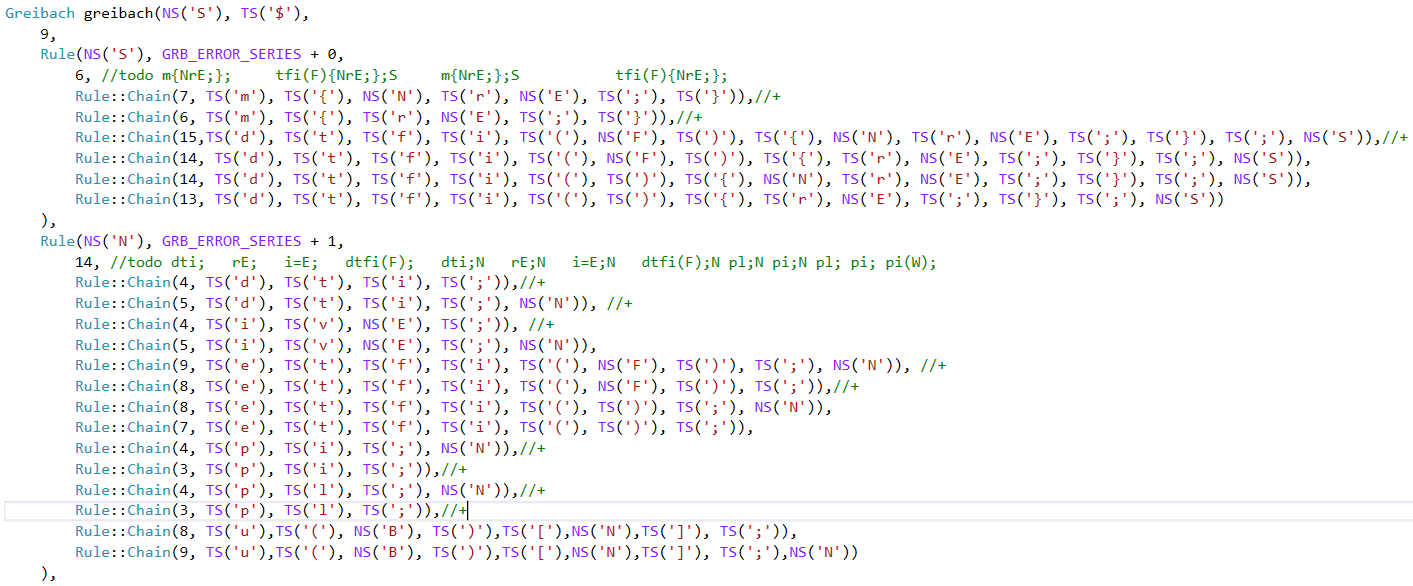
   

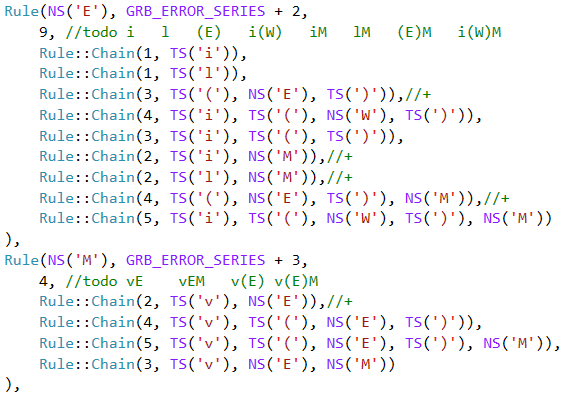
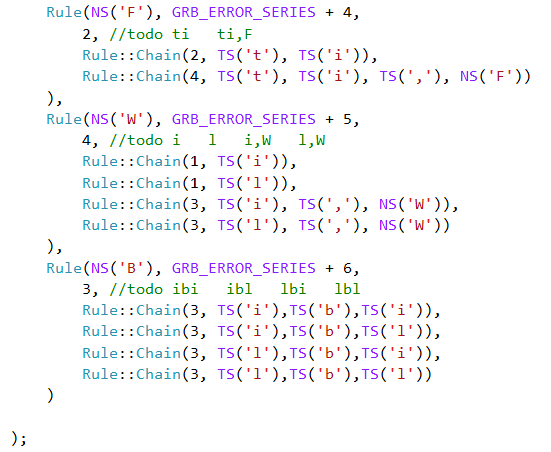
  

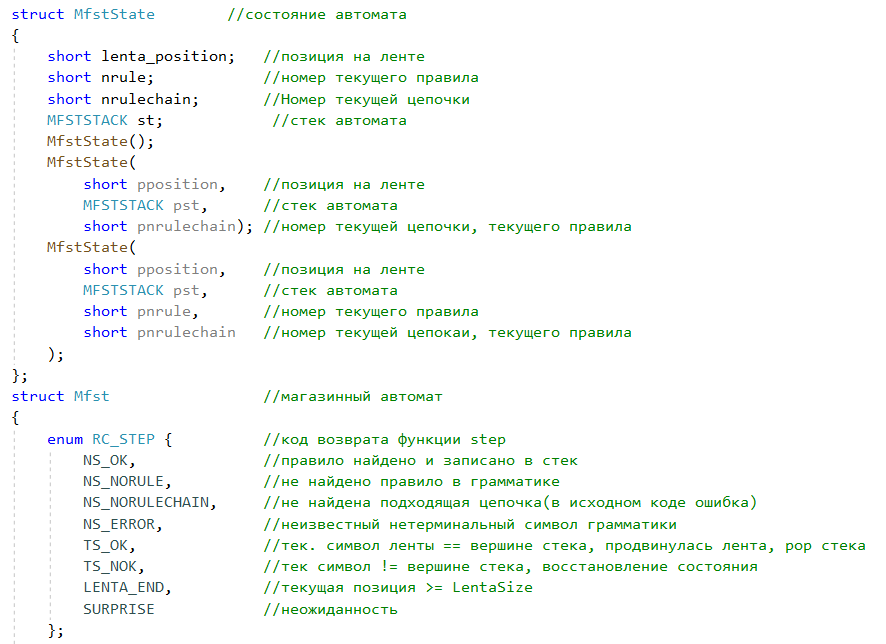
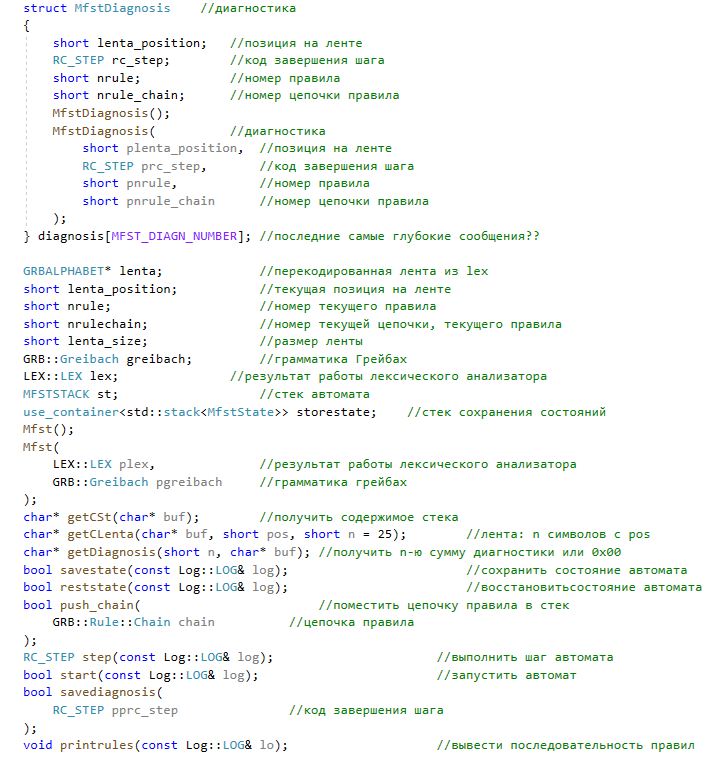
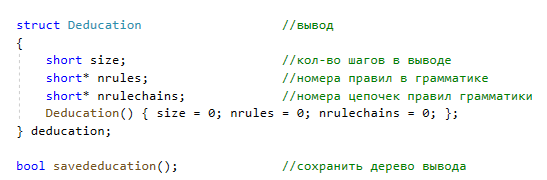
 

**Приложение Б**



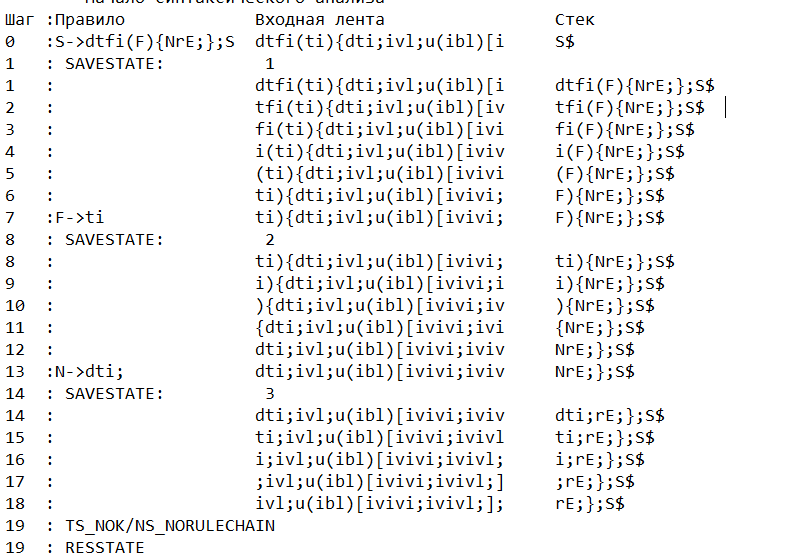
 

**Приложение В**

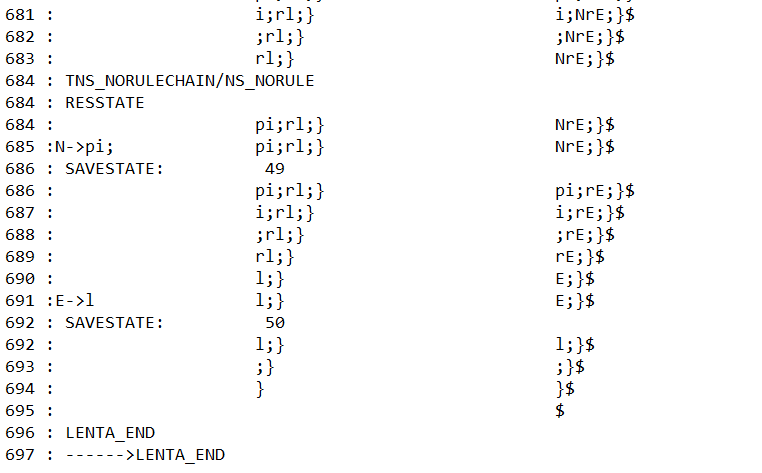
  

**Приложение Г**

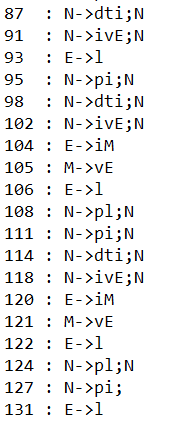
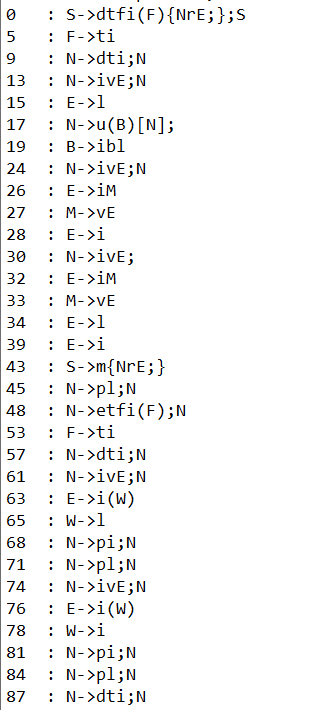
Начало разбора



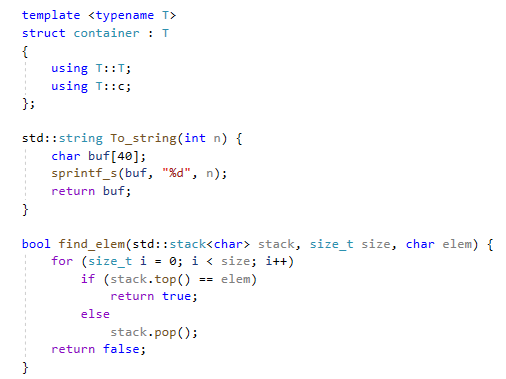
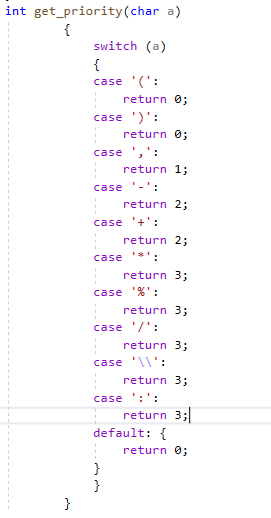
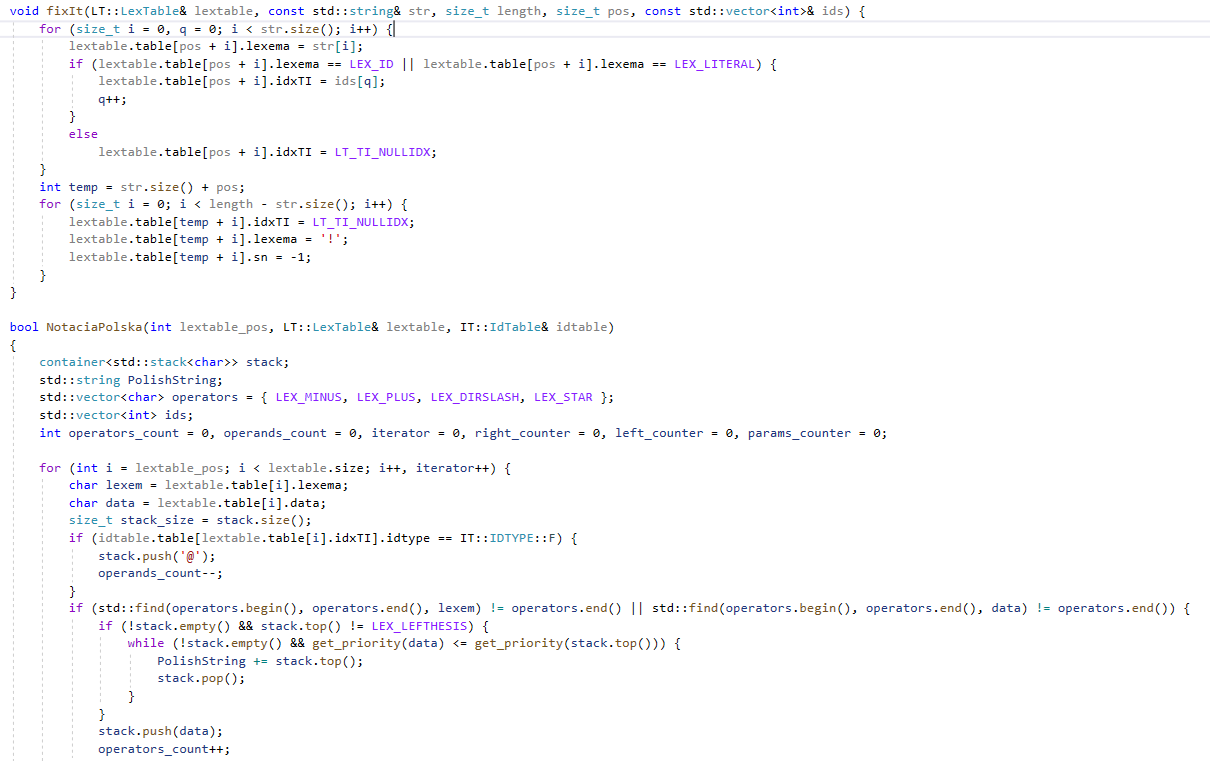
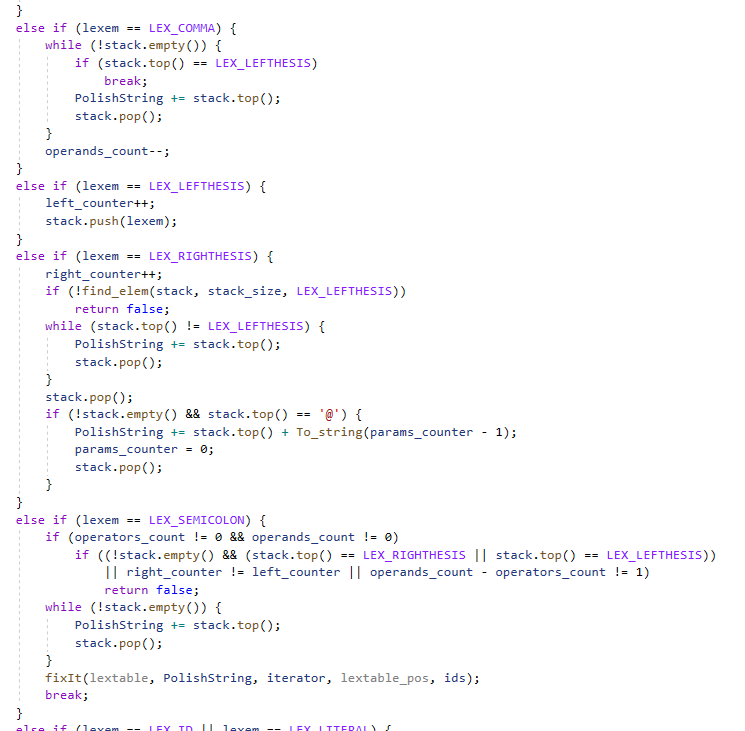
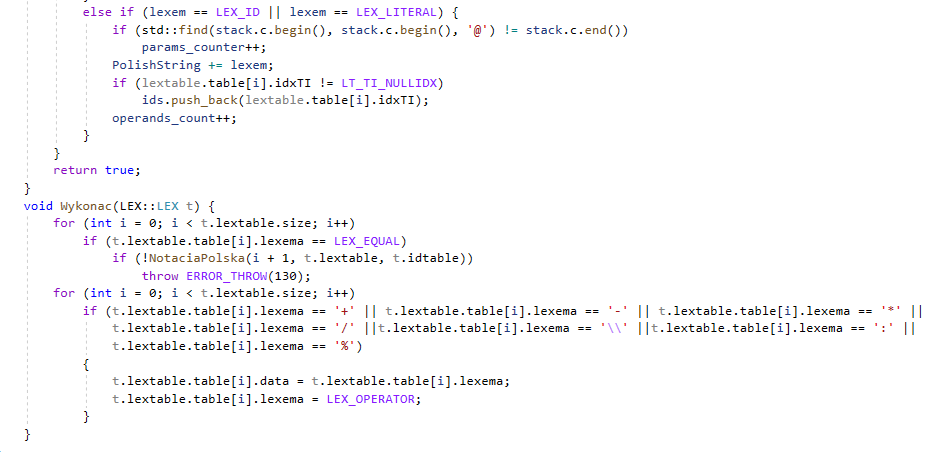
Конец разбора



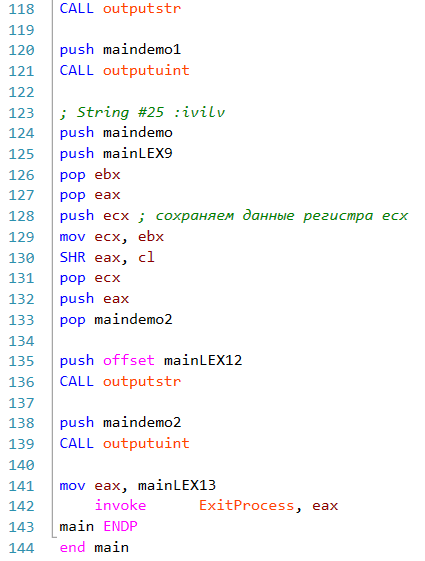
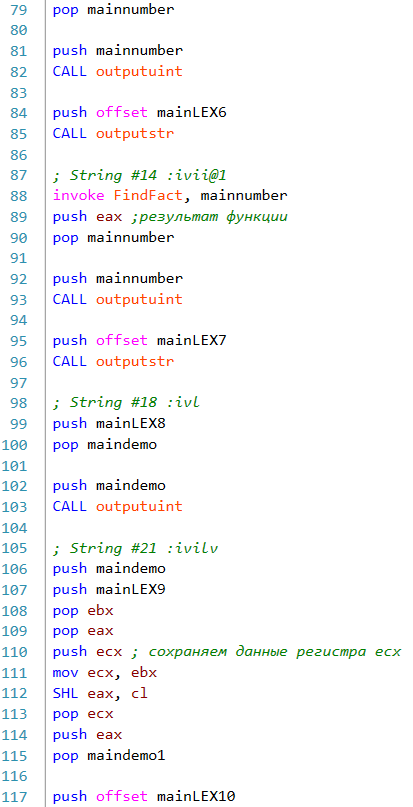
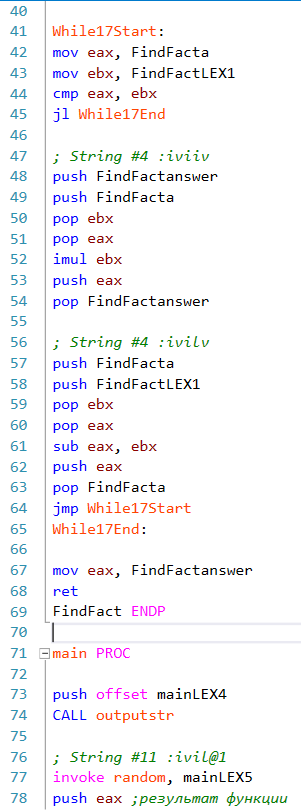
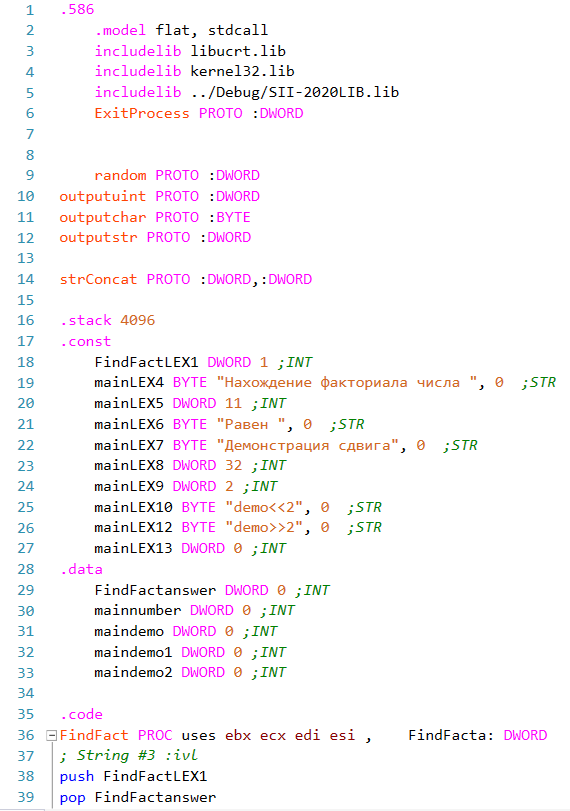
Правила разбора



**Приложение Д**

**Приложение Е**



**ПРИЛОЖЕНИЕ Ж**

