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 FSI-2020»

Выполнил студент Филиппов Серафим Игоревич

(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.)

Нормоконтролер пр.ст. Пахолко Алёна Степановна

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

Курсовой проект защищен с оценкой

Минск 2020

**Содержание**

[Введение 4](#_Toc501385915)

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

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

[1.2. Алфавит языка 5](#_Toc501385918)

[1.3. Символы сепараторы 5](#_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 8](#_Toc501385927)

[1.12. Операции языка 9](#_Toc501385928)

[1.13. Выражения и их вычисления 9](#_Toc501385929)

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

[1.15. Область видимости 10](#_Toc501385931)

[1.16. Семантические проверки 10](#_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 11](#_Toc501385935)

[1.20. Точка входа 12](#_Toc501385936)

[1.21. Препроцессор 12](#_Toc501385937)

[1.22. Соглашения о вызовах 12](#_Toc501385938)

[1.23. Объектный код 12](#_Toc501385939)

[1.24. Классификация сообщений транслятора 12](#_Toc501385940)

[1.25. Контрольный пример 12](#_Toc501385941)

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

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 13](#_Toc501385943)

[2.2 Перечень входных параметров транслятора 14](#_Toc501385944)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 14](#_Toc501385945)

[Глава 3. Разработка лексического анализатора 15](#_Toc501385946)

[3.1 Структура лексического анализатора 15](#_Toc501385947)

[3.2 Контроль входных символов 15](#_Toc501385948)

[3.3 Удаление избыточных символов 16](#_Toc501385949)

[3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов 16](#_Toc501385950)

[3.5 Основные структуры данных 17](#_Toc501385951)

[3.6 Принцип обработки ошибок 18](#_Toc501385952)

[3.7 Структура и перечень сообщений лексического анализатора 18](#_Toc501385953)

[3.8 Параметры лексического анализатора и режимы его работы 18](#_Toc501385954)

[3.9 Алгоритм лексического анализа 19](#_Toc501385955)

[3.10 Контрольный пример 19](#_Toc501385956)

[Глава 4. Разработка синтаксического анализатора 20](#_Toc501385957)

[4.1 Структура синтаксического анализатора 20](#_Toc501385958)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 20](#_Toc501385959)

[4.3 Построение конечного магазинного автомата 22](#_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 27](#_Toc501385970)

[5.4 Принцип обработки ошибок 28](#_Toc501385971)

[5.5 Контрольный пример 28](#_Toc501385972)

[Глава 6. Вычисление выражений 29](#_Toc501385973)

[6.1 Выражения, допускаемые языком 29](#_Toc501385974)

[6.2 Польская запись 29](#_Toc501385975)

[6.3 Программная реализация обработки выражений 32](#_Toc501385976)

[6.4 Контрольный пример 32](#_Toc501385977)

[Глава 7. Генерация кода 31](#_Toc501385978)

[7.1 Структура генератора кода 31](#_Toc501385979)

[7.2 Представление типов данных в оперативной памяти 31](#_Toc501385980)

[7.3 Статическая библиотека 32](#_Toc501385981)

[7.4 Особенности алгоритма генерации кода 32](#_Toc501385981)

[7.5 Входные параметры генератора кода 33](#_Toc501385981)

[7.6 Контрольный пример 33](#_Toc501385981)

[Глава 8. Тестирование транслятора 34](#_Toc501385982)

[8.1 Общие положения 34](#_Toc501385983)

[8.2 Результаты тестирования 34](#_Toc501385984)

[Заключение 36](#_Toc501385982)

[Список использованных источников 37](#_Toc501385982)

[ПРИЛОЖЕНИЯ 38](#_Toc501385982)

[Контрольный пример 38](#_Toc501385988)

[ПРИЛОЖЕНИЕ А 39](#_Toc501385989)

[ПРИЛОЖЕНИЕ Б 53](#_Toc501385990)

[ПРИЛОЖЕНИЕ В 59](#_Toc501385991)

[ПРИЛОЖЕНИЕ Г 63](#_Toc501385992)

# **Введение**

Целью выполнения курсового является приобретение навыков разработки системы программирования (трансляторов, интерпретаторов). В частности, поставлена задача разработки компилятора для моего языка программирования FSI-2020.

Текст программы должен быть оттранслирован в соответствующую последовательность команд, прежде чем он может быть выполнен компьютером. Эта трансляция сама может быть описана программой. Транслирующая программа называется компилятором, а текст, который должен транслироваться, называется исходным текстом [1].

Транслятор FSI-2020 состоит из следующих частей:

– лексический и семантический анализаторы;

– синтаксический анализатор;

– генератор исходного кода на языке ассемблера.

В соответсвии с курсовым проектом были определены следующие задачи:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка программной реализации лексического анализатора;

– разработка программной реализации синтаксического анализатора;

– разработка программной реализации семантического анализатора;

– разработка программной реализации преобразования выражений;

– разработка программной реализации генератора кода;

– выполнить тестирование, разработанного программного обеспечения;

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

# **Глава 1. Спецификация языка программирования**

## **Характеристика языка программирования**

Язык FSI-2020 – это строго типизированный, процедурный, универсальный, регистр зависимый, не объектно-ориентированный язык программирования.

* 1. **Алфавит языка**

Исходный код языка поддерживает латинский алфавит, цифры десятичной системы счисления от 0 до 9, не поддерживает русский алфавит.

Алфавит языка FSI-2020 представлен в таблице 1.1.

Таблица 1.1 ­– Алфавит языка

|  |  |
| --- | --- |
| Символы латинского алфавита | A, B, C, D, E, F, G, H, I, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z;  a, b, c, d, e, f, g, h, i, g, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z. |
| Цифры | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 |
| Специальные символы | , (запятая), ;, +, -, \*, =, /, %, (, ), {, }, ‘ (апостроф), (пробел), “ (кавычки). |

Из допустимых символов формируются лексемы ­– для предопределённых константы, идентификаторы и знаки операций, знаки препинания. В свою очередь, лексемы являются частью выражений; а из выражений составляются инструкции и операторы.

**1.3 Применяемые сепараторы**

Применяемые сепараторы в языке FSI-2020 описаны в таблице 1.2.

Таблица 1.2 ­– Сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение сепаратора |
| ; | Разделитель инструкций. |
| { } | Программный блок, блок условной конфигурации, цикла. |
| ( ) | Параметры;  приоритетность операций. |
| (пробел) | Служит для разделения слов. |
| , | Разделитель параметров в функции. |
| +, -, \*, %, / | Операторы для целочисленных операций. |
| \n | Переход на новую строку. |
| \t | Табуляция. |

Сепараторы используются для определения конца лексемы и начала следующей.

**1.4 Применяемые кодировки**

При написании исходного кода на языке программирования FSI-2020 использовалась кодировка ASCII рисунок 1.1.

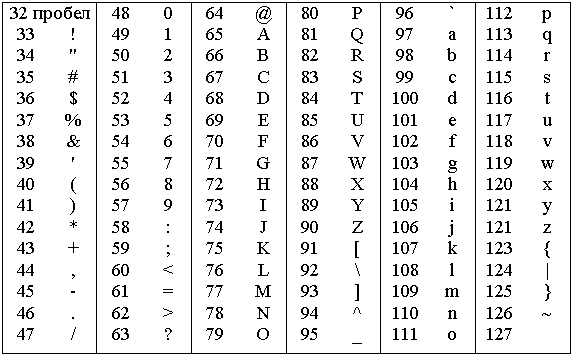


Рисунок 1.1 ­– Кодировка ASCII

На подавляющем большинстве современных компьютеров, минимально адресуемая единица памяти — байт. Обычно символ ASCII расширяют до 8 бит, просто добавляя один нулевой бит в качестве старшего.

**1.5 Типы данных**

В языке программирования FSI-2020 существует 4 типа данных: целочисленный, символьный, строковый, логический, которые представлены в таблице 1.3.

Таблица 1.3 – Типы данных

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| totum | Фундаментальный тип данных. Предусмотрен для объявления целочисленных положительных данных (4 байта).  Автоматически инициализируется нулевым значением.  Возможные операции:  *арифметические*  + – бинарный, суммирование;  - – бинарный, вычитание;  \* – бинарный, умножение;  / – бинарный, умножение;  % – бинарный, деление по модулю:  = – присваивание значения;  Максимальное значение: 2147483647. Минимальное: -2147483648. |

Продолжение таблицы 1.3

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| signum | Фундаментальный тип данных. Предусмотрен для объявления символов. Автоматическая инициализация пустым символом. |
| chorda | Фундаментальный тип данных. Предусмотрен для объявления строк. Автоматическая инициализация строкой нулевой длины. Максимальное количество символов в строке – 50. |
| logicus | Фундаментальный тип данных. Может принимать два возможных значения: истинна (verum) и ложь (falsus). Автоматическая инициализация значением falsus. |

Тип определяет возможные значения и их смысл, операции, а также способы хранения значений типа.

**1.6 Преобразование типов данных**

Преобразование типов данных не поддерживается.

**1.7 Идентификаторы**

Имена идентификаторов допускают наличие латинского алфавита нижнего и верхнего регистра. Максимальная длина 5 символов. Имя идентификатора не может содержать цифры. При превышении длины имени идентификатора будет выведена ошибка. Идентификатор не может совпадать с ключевыми словами.

**1.8 Литералы**

Язык программирования FSI–2020 допускает 4 типа литералов: целого, строкового, символьного и логического представленных в таблице 1.4.

Таблица 1.4 – Литералы

|  |  |
| --- | --- |
| Тип литерала | Описание литерала |
| Литералы целого типа | Могут представляться в виде восьмеричной и двоичной системах счислений.  Литерал в двоичной системе счисления должен состоять только из 0 и 1. Причем, если старший бит будет установлен в 1, литерал будет иметь отрицательное значение и этот бит не будет учитываться в подсчете значения.  Литерал в восьмеричной системе счисления должен заканчиваться латинским символом «o», и до него содержать только цифры из диапазона [0…7]. Причем, если старший разряд будет установлен в значение 7, литерал будет иметь отрицательное значение и старший разряд не будет учитываться в подсчете значения. |

Продолжение таблицы 1.4

|  |  |
| --- | --- |
| Тип литерала | Описание литерала |
| Литералы строкового типа | Состоит из символов, заключенных в “…” (кавычки). Максимальное число которых не может превышать . В случае превышения длины литерала работа транслятора прекращается. Литералы могут содержать только символы, определенные алфавитом языка. |
| Литералы символьного типа | Могут содержать только один символ, заключенный в ‘…’ (одинарные кавычки). Литералы могут содержать только символы, определенные алфавитом языка. |
| Литералы логического типа | Имеется только два литерала логического типа: verum (true), falsus (false). |

Литералы представляют собой константы, включаемые непосредственно в текст программы. В отличие от прочих элементов языка, литералы не могут быть изменены в тексте программы.

**1.9 Объявления данных и область видимости**

Область видимости сделана так, что переменную следует объявить до её использования. Блок функции является областью видимости переменной. Область видимости построена на подобии области видимости в языке C++ (сверху вниз). Объявление данных начинается с ключевого слова annuntiate, указывается тип данных и имя идентификатора. Для объявления переменных используется следующая конструкция: annuntiate ˽ <типданных> ˽ <идентификатор>;

Имена всех идентификаторов должны быть уникальны независимо от их области видимости.

**1.10 Инициализация данных**

Выполнение присваивания при объявлении в языке не поддерживается. Возможно выполнить только присваивание.

**1.11 Инструкции языка**

В языке программирования FSI-2020 применяются инструкции, изображенные в таблице 1.5.

Область видимости переменных, объявленных в блоке условной конструкции или цикле, принадлежит вышестоящей функции, в которой они находятся.

Цикл может содержать в качестве параметра только целочисленный литерал в диапазоне от нуля до 65535. Литерал будет являться показателем того, сколько раз будут выполнены инструкции, находящиеся в его блоке.

Условный оператор может содержать в качестве параметра только логический идентификатор или литерал. При принятии параметром значения истины внутренний блок оператора будет выполнен.

Таблица 1.5 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция языка | Синтаксис |
| Объявление переменной | annuntiate˽<тип данных>˽<идентификатор>; |
| Объявление функции | <тип данных>˽munus˽<идентификатор>(параметры){тело функции}; |
| Присваивание | <идентификатор> = <литерал>;  <идентификатор> = <выражение>;  <идентификатор 1> = <идентификатор 2>; |
| Вызов функций | <идентификатор функции>(<идентификатор, литерал>,…,<идентификатор, литерал>); |
| Вывод данных | loquor <идентификатор-переменная> | <литерал>; |
| Возврат из функции | reditus <идентификатор-переменная>| <литерал>; |
| Условная конструкция | conditio(<logicus идентификатор> | <logicus литерал>)  {…}; |
| Цикл | repeat(<totum литерал>)  {  …  }; |

Инструкция в языке FSI-2020 — наименьшая автономная часть языка программирования. Программа обычно представляет собой последовательность инструкций.

**1.12 Операции языка**

Операция сложения применима к целочисленным типам данных.

Наибольшую приоритетность арифметических операций имеют операции умножения и деления, а сложение и вычитание меньшую. Операции в языке программирования FSI-2020:

* (), приоритетность операций;
* +, сложение;
* \*, умножение;
* /, деление;
* %, деление по модулю.

При одинаковом приоритете первой выполнится операция, расположенная левее. Изменить приоритетность можно с помощью круглых скобок.

В качестве операндов могут выступать только целочисленные литералы или переменные идентификаторы.

**1.13 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.

**1.14 Программные конструкции языка**

Конструкции языка FSI-2020 приведены в таблице 1.7.

Таблица 1.7 – Конструкции языка

|  |  |
| --- | --- |
| Конструкция | Представление в языке |
| Главная функция | pelagus  {  <инструкции языка>  } |
| Функция | <тип данных>˽munus˽<идентификатор>(<тип данных>˽<идентификатор>, …)  {  <инструкции языка>  reditus˽<идентификатор>|<литерал>;  } |
| Точка входа | pelagus |

Конструкции языка используются для управления процессом вычисления.

**1.15 Область видимости идентификаторов**

Имена всех идентификаторов должны быть уникальны независимо от их области видимости.

Идентификаторы, кроме идентификаторов функции, обязательно должны быть объявлены внутри тела какой-нибудь функции и все они являются локальными. Параметры видны только в теле функции, в которой они были объявлены.

**1.16 Семантические проверки**

Семантика изучает смысловое значение единиц языка. Семантические проверки языка FSI-2020 представлены в таблице 1.8.

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Проверка |
| 1 | Наличие функции pelagus, как точки входа в программу. |
| 2 | Наличие только одной точки входа. |
| 3 | Переопределение идентификаторов. |
| 4 | Использование идентификаторов без их объявления. |
| 5 | Правильность возвращаемого типа. |
| 6 | Правильность передаваемых в функцию параметров: количество, типы. |
| 7 | Правильность построения условной конструкции и цикла. |
| 8 | Присваивание соответствующему типу. |
| 9 | Корректность операндов в операциях и выражениях. |

Окончание таблицы 1.8

|  |  |
| --- | --- |
| Номер | Проверка |
| 10 | Нахождение в своей области видимости. |
| 11 | Правильность построения имени идентификатора. |
| 12 | Деление на ноль. |

Семантические проверки являются «смысловыми» проверками входного текста, написанного на языке программирования FSI-2020.

**1.17 Распределение оперативной памяти на этапе выполнения**

Все переменные размещаются в стеке.

**1.18 Стандартная библиотека и её состав**

В языке FSI-2020 присутствует стандартная библиотека, которая автоматически подключается при трансляции исходного кода в язык ассемблера. У каждого типа данных есть свои функции реализующие различные команды. Функции стандартной библиотеки приведены в таблице 1.9.

Таблица 1.9 – Библиотечные функции

|  |  |
| --- | --- |
| Функция | Описание |
| totum randomness(); | Целочисленная функция. Генерирует случайное число от 0 до 1000. |
| totum quadtum(totum); | Целочисленная функция. Вычисляет и возвращает квадрат числа. |
| totum transformatio(signum); | Символьная функция. Преобразовывает букву в ее числовой код. |
| outChar; | Вывод символьного типа в консоль. |
| outString; | Вывод строкового типа в консоль. |
| outInt; | Вывод целочисленного типа в консоль. |
| outBool; | Вывод логического типа в консоль. |

Последние 4 функции в таблице 1.9 используются для вывода фундаментальных типов данных FSI-2020 в консоль. Причём функция «outString» реализована на языке ассемблера.

**1.19 Ввод и вывод данных**

Ввод данных не реализован.

Вывод данных осуществляется с помощью ключевого слова loquor. В качестве аргумента может выступать числовые, строковые, символьные и логические литералы или идентификаторы, причем аргументом не может являться вызов функции. Для вывода данных используются следующие конструкции:

– loquor<идентификатор-переменная>;

– loquor<литерал>;

Каждый вывод данных осуществляется с новой строки.

**1.20 Точка входа**

В языке FSI-2020 точкой входа является ключевое слово “pelagus”. Точка входа не может отсутствовать. Не может быть две точки входа.

**1.21 Препроцессор**

Препроцессор в языке FSI-2020 не предусмотрен.

**1.22 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.

**1.23 Объектный код**

Язык транслируется в язык ассемблера, а далее – в объектный код.

**1.24 Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке FSI-2020 и выявления её транслятором в файл протокола (log) или консоль выводится соответствующее сообщение.

**1.25 Контрольный пример**

Контрольный пример, демонстрирующий главные особенности языка FSI-2020, представлен в Приложении А.

**Глава 2. Структура транслятора**

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

Исходный код, написанный на языке программирования FSI-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.



Рисунок 2.1 – Структура транслятора FSI-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 очередь синтаксического анализатора, к нему на вход поступает таблица лексем, полученная на этапе лексического анализа. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки, формирует дерево разбора. Если программа построена синтаксически правильно, то осуществляется переход к этапу семантического анализа, стоящему далее, при ином раскладе работа транслятора останавливается.

Если на этапе семантического анализа ошибки не были выявлены, то идет переход к генерации кода, иначе работа транслятора останавливается.

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 FSI-2020, прошедший все предыдущие этапы, в код на языке Ассемблера.

**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: | Указывает им выходного файла. Если не указан явно, то имя протокола не формируется. | Не обязательный |
| -log: | Указывает имя протокола. Если не указан явно, то имя протокола формируется конкатенацией расположения файла исполнения «.exe» исходного кода и постфикса «.log» | Не обязательный |

Параметр -out в программе не используется.

**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 находятся информация о входных параметрах, количестве символов исходного кода, таблицы лексем, идентификаторов, работы синтаксического анализатора, дерево разбора.

# **Глава 3. Разработка лексического анализатора**

**3.1 Структура лексического анализатора**

Поскольку лексический анализатор представляет собой первую фазу компилятора, его основная задача состоит в чтении входных символов исходной программы, их группировании в лексемы и вывод последовательностей токенов для всех лексем исходной программы [2]. Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 – Структура лексического анализатора

Исходный код на языке FSI-2020 является входными данными.

Таблицы лексем и идентификаторов являются выходными данными.

**3.2 Контроль входных символов**

Таблица допустимости представлена на рисунке 3.2.

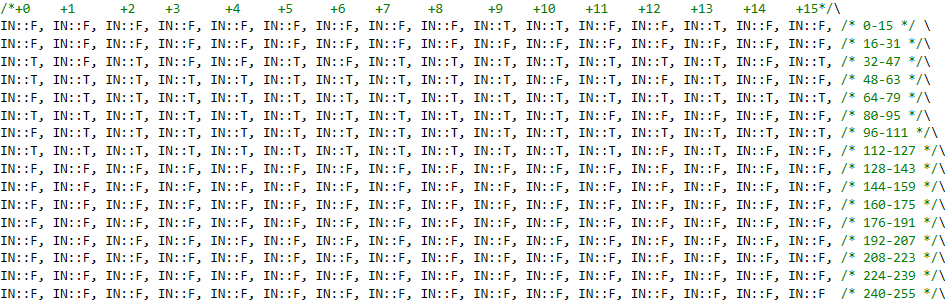


Рисунок 3.2 – Таблица допустимости входных символов

Таблица допустимости была сформирована на основе кодировки ASCII. Таблица необходима для проверки входных символов на допустимость. Символы могут быть разрешенными, запрещенными, игнорируемыми. В таблице записаны различные числовые значения, соответствующие символам в данной таблице:

– «T», разрешенные алфавитом символы;

– «F», запрещенные алфавитом символы;

– «I», игнорируемые алфавитом символы;

**3.3 Удаление избыточных символов**

Избыточными символами представлены пробелы, символы перехода на новую строку, табуляции.

Происходит удаление избыточных символов, а после разбиение текста на слова и лексемы.

## **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 |
| totum | n |
| chorda | s |
| logicus | b |
| signum | j |
| loquor | p |
| randomness | x |
| transformation | y |
| quadrum | q |
| conditio | u |
| munus | f |
| annuntiate | d |
| reditus | r |
| pelagus | m |
| repeat | z |
| + | v |
| - | v |
| \* | v |
| / | v |
| % | v |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| Цепочка | Лексема |
| ( | ) |
| ) | ( |
| , | , |
| ; | ; |
| { | { |
| } | } |
| Идентификатор | i |
| Числовой литерал в восьмеричной форме | l |
| Числовой литерал в двоичной форме | l |
| Символьный литерал | l |
| Строковый литерал | l |
| Логический литерал | l |
| = | = |

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. Если выражение не получается разобрать ни по одной из цепочек конечного автомата, оно идентифицируется как идентификатор. Пример реализации таблицы лексем представлен в приложении А.

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

За основу названий ключевых слов был взят латинский язык.

**3.5 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных языка FSI-2020, используемых для хранения, представлены в приложении А. В таблице лексем содержится лексема, номер строки в исходном коде, индекс таблицы идентификаторов и область видимости. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора и его значение.

Таблицы лексем и идентификаторов записываются в текстовый файл «Tables», который расположен в той же директории, что и файл загрузочного модуля (exe).

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

Лексический анализ — процесс аналитического разбора входной последовательности символов на распознанные группы — лексемы — с целью получения на выходе идентифицированных последовательностей, называемых «токенами»

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

Таблица 3.2 – Ошибки, выявляемые на этапе лексического анализа

|  |  |
| --- | --- |
| Номер | Сообщение |
| 1 | Превышение размера идентификатора. |
| 2 | Нераспознанный тип идентификатора. |
| 3 | Стек видимости не опустел при окончании разбора лексем. Возможно, вы забыли добавить '}'. |
| 4 | Стек видимости опустел при разборе лексем. Возможно, вы добавили лишнюю '}'. |
| 5 | Превышен допустимый размер литерала. |
| 6 | Литерал не может быть пустым. |

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 113-118. Также сам текст ошибки содержит в себе префикс «[Lex]».

**3.7 Принцип обработки ошибок**

В случае обнаружения критической ошибки, которая не позволяет работать анализаторам или генератору правильно функционировать, транслятор прекращает свою работу и в log-файл записывается ошибка. При наличии слова несовпадающим ни с ключевым словом, ни с литералом он идентифицируется анализатором как идентификатор, причем если длина идентификатора содержит более 5 символов будет прекращена дальнейшая работа транслятора. Некоторые ошибки семантического профиля были реализованы в лексическом анализаторе.

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

Входными параметрами для лексического анализатора являются таблица лексем и таблица идентификаторов.

Входные параметры транслятора языка FSI-2020 представлены в таблице 3.3.

Таблица 3.3 Входные параметры транслятора языка FSI-2020

|  |  |
| --- | --- |
| Входной параметр | Описание |
| -lexTable | Параметр для вывода таблицы лексем в текстовый файл таблиц |
| -idTable | Параметр для вывода таблицы идентификаторов в текстовый файл таблиц |

## Входные параметры используются для вывода результата работы лексического анализатора и выступают в роли входных параметров для последующих анализов.

## **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 языке FSI-2020 в случае, если автомат не был подобран, слово идентифицируется как идентификатор. Если идентификатор имеет длину более 5 символов будет вызвана соответствующая ошибка и дальнейший разбор прекратится. Если токен разобран, то дальнейшие действия, которые будут с ним производиться, будут зависеть от того, чем он является. Если токен является литералом или идентификатором, то он заносится в таблицу идентификаторов с соответствующей информацией.

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, то он заносится в таблицу идентификаторов со своим именем.

В случае, если является литералом, то заносится в таблицу идентификаторов с именем «\_L<n>», где n является номером литерала.

Когда встречаем токен, являющийся ключевым словом, которое отвечает за тип данных, заносим лексему, соответствующую ему, в таблицу лексем и запоминаем его область видимости для последующего анализа.

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

Граф конечного автомата для этой лексемы представлен на рисунке 3.3. S0 – начальное состояние, S4 – конечное состояние автомата.

m

u

t

o

t

Рисунок 3.3 – Граф переходов для цепочки «totum»

## **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.**

Вторая фаза работы компилятора называется синтаксическим анализом, назначением которой является распознавание синтаксических конструкций языка и формирование промежуточного кода. Правила языка программирования описываются с помощью контекстно свободных грамматик (тип 2 иерархии Хомского). Программа, выполняющая синтаксический анализ, называется синтаксическим анализатором. Исходными данными синтаксического анализатора являются таблицы лексем и идентификаторов. Лексемы являются для синтаксического анализатора терминальными символами контекстно свободной грамматики. Результат работы синтаксического анализатора – дерево разбора (промежуточное представление кода). Распознавателями для контекстно-свободных языков являются односторонние недетерминированные распознаватели с ограниченной магазинной памятью (МП-автоматы).

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**

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

T – множество терминальных символов,

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 FSI-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 FSI-2020

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | nfi(F){NrL;};S  sfi(F){NrL;};S  bfi(F){NrL;};S  jfi(F){NrL;};S  m{NrL;};S  m{NrL;} | Порождает правила, описывающее общую структуру программы. |
| N | dni;N  dni;  dsi;N  dsi;  dji;N  dji;  dbi;N  dbi;  i=E;  i=E;N  u(L){N};  u(L){N};N  u(L){};  u(L){};N  z(L){N};  z(L){N};N  z(L){};  z(L){};N  pL;N  pL; | Порождает правила, описывающие конструкции языка. |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| E | i  iM  l  lM  (E)  (E)M  i(W)  x()  y(i)  y(l)  q(i)  q(l) | Порождает правила, описывающие выражения. |
| F | ni  ni,F  si  si,F  ji  ji,F  bi  bi,F | Порождает правила, описывающие параметры локальной функции при её объявлении. |
| W | i  l  i,W  l,W | Порождает правила, описывающие принимаемые параметры функции. |
| L | i  l | Порождает правила, описывающие подстановку литералов и идентификаторов после ключевых слов, где это необходимо. |

## Приведенные лексемы в цепочках правил в таблице 4.1 были подробно описаны в разделе 3.4.

## **4.3 Построение конченого магазинного автомата**

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

Конечный автомат с магазинной памятью представляет собой семерку. Структура данного автомата показана в приложении Б.

Принцип действия конечного магазинного автомата представлен на рисунке 4.2.

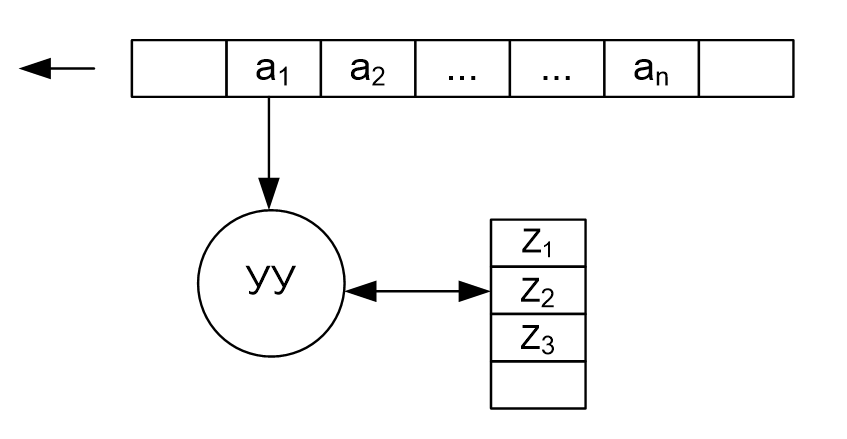
****

Рисунок 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**

## Основные структуры данных синтаксического анализатора включают в себя структуру магазинного конечного автомата и структуру грамматики Грейбах, опи-сывающей правила языка FSI-2020. Данные структуры представлены в приложе-нии Б.

## **4.5 Описание алгоритма синтаксического разбора**

На входе конченого автомата имеется лента, на которой находятся входные символы, а также грамматика в форме Грейбах.

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**

При возникновении ошибки будет выполнена программно-реализованная диагностика.

Перечень сообщений, генерируемых на этапе синтаксического анализа, представлен в таблице 4.2.

Таблица 4.2 – Ошибки, выявляемые на этапе синтаксического анализа

|  |  |
| --- | --- |
| Номер | Сообщение |
| 1 | Неверная структура программы |
| 2 | Ошибочный оператор |
| 3 | Ошибка в выражении |
| 4 | Ошибка в операторах выражений |
| 5 | Ошибка в параметрах функции |
| 6 | Ошибка в параметрах вызываемой функции |
| 7 | Синтаксическая ошибка |

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 600-606. Также сам текст ошибки содержит в себе префикс «[SYN]».

## **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 в протокол работы – log.

## **4.8 Принцип обработки ошибок**

При обнаружении ошибки в цепочке какого-либо правила, синтаксического анализатора идет вверх по дереву разбора, пока не найдет верный вариант. Иначе проводится диагностика ошибок и вывод соответствующего сообщения.

## **4.9 Контрольный пример**

Пример начала и конца разбора синтаксическим анализатором исходного кода на языке FSI-2020 (Лист. 4.1).

|  |
| --- |
| Шаг: Правило Входная лента Cтeк  0 : S->nfi(F){NrL;};S nfi(ni){i=y(l);ri;};m{dsi S$  0 : SAVESTATE: 1  0 : nfi(ni){i=y(l);ri;};m{dsi nfi(F){NrL;};S$  1 : fi(ni){i=y(l);ri;};m{dsi; fi(F){NrL;};S$  2 : i(ni){i=y(l);ri;};m{dsi;d i(F){NrL;};S$  3 : (ni){i=y(l);ri;};m{dsi;db (F){NrL;};S$  4 : ni){i=y(l);ri;};m{dsi;dbi F){NrL;};S$  5 : F->ni ni){i=y(l);ri;};m{dsi;dbi F){NrL;};S$  5 : SAVESTATE: 2  …  904 : L->l l;}; L;};$  904 : SAVESTATE: 58  904 : l;}; l;};$  905 : ;}; ;};$  906 : }; };$  907 : ; ;$  908 : $  909 : LENTA\_END  910 : ------>LENTA\_END  -------------------------------------------------------------------------- |

Листинг 4.1 – Начало и конец разбора синтаксического анализатора

Дерево разбора исходного кода также представлено в приложении Б.

# **Глава 5. Разработка семантического анализатора**

## **5.1 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.

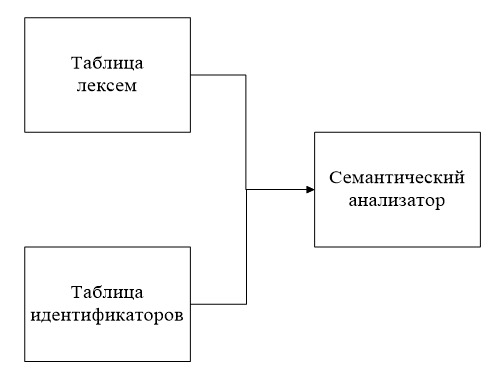


Рисунок 5.1 – Структура семантического анализатора

## Семантический анализ происходит при выполнении фазы лексического анализа и реализуется в виде отдельных проверок текущих ситуаций в конкретных случаях

## **5.2 Функции семантического анализатора**

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

Функции, представляющие проверку правил, представлены в таблице 5.1.

Таблица 5.1 – Семантические проверки

|  |  |
| --- | --- |
| Функция | Описание |
| CheckMain | Проверка на наличие входной точки или присутствии более одной точки входа |
| CheckId | Проверка на переопределение идентификаторов |
| Parametrs | Проверка типов и количества параметров функций |
| Ret | Проверка на соответствие возвращаемого значения |
| Vision | Проверка на область видимости идентификаторов |
| FirstLetterId | Проверка на наличие цифры в начале имени идентификатора |
| Operation | Проверка на соответствие операндов в арифметических операциях |
| Appropriation | Проверка на соответствие соответствующего идентификатора при присваивании |
| LibraryParametrs | Проверка параметров библиотечных функций, а также условного оператора и цикла |
| DivisionByZero | Проверка на явное деление на ноль |

Функции вызываются последовательно в порядке, указанном в таблице.

Функция LibraryParametrs проверяет соответствие количество и типы параметров библиотечных функций, а также условной конструкции и цикла. Например, в качестве параметра условной конструкции может быть только логический литерал или переменная.

**5.3 Структура и перечень сообщений семантического анализатора**

Перечень сообщений, генерируемых на этапе семантического анализа, представлен в таблице 4.2.

Таблица 4.2 – Ошибки, выявляемые на этапе семантического анализа

|  |  |
| --- | --- |
| Номер | Сообщение |
| 1 | Нарушение области видимости идентификаторов |
| 2 | Нет точки входа |
| 3 | pelagus (main) должен быть только один |
| 4 | Ошибка возвращаемого типа |
| 5 | Неверное количество параметров функций |
| 6 | Ошибка в условном операторе или цикле |
| 7 | Неверный тип параметра в функции quadrum |
| 8 | Неверный тип параметра в функции transformation |
| 9 | Неверные типы параметров в вызываемой функции |
| 10 | Попытка присваивания не соответствующему типу |

Окончание таблицы 4.2

|  |  |
| --- | --- |
| Номер | Сообщение |
| 11 | Операндом целочисленной операции может быть только целочисленная переменная или литерал |
| 12 | Имя идентификатора не может начинаться с цифры |
| 13 | Переопределение идентификатора |
| 14 | Явное деление на ноль |

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 400-413. Также сам текст ошибки содержит в себе префикс «[SEM]».

Ошибка под номером 14 будет вызываться тогда, когда после операции деления нацело или деления по модулю будет идти литерал, который будет равняться нулю

## **5.4 Принцип обработки ошибок**

Семантический анализатор вызывает последовательно функции ответственные за определенные заданные проверки. При возникновении хотя бы одной ошибки анализатор прекращает дальнейшую работу и последующие функции вызваны не будут.

В случае обнаружения ошибки, работа транслятора прекращается, сообщение об ошибке выводится в протокол работы транслятора. Переход к генерации кода не будет осуществлен.

**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. Вычисление выражений**

**6.1 Выражения, допускаемые языком**

В языке FSI-2020 для выражений с использованием целочисленных идентификаторов и литералов используются следующие арифметические операции:

* сложения: +;
* вычитания: -;
* умножения: \*;
* деления: /;
* деления по модулю: %.

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

* (;
* );

Приоритетность операций представлена в таблице 6.1.

Таблица 6.1 – Приоритетность операций

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

В качестве операндов могут выступать только целочисленные литералы и идентификаторы переменные.

## **6.2 Польская запись и принцип её построения**

Выражения в языке FSI-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 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 «@».

## **6.3 Программная реализация обработки выражения**

В языке FSI-2020 после этапов лексического, синтаксического и семантического анализов происходит преобразование в польскую запись. Польская запись реализована на основании алгоритма, описанного в пункте 6.2. Программная реализация алгоритма преобразования выражений к польской записи представлена в приложении В.

## **6.4 Контрольный пример**

Пример разбора выражения из контрольного примера показан на рисунке 6.2.

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| (x+y)\*z |  |  |
| x+y)\*z |  | ( |
| +y)\*z | x |  |
| y)\*z | x | ( + |
| )\*z | x y | ( + |
| \*z | x y + |  |
| z | x y + | \* |
|  | x y + z | \* |
|  | x y + z \* |  |

В языке FSI-2020 происходит преобразование в польскую запись после этапов лексического, синтаксического и семантического.

# **Глава 7. Генерация кода**

## **7.1 Структура генератора кода**

Генерация кода – четвертая последняя фаза работы транслятора. Исходными данными для генератора кода является промежуточное представление исходной программы.

Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. Формируется 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 FSI-2020 представлена на рисунке 7.1.



Рисунок 7.1 – Структура генератора кода

Одной из основных задач на этапе генерации кода является планирование памяти для переменных, литералов. Решение о распределении памяти принимается либо в процессе генерации промежуточного кода, либо при генерации целевого кода.

**7.2 Представление типов данных в оперативной памяти**

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Идентификаторы языка FSI-2020 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке FSI-2020 и на языке ассемблера приведены в таблице 7.1.

Представление строковых идентификаторов в сегменте данных представлены таким образом, что идентификатор будет хранить в себе 32 бита – адрес смещения на начал строки.

Таблица 7.1 – Структура генератора кода

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке FSI-2020 | Тип идентификатора на языке ассемблера | Хранимые данные |
| totum | sdword | Хранит целочисленный тип данных. |
| chorda | dword | Хранит адрес смещения на начало строки. |
| signum | dword | Хранит символ. |
| logicus | dword | Хранит значение 0 или 1. |

В MASM определены несколько внутренних типов данных, значения которых могут быть присвоены переменным, либо они могут являться результатом выполнения выражения. Например, в переменной типа dword можно хранить любое 32- разрядное целое значение. Однако на некоторые типы накладываются более жесткие ограничения. При определении данных должен быть указан хотя бы один инициализатор, даже если переменной не назначается какого-то конкретного значения. Все дополнительные инициализаторы перечисляются через запятую. Для целочисленных типов данных инициализатор является целочисленной константой или выражением, значение которого соответствует размеру определяемых данных (byte, word, и т.д.) [3].

## **7.3 Статическая библиотека**

В языке FSI-2020 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++ и ассемблера. Объявление функций статической библиотеки генерируется автоматически. Также в стандартной библиотеке реализованы функции для манипулирования выводом. Эти функции представлены в таблице 1.9.

**7.4 Особенности алгоритма генерации кода**

Алгоритм генерации кода выглядит следующим образом:

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

|  |
| --- |
| .586  .model flat, stdcall  includelib ../Debug/StaticLib.lib  randomness PROTO  transformatio PROTO: DWORD  quadrum PROTO: SDWORD  outInt PROTO: SDWORD  outChar PROTO: DWORD  outString PROTO: DWORD, :DWORD  outBool PROTO: SDWORD  ExitProcess PROTO : DWORD |

Листинг 7.1 – Заголовочная информация

2) Проходим полностью таблицу идентификаторов и заполняет поле .const литералами (Лист. 7.2).

|  |
| --- |
| .const  \_L2 dword 'E', 0  \_L7 byte "collapse/chaos", 10, 0  \_L8 dword 1  \_L9 sdword -3  \_L10 sdword -2  \_L11 byte "one", 10, 0  \_L12 sdword 3  \_L13 sdword 2  \_L14 byte "two", 10, 0  \_L15 sdword 9  \_L16 sdword 5  \_L17 sdword 3  \_L18 sdword 10  \_L19 sdword 0 |

Листинг 7.2 – Сегмент констант

3) Проходим таблицу лексем и ищем объявление переменных и объявляем данные переменные в соответствующем поле .data (Лист. 7.3).

|  |
| --- |
| .data  a dword 0  z dword 0  one sdword 0  two sdword 0  counter sdword 0 |

Листинг 7.3 – Сегмент данных

4) Снова идём по таблице лексем и начинаем описывать функции и конструкции, написанные на языке FSI-2020. В случае, если встречаем лексему «=», описываем вычисление выражения или присваиваем переменной данные.

**7.5 Входные параметры генератора кода**

Входными параметрами генератора кода языка FSI-2020 являются таблица лексем и таблица идентификаторов.

**7.6 Контрольный пример**

Генерируемый код записывается в файл «asm.asm». Сгенерированный код можно посмотреть в приложении Г.

**Глава 8. Тестирование транслятора**

**8.1 Общие положения**

Следующий пункт показывает контрольный пример, демонстрирующий правильную работу компилятора. Выполняется подбор тестов. При обнаружении ошибки в консоль выводится соответствующее сообщение вместе ссылкой на полную информацию в файле протокола.

**8.2 Результаты тестирования**

В данной главе описаны возможные ошибки, возникающие на различных этапах работы транслятора. Ошибки записываются в log. Тесты, показанные в таблице 8.1 показывают работу транслятора на различных этапах транслятора.

Таблица 8.1 ­– Результаты тестирования транслятора

|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| pelagus{л;}; | Ошибка 111: [SYS] Недопустимый символ в исходном файле (-in) |
| pelagus{annuntiate totum aaaaaa;}; | Ошибка 113: [LEX] Превышен размер идентификатора |
| pelagus{loquor aa;}; | Ошибка 114: [LEX] Нераспознанный тип идентификатора |
| pelagus{loquor ‘ss’;}; | Ошибка 117: [LEX] Превышен допустимый размер литерала |
| pelagus{loquor “”;}; | Ошибка 118: [LEX] Литерал не может быть пустым |
| totum fa(){annuntiate totum a;};  pelagus{annuntiate totum a;  reditus 1}; | Ошибка 400: [SEM] Нарушение области видимости идентификаторов |
| totum fa(){annuntiate totum a;  reditus a}; | Ошибка 401: [SEM] Нет точки входа |
| pelagus{};  pelagus{}; | Ошибка 402: [SEM] pelagus(main) должен быть только один |
| pelagus{annuntiate totum a;  annuntiate signum a}; | Ошибка 403: [SEM] Переопределение идентификатора |
| pelagus{reditus ‘s’}; | Ошибка 404: [SEM] Ошибка возвращаемого типа |
| totum fa(){annuntiate totum a;  reditus a;};  pelagus{annuntiate totum a;  a = fa();  reditus 1}; | Ошибка 405: [SEM] Неверное количество параметров функции |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| pelagus{conditio(01){};  reditus 1}; | Ошибка 406: [SEM] Ошибка в условном операторе или цикле |
| pelagus{quadrum(‘s’);  reditus 1}; | Ошибка 407: [SEM] Неверный тип параметра в функции quadrum |
| pelagus{transformatio(“ss”);  reditus 1}; | Ошибка 408: [SEM] Неверный тип параметра в функции transformatio |
| totum fa(totum s){redirus s};  pelagus{annuntiate totum a;  a = fa(‘s’);  reditus 1}; | Ошибка 409: [SEM] Неверные типы параметров в вызываемой функции |
| pelagus{annuntiate totum a;  a = 6o/0;  reditus 1}; | Ошибка 410: [SEM] Явное деление на ноль |
| pelagus{annuntiate totum a;  a = “nein”;  reditus 1}; | Ошибка 411: [SEM] Попытка присваивания не соответствующему типу |
| pelagus{annuntiate totum a;  annuntiate signum c;  c = ‘s’;  a = 6o/c;  reditus 1}; | Ошибка 412: [SEM] Операндом целочисленной операции может быть только целочисленная переменная или литерал |

При возникновении ошибки дальнейшая работа транслятора прекращается, в консоль выводится сообщение об ошибке, а в файл протокола подробная информация.

**Заключение**

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

В языке программирования FSI-2020:

* реализованы 5 арифметических операторов для вычисления выражений;
* реализованы четыре типа данных и соответствующие им литералы;
* поддерживается оператор вывода;
* реализованы 7 функций стандартной библиотеки;
* возможно создание и вызов пользовательских функций;
* реализована генерация кода на язык ассемблера;
* обрабатывается 32 возможных исключительных ситуаций;
* реализован условный оператор;
* реализован цикл.

**Список использованных источников**

1 Вирт. Н. Построение Компиляторов / Пер. с англ. Борисов Е.В., Чернышов Л.Н. – М.: ДМК Пресс, 2010. – 192 с.

2 Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.

3 Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с

ПРИЛОЖЕНИЯ

Контрольный пример

|  |
| --- |
| totum munus fa(totum s)  {  s = transformatio('E');  reditus s;  };  pelagus  {  annuntiate chorda a;  annuntiate logicus z;  annuntiate totum one;  annuntiate totum two;  a = "collapse/chaos";  one = fa(one);  loquor one;  z = verum;  one = 111;  conditio(z){  one = quadrum(110);  };  loquor "one";  one = (one + 3o) \* 010;  loquor one;  loquor "two";  two = one - 01001 % 5o;  loquor two;  loquor a;  two = randomness();  repeat(011){  two = two / 12o;  loquor two;  };  reditus 0;  }; |

ПРИЛОЖЕНИЕ А

|  |
| --- |
| namespace LT  {  struct Entry // строка таблицы лексем  {  char lexema; // лексема  int sn = 0; // номер строки в исходном тексте  int idxTI; // индекс в таблице идентефикаторов  char\* lt\_vision; //обл видимости  char operat; //знак  int count; //номер лексемы  };  struct LexTable // экземпляр такблицы лексем  {  int maxsize; // емкость таблицы лексем < LT\_MAXSIZE  int size; // текущий размер таблицы лексем < MAXSIZE  Entry\* table; // массив строк таблицы лексем  };  LexTable Create( // создать таблицу лексем  int size // емкость таблицы лексем < LT\_MAXSIZE  );  void Add( // добавить строку в таблицу лексем  LexTable& lextable, // экземпляр таблицы лексем  Entry entry // строка таблицы лексем  );  Entry Getentry(LexTable& lextable, int n);  void SetEntry(LT::LexTable\* entry, unsigned char lexema, int indx, int line, int count, char\* vision, char operat);  void Delete(LexTable& lextable); // удалить таблицу лексем ( освободить память )  }; |

|  |
| --- |
| namespace IT  {  enum IDDATATYPE {  NON = 0,  INT = 1, //integer  STR = 2, //string  BOO = 3, //bool  CHR = 4, //char  };  enum IDTYPE {  N = 0, //ничего  V = 1, //переменная  F = 2, //функция  P = 3, //параметр  L = 4 //литерал  };  struct Entry // строка таблицы идентефикаторов  {  int idxfirstLE; // индекс первой строки в таблице лексем  char id[ID\_MAXSIZE]; //идентефикатор (автоматически усекается до ID\_MAXSIZE)  IDDATATYPE iddatatype; //тип данных  IDTYPE idtype;  union  {  int vint;  int vbool;  char vchar[4];  struct  {  char len;  char str[TI\_STR\_MAXSIZE - 1];  } vstr[TI\_STR\_MAXSIZE];  }value;  };  struct IdTable // экземпляр таблицы идентефикаторов  {  short lit = 0;  int maxsize; // емкость таблицы идентефикаторов < TI\_MAXSIZE  int size; // текущий размер таблицы идентефикаторов < maxsize  Entry\* table; // массив строк таблицы идентефикаторов  };  IdTable Create( // создать таблицу идентефикаторов  int size // емкость таблицы идентефикаторов < TI\_MAXSIZE  );  void Add( // добавить строку в таблицу идентефикаторов  IdTable& idtable, // экземпляр таблицы идентефикаторов  Entry entry // строка таблицы идентефикаторов  );  Entry Getentry(IdTable& idtable, int n);  void SetEntry(IdTable& entry, int idxfirstLE, char id[ID\_MAXSIZE], int line, IDDATATYPE iddatatype, IDTYPE idtype);  int IsId(IdTable& idtable, char id[ID\_MAXSIZE]);  void Delete(IdTable& idtable); // удалить таблицу идентефикаторов ( освободить память );  } |

|  |
| --- |
| ТАБЛИЦА ЛЕКСЕМ  Строка | Лексема | id в idtable | Область  1 n global  1 f global  1 i 0 global  1 ( fa  1 n fa  1 i 1 fa  1 ) fa  2 { fa  3 i 1 fa  3 = fa  3 l 2 fa  3 @ fa  3 ; fa  3 # fa  3 ; fa  4 r fa  4 i 1 fa  4 ; fa  5 } fa  5 ; global  6 m global  7 { pelagus  8 d pelagus  8 s pelagus  8 i 3 pelagus  8 ; pelagus  9 d pelagus  9 b pelagus  9 i 4 pelagus  9 ; pelagus  10 d pelagus  10 n pelagus  10 i 5 pelagus  10 ; pelagus  11 d pelagus  11 n pelagus  11 i 6 pelagus  11 ; pelagus  12 i 3 pelagus  12 = pelagus  12 l 7 pelagus  12 ; pelagus  13 i 5 pelagus  13 = pelagus  13 i 5 pelagus  13 @ 0 pelagus  13 ; pelagus  13 # pelagus  13 ; pelagus  14 p pelagus  14 i 5 pelagus  14 ; pelagus  15 i 4 pelagus  15 = pelagus  15 l 8 pelagus  15 ; pelagus  16 i 5 pelagus  16 = pelagus  16 l 9 pelagus  16 ; pelagus  17 u pelagus  17 ( pelagus  17 i 4 pelagus  17 ) pelagus  17 { pelagus  18 i 5 pelagus  18 = pelagus  18 l 10 pelagus  18 @ pelagus  18 ; pelagus  18 # pelagus  18 ; pelagus  19 } pelagus  19 ; pelagus  20 p pelagus  20 l 11 pelagus  20 ; pelagus  21 i 5 pelagus  21 = pelagus  21 i 5 pelagus  21 l 12 pelagus  21 v pelagus  21 l 13 pelagus  21 v pelagus  21 # pelagus  21 # pelagus  21 ; pelagus  22 p pelagus  22 i 5 pelagus  22 ; pelagus  23 p pelagus  23 l 14 pelagus  23 ; pelagus  24 i 6 pelagus  24 = pelagus  24 i 5 pelagus  24 l 15 pelagus  24 l 16 pelagus  24 v pelagus  24 v pelagus  24 ; pelagus  25 p pelagus  25 i 6 pelagus  25 ; pelagus  26 p pelagus  26 i 3 pelagus  26 ; pelagus  27 i 6 pelagus  27 = pelagus  27 @ pelagus  27 ; pelagus  27 # pelagus  27 ; pelagus  28 z pelagus  28 ( pelagus  28 l 17 pelagus  28 ) pelagus  28 { pelagus  29 i 6 pelagus  29 = pelagus  29 i 6 pelagus  29 l 18 pelagus  29 v pelagus  29 ; pelagus  30 p pelagus  30 i 6 pelagus  30 ; pelagus  31 } pelagus  31 ; pelagus  32 r pelagus  32 l 19 pelagus  32 ; pelagus  33 } pelagus  33 ; global |

|  |
| --- |
| ТАБЛИЦА ИДЕНТИФИКАТОРОВ    Индекс | Идентификатор | Тип | Значение  0 fa F totum 0  1 s P totum 0  2 \_L2 L signum 'E'  3 a V chorda ""  4 z V logicus falsus  5 one V totum 0  6 two V totum 0  7 \_L7 L chorda "collapse/chaos"  8 \_L8 L logicus verum  9 \_L9 L totum -3  10 \_L10 L totum -2  11 \_L11 L chorda "one"  12 \_L12 L totum 3  13 \_L13 L totum 2  14 \_L14 L chorda "two"  15 \_L15 L totum 9  16 \_L16 L totum 5  17 \_L17 L totum 3  18 \_L18 L totum 10  19 \_L19 L totum 0 |

|  |
| --- |
| #define RANDOMNESS 11, \  FST::NODE(1, FST::RELATION('r', 1)),\  FST::NODE(1, FST::RELATION('a', 2)),\  FST::NODE(1, FST::RELATION('n', 3)),\  FST::NODE(1, FST::RELATION('d', 4)),\  FST::NODE(1, FST::RELATION('o', 5)),\  FST::NODE(1, FST::RELATION('m', 6)),\  FST::NODE(1, FST::RELATION('n', 7)),\  FST::NODE(1, FST::RELATION('e', 8)),\  FST::NODE(1, FST::RELATION('s', 9)),\  FST::NODE(1, FST::RELATION('s', 10)),\  FST::NODE()  #define TRANSFORMATIO 14, \  FST::NODE(1, FST::RELATION('t', 1)),\  FST::NODE(1, FST::RELATION('r', 2)),\  FST::NODE(1, FST::RELATION('a', 3)),\  FST::NODE(1, FST::RELATION('n', 4)),\  FST::NODE(1, FST::RELATION('s', 5)),\  FST::NODE(1, FST::RELATION('f', 6)),\  FST::NODE(1, FST::RELATION('o', 7)),\  FST::NODE(1, FST::RELATION('r', 8)),\  FST::NODE(1, FST::RELATION('m', 9)),\  FST::NODE(1, FST::RELATION('a', 10)),\  FST::NODE(1, FST::RELATION('t', 11)),\  FST::NODE(1, FST::RELATION('i', 12)),\  FST::NODE(1, FST::RELATION('o', 13)),\  FST::NODE()  #define CHAR 7, \  FST::NODE(1, FST::RELATION('s', 1)),\  FST::NODE(1, FST::RELATION('i', 2)),\  FST::NODE(1, FST::RELATION('g', 3)),\  FST::NODE(1, FST::RELATION('n', 4)),\  FST::NODE(1, FST::RELATION('u', 5)),\  FST::NODE(1, FST::RELATION('m', 6)),\  FST::NODE()  #define LITERAL\_CHAR 4, \  FST::NODE(2, FST::RELATION('\'', 1), FST::RELATION('\'', 2)),\  FST::NODE(62, \  FST::RELATION('a', 2), FST::RELATION('b', 2), FST::RELATION('c', 2), FST::RELATION('d', 2), FST::RELATION('e', 2), FST::RELATION('f', 2),\  FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('i', 2), FST::RELATION('j', 2), FST::RELATION('k', 2), FST::RELATION('l', 2),\  FST::RELATION('m', 2), FST::RELATION('n', 2), FST::RELATION('o', 2), FST::RELATION('p', 2), FST::RELATION('q', 2), FST::RELATION('r', 2),\  FST::RELATION('s', 2), FST::RELATION('t', 2), FST::RELATION('u', 2), FST::RELATION('v', 2), FST::RELATION('w', 2), FST::RELATION('x', 2),\  FST::RELATION('y', 2), FST::RELATION('z', 2), FST::RELATION('1', 2), FST::RELATION('2', 2), FST::RELATION('3', 2), FST::RELATION('4', 2),\  FST::RELATION('5', 2), FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2), FST::RELATION('9', 2), FST::RELATION('0', 2),\  \  FST::RELATION('A', 2), FST::RELATION('B', 2), FST::RELATION('C', 2), FST::RELATION('D', 2), FST::RELATION('E', 2), FST::RELATION('F', 2),\  FST::RELATION('G', 2), FST::RELATION('H', 2), FST::RELATION('I', 2), FST::RELATION('J', 2), FST::RELATION('K', 2), FST::RELATION('L', 2),\  FST::RELATION('M', 2), FST::RELATION('N', 2), FST::RELATION('O', 2), FST::RELATION('P', 2), FST::RELATION('Q', 2), FST::RELATION('R', 2),\  FST::RELATION('S', 2), FST::RELATION('T', 2), FST::RELATION('U', 2), FST::RELATION('V', 2), FST::RELATION('W', 2), FST::RELATION('X', 2),\  FST::RELATION('Y', 2), FST::RELATION('Z', 2)), \  \  FST::NODE(1, FST::RELATION('\'', 3)),\  FST::NODE()  #define FOR 7, \  FST::NODE(1, FST::RELATION('r', 1)),\  FST::NODE(1, FST::RELATION('e', 2)),\  FST::NODE(1, FST::RELATION('p', 3)),\  FST::NODE(1, FST::RELATION('e', 4)),\  FST::NODE(1, FST::RELATION('a', 5)),\  FST::NODE(1, FST::RELATION('t', 6)),\  FST::NODE()  #define POW 8, \  FST::NODE(1, FST::RELATION('q', 1)),\  FST::NODE(1, FST::RELATION('u', 2)),\  FST::NODE(1, FST::RELATION('a', 3)),\  FST::NODE(1, FST::RELATION('d', 4)),\  FST::NODE(1, FST::RELATION('r', 5)),\  FST::NODE(1, FST::RELATION('u', 6)),\  FST::NODE(1, FST::RELATION('m', 7)),\  FST::NODE()  #define IF 9, \  FST::NODE(1, FST::RELATION('c', 1)),\  FST::NODE(1, FST::RELATION('o', 2)),\  FST::NODE(1, FST::RELATION('n', 3)),\  FST::NODE(1, FST::RELATION('d', 4)),\  FST::NODE(1, FST::RELATION('i', 5)),\  FST::NODE(1, FST::RELATION('t', 6)),\  FST::NODE(1, FST::RELATION('i', 7)),\  FST::NODE(1, FST::RELATION('o', 8)),\  FST::NODE()  #define WHILE 6, \  FST::NODE(1, FST::RELATION('r', 1)),\  FST::NODE(1, FST::RELATION('u', 2)),\  FST::NODE(1, FST::RELATION('r', 3)),\  FST::NODE(1, FST::RELATION('s', 4)),\  FST::NODE(1, FST::RELATION('u', 5)),\  FST::NODE(1, FST::RELATION('s', 5)),\  FST::NODE()  #define TRUE 6, \  FST::NODE(1, FST::RELATION('v', 1)),\  FST::NODE(1, FST::RELATION('e', 2)),\  FST::NODE(1, FST::RELATION('r', 3)),\  FST::NODE(1, FST::RELATION('u', 4)),\  FST::NODE(1, FST::RELATION('m', 5)),\  FST::NODE()  #define FALSE 7, \  FST::NODE(1, FST::RELATION('f', 1)),\  FST::NODE(1, FST::RELATION('a', 2)),\  FST::NODE(1, FST::RELATION('l', 3)),\  FST::NODE(1, FST::RELATION('s', 4)),\  FST::NODE(1, FST::RELATION('u', 5)),\  FST::NODE(1, FST::RELATION('s', 6)),\  FST::NODE()  #define BOOL 8, \  FST::NODE(1, FST::RELATION('l', 1)),\  FST::NODE(1, FST::RELATION('o', 2)),\  FST::NODE(1, FST::RELATION('g', 3)),\  FST::NODE(1, FST::RELATION('i', 4)),\  FST::NODE(1, FST::RELATION('c', 5)),\  FST::NODE(1, FST::RELATION('u', 6)),\  FST::NODE(1, FST::RELATION('s', 7)),\  FST::NODE()  #define DECLARE 11, \  FST::NODE(1, FST::RELATION('a', 1)),\  FST::NODE(1, FST::RELATION('n', 2)),\  FST::NODE(1, FST::RELATION('n', 3)),\  FST::NODE(1, FST::RELATION('u', 4)),\  FST::NODE(1, FST::RELATION('n', 5)),\  FST::NODE(1, FST::RELATION('t', 6)),\  FST::NODE(1, FST::RELATION('i', 7)),\  FST::NODE(1, FST::RELATION('a', 8)),\  FST::NODE(1, FST::RELATION('t', 9)),\  FST::NODE(1, FST::RELATION('e', 10)),\  FST::NODE()  #define INTEGER 6, \  FST::NODE(1, FST::RELATION('t', 1)),\  FST::NODE(1, FST::RELATION('o', 2)),\  FST::NODE(1, FST::RELATION('t', 3)),\  FST::NODE(1, FST::RELATION('u', 4)),\  FST::NODE(1, FST::RELATION('m', 5)),\  FST::NODE()  #define STRING 7, \  FST::NODE(1, FST::RELATION('c', 1)),\  FST::NODE(1, FST::RELATION('h', 2)),\  FST::NODE(1, FST::RELATION('o', 3)),\  FST::NODE(1, FST::RELATION('r', 4)),\  FST::NODE(1, FST::RELATION('d', 5)),\  FST::NODE(1, FST::RELATION('a', 6)),\  FST::NODE()  #define FUNCTION 6, \  FST::NODE(1, FST::RELATION('m', 1)),\  FST::NODE(1, FST::RELATION('u', 2)),\  FST::NODE(1, FST::RELATION('n', 3)),\  FST::NODE(1, FST::RELATION('u', 4)),\  FST::NODE(1, FST::RELATION('s', 5)),\  FST::NODE()  #define RETURN 8, \  FST::NODE(1, FST::RELATION('r', 1)),\  FST::NODE(1, FST::RELATION('e', 2)),\  FST::NODE(1, FST::RELATION('d', 3)),\  FST::NODE(1, FST::RELATION('i', 4)),\  FST::NODE(1, FST::RELATION('t', 5)),\  FST::NODE(1, FST::RELATION('u', 6)),\  FST::NODE(1, FST::RELATION('s', 7)),\  FST::NODE()  #define PRINT 7, \  FST::NODE(1, FST::RELATION('l', 1)),\  FST::NODE(1, FST::RELATION('o', 2)),\  FST::NODE(1, FST::RELATION('q', 3)),\  FST::NODE(1, FST::RELATION('u', 4)),\  FST::NODE(1, FST::RELATION('o', 5)),\  FST::NODE(1, FST::RELATION('r', 6)),\  FST::NODE()  #define MAIN 8, \  FST::NODE(1, FST::RELATION('p', 1)),\  FST::NODE(1, FST::RELATION('e', 2)),\  FST::NODE(1, FST::RELATION('l', 3)),\  FST::NODE(1, FST::RELATION('a', 4)),\  FST::NODE(1, FST::RELATION('g', 5)),\  FST::NODE(1, FST::RELATION('u', 6)),\  FST::NODE(1, FST::RELATION('s', 7)),\  FST::NODE()  #define ID 2, \  FST::NODE(62, \  FST::RELATION('a', 0), FST::RELATION('b', 0), FST::RELATION('c', 0), FST::RELATION('d', 0), FST::RELATION('e', 0), FST::RELATION('f', 0),\  FST::RELATION('g', 0), FST::RELATION('h', 0), FST::RELATION('i', 0), FST::RELATION('j', 0), FST::RELATION('k', 0), FST::RELATION('l', 0),\  FST::RELATION('m', 0), FST::RELATION('n', 0), FST::RELATION('o', 0), FST::RELATION('p', 0), FST::RELATION('q', 0), FST::RELATION('r', 0),\  FST::RELATION('s', 0), FST::RELATION('t', 0), FST::RELATION('u', 0), FST::RELATION('v', 0), FST::RELATION('w', 0), FST::RELATION('x', 0),\  FST::RELATION('y', 0), FST::RELATION('z', 0),\  \  FST::RELATION('1', 1), FST::RELATION('2', 1), FST::RELATION('3', 1), FST::RELATION('4', 1), FST::RELATION('5', 1), FST::RELATION('6', 1),\  FST::RELATION('7', 1), FST::RELATION('8', 1), FST::RELATION('9', 1), FST::RELATION('0', 1), FST::RELATION('1', 1), FST::RELATION('2', 1),\  FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1),\  FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1),\  FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1),\  FST::RELATION('s', 1), FST::RELATION('t', 1), FST::RELATION('u', 1), FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1),\  FST::RELATION('y', 1), FST::RELATION('z', 1)),\  FST::NODE()  #define X2 2, \  FST::NODE(4, \  FST::RELATION('0', 0), FST::RELATION('1', 0),\  \  FST::RELATION('0', 1), FST::RELATION('1', 1)),\  FST::NODE()  #define X8 3, \  FST::NODE(14, \  FST::RELATION('1', 0), FST::RELATION('2', 0), FST::RELATION('3', 0), FST::RELATION('4', 0), FST::RELATION('5', 0),\  FST::RELATION('6', 0), FST::RELATION('7', 0), \  \  FST::RELATION('1', 1), FST::RELATION('2', 1), FST::RELATION('3', 1), FST::RELATION('4', 1), FST::RELATION('5', 1),\  FST::RELATION('6', 1), FST::RELATION('7', 1)), \  FST::NODE(1, FST::RELATION('o', 2)),\  FST::NODE()  #define LITERAL\_STR 4, \  FST::NODE(2, FST::RELATION('\"', 1), FST::RELATION('\"', 2)),\  FST::NODE(158, \  FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1),\  FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1),\  FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1),\  FST::RELATION('s', 1), FST::RELATION('t', 1), FST::RELATION('u', 1), FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1),\  FST::RELATION('y', 1), FST::RELATION('z', 1), FST::RELATION('1', 1), FST::RELATION('2', 1), FST::RELATION('3', 1), FST::RELATION('4', 1),\  FST::RELATION('5', 1), FST::RELATION('6', 1), FST::RELATION('7', 1), FST::RELATION('8', 1), FST::RELATION('9', 1), FST::RELATION('0', 1),\  \  FST::RELATION('а', 1), FST::RELATION('б', 1), FST::RELATION('в', 1), FST::RELATION('г', 1), FST::RELATION('д', 1), FST::RELATION('е', 1),\  FST::RELATION('ё', 1), FST::RELATION('ж', 1), FST::RELATION('з', 1), FST::RELATION('и', 1), FST::RELATION('й', 1), FST::RELATION('к', 1),\  FST::RELATION('л', 1), FST::RELATION('м', 1), FST::RELATION('н', 1), FST::RELATION('о', 1), FST::RELATION('п', 1), FST::RELATION('р', 1),\  FST::RELATION('с', 1), FST::RELATION('т', 1), FST::RELATION('у', 1), FST::RELATION('ф', 1), FST::RELATION('х', 1), FST::RELATION('ц', 1),\  FST::RELATION('ч', 1), FST::RELATION('ш', 1), FST::RELATION('щ', 1), FST::RELATION('ъ', 1), FST::RELATION('ы', 1), FST::RELATION('ь', 1),\  FST::RELATION('+', 1), FST::RELATION('{', 1), FST::RELATION('}', 1), FST::RELATION(' ', 1), FST::RELATION('.', 1), FST::RELATION(',', 1),\  FST::RELATION('\'', 1), FST::RELATION('/', 1), FST::RELATION(';', 1), FST::RELATION('\*', 1), FST::RELATION('-', 1), FST::RELATION(')', 1),\  FST::RELATION('(', 1), \  \  FST::RELATION('a', 2), FST::RELATION('b', 2), FST::RELATION('c', 2), FST::RELATION('d', 2), FST::RELATION('e', 2), FST::RELATION('f', 2),\  FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('i', 2), FST::RELATION('j', 2), FST::RELATION('k', 2), FST::RELATION('l', 2),\  FST::RELATION('m', 2), FST::RELATION('n', 2), FST::RELATION('o', 2), FST::RELATION('p', 2), FST::RELATION('q', 2), FST::RELATION('r', 2),\  FST::RELATION('s', 2), FST::RELATION('t', 2), FST::RELATION('u', 2), FST::RELATION('v', 2), FST::RELATION('w', 2), FST::RELATION('x', 2),\  FST::RELATION('y', 2), FST::RELATION('z', 2), FST::RELATION('1', 2), FST::RELATION('2', 2), FST::RELATION('3', 2), FST::RELATION('4', 2),\  FST::RELATION('5', 2), FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2), FST::RELATION('9', 2), FST::RELATION('0', 2),\  \  FST::RELATION('а', 2), FST::RELATION('б', 2), FST::RELATION('в', 2), FST::RELATION('г', 2), FST::RELATION('д', 2), FST::RELATION('е', 2),\  FST::RELATION('ё', 2), FST::RELATION('ж', 2), FST::RELATION('з', 2), FST::RELATION('и', 2), FST::RELATION('й', 2), FST::RELATION('к', 2),\  FST::RELATION('л', 2), FST::RELATION('м', 2), FST::RELATION('н', 2), FST::RELATION('о', 2), FST::RELATION('п', 2), FST::RELATION('р', 2),\  FST::RELATION('с', 2), FST::RELATION('т', 2), FST::RELATION('у', 2), FST::RELATION('ф', 2), FST::RELATION('х', 2), FST::RELATION('ц', 2),\  FST::RELATION('ч', 2), FST::RELATION('ш', 2), FST::RELATION('щ', 2), FST::RELATION('ъ', 2), FST::RELATION('ы', 2), FST::RELATION('ь', 2),\  FST::RELATION('+', 2), FST::RELATION('{', 2), FST::RELATION('}', 2), FST::RELATION(' ', 2), FST::RELATION('.', 2), FST::RELATION(',', 2),\  FST::RELATION('\'', 2), FST::RELATION('/', 2), FST::RELATION(';', 2), FST::RELATION('\*', 2), FST::RELATION('-', 2), FST::RELATION(')', 2),\  FST::RELATION('(', 2)),\  \  FST::NODE(1, FST::RELATION('\"', 3)),\  FST::NODE()  #define PLUS 2, \  FST::NODE(1, FST::RELATION('+', 1)),\  FST::NODE()  #define MINUS 2, \  FST::NODE(1, FST::RELATION('-', 1)),\  FST::NODE()  #define STAR 2, \  FST::NODE(1, FST::RELATION('\*', 1)),\  FST::NODE()  #define DIRSLASH 2, \  FST::NODE(1, FST::RELATION('/', 1)),\  FST::NODE()  #define MODULE 2, \  FST::NODE(1, FST::RELATION('%', 1)),\  FST::NODE()  #define SEMICOLON 2, \  FST::NODE(1, FST::RELATION(';', 1)),\  FST::NODE()  #define COMMA 2, \  FST::NODE(1, FST::RELATION(',', 1)),\  FST::NODE()  #define LEFTBRACE 2, \  FST::NODE(1, FST::RELATION('{', 1)),\  FST::NODE()  #define BRACELET 2, \  FST::NODE(1, FST::RELATION('}', 1)),\  FST::NODE()  #define LEFTTHESIS 2, \  FST::NODE(1, FST::RELATION('(', 1)), \  FST::NODE()  #define RIGHTTHESIS 2, \  FST::NODE(1, FST::RELATION(')', 1)), \  FST::NODE()  #define RAVNO 2, \  FST::NODE(1, FST::RELATION('=', 1)),\  FST::NODE() |
|  |

ПРИЛОЖЕНИЕ Б

|  |
| --- |
| namespace MFST  {  struct MfstState //Состояние автомата (для сохранения)  {  short lenta\_position; // позиция на ленте  short nrule; // номер текущего правила  short nrulechain; // номер текущей цепочки, текущего правила  MFSTSTSTACK st; // стек автомата  MfstState();  MfstState(short pposition, MFSTSTSTACK pst, short pnrulechain);  //(позиция на ленте, стек автомата, номер текущей цепочки)  MfstState(short pposition, MFSTSTSTACK pst, short pnrule, short pnrulechain);  //(позиция на ленте, стек автомата, номер текущего правила, номер текущей цепочки)  };  struct Mfst //магазинный автомат  {  enum RC\_STEP // код возврата функции step  {  NS\_OK, //найдено правило и цепочка, цепочка записана в стек  NS\_NORULE, //не найдено правило грамматики(ошибка в грамматике)  NS\_NORULECHAIN, //не найдена подходящая цепочка правила (ошибка в исходном коде)  NS\_ERROR, //неизвестный нетерминальный символ грамматики  TS\_OK, //тек. символ ленты == вершине стека, продвинулась лента, рор стека  TS\_NOK, //тек. символ ленты != вершине стека, восстановлено состояние  LENTA\_END, //текущая позиция ленты >= lenta\_size  SURPRISE //неожиданный код возврата (ошибка в step)  };  struct MfstDiagnosis //диагностика  {  short lenta\_position; // позиция на ленте  RC\_STEP rc\_step; // код завершения шага  short nrule; // номер правила  short nrule\_chain; // номер цепочки правила  MfstDiagnosis();  MfstDiagnosis(short plenta\_position, RC\_STEP prc\_step, short pnrule, short pnrule\_chain);  //(позиция в ленте, код завершения шага, номер правила, номер цепочки правила)  }diagnosis[MFST\_DIAGN\_NUMBER]; // последние самые глубокие сообщения  GRBALPHABET\* lenta; //перекодированная (TS/NS) лента (из LEX)  short lenta\_position; // текущая позиция на ленте  short nrule; //номер текущего правила  short nrulechain; //номер текущей цепочки, текущего правила  short lenta\_size; //размер ленты  GRB::Graibach grebach; //граматика Грейбах  LT::LexTable lex; // результат работы лексического анализатора  IT::IdTable idtable;  MFSTSTSTACK st; //стек автомата  use\_container<std::stack <MfstState>> storestate; // стек для сохранения состояний  Mfst();  Mfst(LT::LexTable plex, IT::IdTable pid, GRB::Graibach pgrebach);  //(лекс табл, грамматика Грейбах  char\* getCSt(char\* buf); //получить содержимое стека  char\* getCLenta(char\* buf, short pos, short n = 25); //лента: n символов с pos  char\* getDiagnosis(short n, char\* buf); //получить n-ую строки или 0х00  bool savestate(); //сохранить состояние автомата  bool reststate(); //восстановить состояние автомата  bool push\_chain(GRB::Rule::Chain chain); //поместить цепочку правила в стек  RC\_STEP step(); //выполнить шаг автомата  bool start(Log::LOG log); //запустить автомат  bool savediagnosis(RC\_STEP pprc\_step); //код завершения шага  void printrules(Log::LOG log); //вывести последовательность правил  struct Deducation //вывод  {  short size; //количество шагов в выводе  short\* nrules; //номера правил грамматики  short\* nrulechains; //номера цепочек правил граматики (nrules)  Deducation() { size = 0; nrules = 0; nrulechains = 0; };  } deducation;  bool savededucation(); //сохранить дерево вывода  };  } |

|  |
| --- |
| namespace GRB  {  #define NS(n) Rule::Chain::N(n)  #define TS(n) Rule::Chain::T(n)  Graibach greibach(NS('S'), TS('$'), //стартовый символ, дно стека  7, //количество правил  Rule(NS('S'), GRB\_ERROR\_SERIES, //Неверная структура программы  6,  Rule::Chain(14, TS('n'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('L'), TS(';'), TS('}'), TS(';'), NS('S')),    Rule::Chain(14, TS('s'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('L'), TS(';'), TS('}'), TS(';'), NS('S')),    Rule::Chain(14, TS('b'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('L'), TS(';'), TS('}'), TS(';'), NS('S')),    Rule::Chain(14, TS('j'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('L'), TS(';'), TS('}'), TS(';'), NS('S')),    Rule::Chain(8, TS('m'), TS('{'), NS('N'), TS('r'), NS('L'), TS(';'), TS('}'), TS(';')),    Rule::Chain(9, TS('m'), TS('{'), NS('N'), TS('r'), NS('L'), TS(';'), TS('}'), TS(';'), NS('S'))    ),  Rule(NS('N'), GRB\_ERROR\_SERIES + 1, //Ошибочный оператор  20,  Rule::Chain(5, TS('d'), TS('n'), TS('i'), TS(';'), NS('N')),  Rule::Chain(4, TS('d'), TS('n'), TS('i'), TS(';')),    Rule::Chain(5, TS('d'), TS('s'), TS('i'), TS(';'), NS('N')),  Rule::Chain(4, TS('d'), TS('s'), TS('i'), TS(';')),  Rule::Chain(5, TS('d'), TS('j'), TS('i'), TS(';'), NS('N')),  Rule::Chain(4, TS('d'), TS('j'), TS('i'), TS(';')),  Rule::Chain(5, TS('d'), TS('b'), TS('i'), TS(';'), NS('N')),  Rule::Chain(4, TS('d'), TS('b'), TS('i'), TS(';')),  Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),    Rule::Chain(8, TS('u'), TS('('), NS('L'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';')),  Rule::Chain(9, TS('u'), TS('('), NS('L'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';'), NS('N')),  Rule::Chain(7, TS('u'), TS('('), NS('L'), TS(')'), TS('{'), TS('}'), TS(';')),  Rule::Chain(8, TS('u'), TS('('), NS('L'), TS(')'), TS('{'), TS('}'), TS(';'), NS('N')),  Rule::Chain(8, TS('z'), TS('('), TS('l'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';')),  Rule::Chain(9, TS('z'), TS('('), TS('l'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';'), NS('N')),  Rule::Chain(7, TS('z'), TS('('), TS('l'), TS(')'), TS('{'), TS('}'), TS(';')),  Rule::Chain(8, TS('z'), TS('('), TS('l'), TS(')'), TS('{'), TS('}'), TS(';'), NS('N')),  Rule::Chain(4, TS('p'), NS('L'), TS(';'), NS('N')),    Rule::Chain(3, TS('p'), NS('L'), TS(';'))    ),  Rule(NS('L'), GRB\_ERROR\_SERIES + 6,  2,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l'))  ),  Rule(NS('E'), GRB\_ERROR\_SERIES + 2, //ошибка в выражении  12,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(2, TS('i'), NS('M')),  Rule::Chain(2, TS('l'), NS('M')),  Rule::Chain(3, TS('('), NS('E'), TS(')')),  Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),  //библиотечные функции  Rule::Chain(3, TS('x'), TS('('), TS(')')),  Rule::Chain(4, TS('y'), TS('('), TS('i'), TS(')')),  Rule::Chain(4, TS('y'), TS('('), TS('l'), TS(')')),  Rule::Chain(4, TS('q'), TS('('), TS('i'), TS(')')),  Rule::Chain(4, TS('q'), TS('('), TS('l'), TS(')'))    ),  Rule(NS('M'), GRB\_ERROR\_SERIES + 3,  4,  Rule::Chain(2, TS('v'), NS('E')),  Rule::Chain(3, TS('v'), NS('E'), NS('M')),  Rule::Chain(2, TS('v'), NS('E')),  Rule::Chain(3, TS('v'), NS('E'), NS('M'))  ),  Rule(NS('F'), GRB\_ERROR\_SERIES + 4,  8,  Rule::Chain(2, TS('n'), TS('i')),  Rule::Chain(4, TS('n'), TS('i'), TS(','), NS('F')),  Rule::Chain(2, TS('s'), TS('i')),  Rule::Chain(4, TS('s'), TS('i'), TS(','), NS('F')),  Rule::Chain(2, TS('j'), TS('i')),  Rule::Chain(4, TS('j'), TS('i'), TS(','), NS('F')),  Rule::Chain(2, TS('b'), TS('i')),  Rule::Chain(4, TS('b'), TS('i'), TS(','), NS('F'))  ),  Rule(NS('W'), GRB\_ERROR\_SERIES + 5,  4,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('i'), TS(','), NS('W')),  Rule::Chain(3, TS('l'), TS(','), NS('W'))  )  ); |

|  |
| --- |
| Дерево разбора  0 : S->nfi(F){NrL;};S  4 : F->ni  8 : N->i=E;  10 : E->y(l)  16 : L->i  20 : S->m{NrL;};  22 : N->dsi;N  26 : N->dbi;N  30 : N->dni;N  34 : N->dni;N  38 : N->i=E;N  40 : E->l  42 : N->i=E;N  44 : E->i(W)  46 : W->i  49 : N->pL;N  50 : L->i  52 : N->i=E;N  54 : E->l  56 : N->i=E;N  58 : E->l  60 : N->u(L){N};N  62 : L->i  65 : N->i=E;  67 : E->q(l)  74 : N->pL;N  75 : L->l  77 : N->i=E;N  79 : E->(E)M  80 : E->iM  81 : M->vE  82 : E->l  84 : M->vE  85 : E->l  87 : N->pL;N  88 : L->i  90 : N->pL;N  91 : L->l  93 : N->i=E;N  95 : E->iM  96 : M->vE  97 : E->lM  98 : M->vE  99 : E->l  101 : N->pL;N  102 : L->i  104 : N->pL;N  105 : L->i  107 : N->i=E;N  109 : E->x()  113 : N->z(l){N};  118 : N->i=E;N  120 : E->iM  121 : M->vE  122 : E->l  124 : N->pL;  125 : L->i  130 : L->l |

ПРИЛОЖЕНИЕ В

|  |
| --- |
| int prior(char l)  {  if (l == '+' || l == '-')  return 1;  if (l == '\*' || l == '/' || l == '%')  return 2;  return 3;  }  bool Poland::PolishNotation(int i, LT::LexTable\* lextable, IT::IdTable& idTable)  {  std::stack<LT::Entry> stack;//операциии  std::queue<LT::Entry> queue;//ids & literals  short marker = i;  for (i; (\*lextable).table[i].operat != ';'; i++)  {  if ((\*lextable).table[i].operat >= 48 && (\*lextable).table[i].operat <= 122)  {  if ((\*lextable).table[i].idxTI != TI\_NULLIDX /\*|| (\*lextable).table[i].operat == 'o'\*/)//если это идентификатор  {  if (idTable.table[(\*lextable).table[i].idxTI].idtype == IT::IDTYPE::F)  {  do  {  if ((\*lextable).table[i].operat != '(' && (\*lextable).table[i].operat != ',')  {  queue.push((\*lextable).table[i]);  i++;  }  else  {  i++;  }  } while ((\*lextable).table[i].operat != ')');  // вместо ) ставлю id функции и заменяю лексему на @  (\*lextable).table[marker] = queue.front();  queue.pop();  marker++;  queue.front().lexema = '@';  queue.push(queue.front());  queue.pop();  i++;  //поставлю все до @ в таблицу лексем  for (marker; marker < marker + queue.size() - 1; marker++)  {  (\*lextable).table[marker] = queue.front();  queue.pop();  }  }  }  else if ((\*lextable).table[i].operat == 'x' || (\*lextable).table[i].operat == 'y' || (\*lextable).table[i].operat == 'q')  {  do  {  if ((\*lextable).table[i].operat != '(' && (\*lextable).table[i].operat != ',')  {  queue.push((\*lextable).table[i]);  i++;  }  else  {  i++;  }  } while ((\*lextable).table[i].operat != ')');  // вместо ) ставлю id функции и заменяю лексему на @  (\*lextable).table[marker] = queue.front();  queue.pop();  marker++;  queue.front().lexema = '@';  queue.push(queue.front());  queue.pop();  i++;  //поставлю все до @ в таблицу лексем  for (marker; marker < marker + queue.size() - 1; marker++)  {  (\*lextable).table[marker] = queue.front();  queue.pop();  }  }  }  if ((\*lextable).table[i].operat < 48)//( ) \* + - /  {  if (stack.size() > 0)  {  if (prior((\*lextable).table[i].operat) <= prior(stack.top().operat) && (\*lextable).table[i].operat != ')' && stack.top().operat != '(')  {  queue.push(stack.top());  stack.pop();  stack.push((\*lextable).table[i]);  }  else if (prior((\*lextable).table[i].operat) > prior(stack.top().operat) && (\*lextable).table[i].operat != ')' && (\*lextable).table[i].operat != '(')  {  stack.push((\*lextable).table[i]);  }  else if ((\*lextable).table[i].operat == ')')  {  while (stack.top().operat != '(')  {  queue.push(stack.top());  stack.pop();  }  stack.pop();//удалю (  }  else  {  stack.push((\*lextable).table[i]);  }  }  else  {  stack.push((\*lextable).table[i]);  }  }  if ((\*lextable).table[i].operat >= 48 && (\*lextable).table[i].operat <= 122)  {  queue.push((\*lextable).table[i]);  }  if ((\*lextable).table[i].operat == ';')  break;  }  while (stack.size() != 0)  {  queue.push(stack.top());  stack.pop();  }  //изменяю таблицу лексем  for (marker; marker < i; marker++)  {  if (!queue.empty())  {  (\*lextable).table[marker] = queue.front();  queue.pop();  }  else  {  (\*lextable).table[marker].idxTI = -1;  (\*lextable).table[marker].lexema = '#';  //(\*lextable).table[marker].sn = -1;  //(\*lextable).table[marker].lt\_vision = -3;  }  }  return true;  } |

ПРИЛОЖЕНИЕ Г

|  |
| --- |
| .586  .model flat, stdcall  includelib ../Debug/StaticLib.lib  randomness PROTO  transformatio PROTO: DWORD  quadrum PROTO: SDWORD  outInt PROTO: SDWORD  outChar PROTO: DWORD  outString PROTO: DWORD, :DWORD  outBool PROTO: SDWORD  ExitProcess PROTO : DWORD  .stack 4096  .const  \_L2 dword 'E', 0  \_L7 byte "collapse/chaos", 10, 0  \_L8 dword 1  \_L9 sdword -3  \_L10 sdword -2  \_L11 byte "one", 10, 0  \_L12 sdword 3  \_L13 sdword 2  \_L14 byte "two", 10, 0  \_L15 sdword 9  \_L16 sdword 5  \_L17 sdword 3  \_L18 sdword 10  \_L19 sdword 0  consoletitle db 'FSI-2020', 0  .data  a dword 0  z dword 0  one sdword 0  two sdword 0  counter sdword 0  .code  fa PROC s:sdword  push \_L2  call transformatio  push eax  pop s  mov eax, s  ret  mov eax,s  ret  fa ENDP  main PROC  push offset \_L7  pop a  push one  call fa  push eax  pop one  push one  call outInt  push \_L8  pop z  push \_L9  pop one  mov eax, z  mov ebx, 0  cmp eax, ebx  ja true\_if0  jmp continue\_if0  true\_if0:  push \_L10  call quadrum  push eax  pop one  continue\_if0:  push offset consoletitle  push offset \_L11  call outString  push one  push \_L12  pop eax  pop ebx  add eax, ebx  push eax  push \_L13  pop eax  pop ebx  imul ebx  push eax  pop one  push one  call outInt  push offset consoletitle  push offset \_L14  call outString  push one  push \_L15  push \_L16  pop ebx  pop eax  mov edx, 0  idiv ebx  push edx  pop ebx  pop eax  sub eax, ebx  push eax  pop two  push two  call outInt  push offset consoletitle  push a  call outString  call randomness  push eax  pop two  mov ecx, \_L17  push \_L17  pop counter  inc counter  \_for0:  push two  push \_L18  pop ebx  pop eax  mov edx, 0  idiv ebx  push eax  pop two  push two  call outInt  dec counter  mov ecx, counter  loop \_for0  push \_L19  call ExitProcess  main ENDP  end main |