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 SES-2023»

Выполнил студент Сафонов Евгений Сергеевич

(Ф.И.О.)

Руководитель проекта ст. преподаватель Наркевич Аделина Сергеевна

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

Заведующий кафедрой к.т.н., доц. Смелов В.В

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

Консультанты Фамилия Имя Отчество

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

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

Нормоконтролер Фамилия Имя Отчество

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

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[Введение 2](#_Toc153728939)

[1 Спецификация языка программирования 2](#_Toc153728940)

[1.1 Характеристика языка программирования 2](#_Toc153728941)

[1.2 Определение алфавита языка программирования 2](#_Toc153728942)

[1.3 Применяемые сепараторы 2](#_Toc153728943)

[1.4 Применяемые кодировки 2](#_Toc153728944)

[1.5 Типы данных 2](#_Toc153728945)

[1.6 Преобразование типов данных 2](#_Toc153728946)

[1.7 Идентификаторы 2](#_Toc153728947)

[1.8 Литералы 2](#_Toc153728948)

[1.9 Объявление данных 2](#_Toc153728949)

[1.10 Инициализация данных 2](#_Toc153728950)

[1.11 Инструкции языка 2](#_Toc153728951)

[1.12 Операции языка 2](#_Toc153728952)

[1.13 Выражения и их вычисления 2](#_Toc153728953)

[1.14 Конструкции языка 2](#_Toc153728954)

[1.15 Область видимости идентификаторов 2](#_Toc153728955)

[1.16 Семантические проверки 2](#_Toc153728956)

[1.17 Распределение оперативной памяти на этапе выполнения 2](#_Toc153728957)

[1.18 Стандартная библиотеки и её состав 2](#_Toc153728958)

[1.19 Ввод и вывод данных 2](#_Toc153728959)

[1.20 Точка входа 2](#_Toc153728960)

[1.21 Препроцессор 2](#_Toc153728961)

[1.22 Соглашения о вызовах 2](#_Toc153728962)

[1.23 Объектный код 2](#_Toc153728963)

[1.24 Классификация сообщений транслятора 2](#_Toc153728964)

[1.25 Контрольный пример 2](#_Toc153728965)

[2. Структура транслятора 2](#_Toc153728966)

[2.1 Компоненты транслятора их назначение и принципы взаимо-действия 2](#_Toc153728967)

[2.2 Перечень параметров транслятора 2](#_Toc153728968)

[2.3 Протоколы, формируемые транслятором 2](#_Toc153728969)

[3. Разработка лексического анализатора 2](#_Toc153728970)

[3.1 Структура лексического анализатора 2](#_Toc153728971)

[3.2 Входные и выходные данные 2](#_Toc153728972)

[3.3 Параметры лексического анализатора 2](#_Toc153728973)

[3.4 Алгоритм лексического анализатора 2](#_Toc153728974)

[3.5 Контроль входных символов 2](#_Toc153728975)

[3.6 Удаление избыточных символов 2](#_Toc153728976)

[3.7 Перечень ключевых слов 2](#_Toc153728977)

[3.8 Основные структуры данных 2](#_Toc153728978)

[3.9 Структура и перечень сообщений лексического анализатора 2](#_Toc153728979)

[3.10 Принцип обработки ошибок 2](#_Toc153728980)

[3.11 Контрольный пример 2](#_Toc153728981)

[4. Разработка синтаксического анализатора 2](#_Toc153728982)

[4.1 Структура синтаксического анализатора 2](#_Toc153728983)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 2](#_Toc153728984)

[4.3 Построение конечного магазинного автомата 2](#_Toc153728985)

[4.4 Основные структуры данных 2](#_Toc153728986)

[4.5 Описание алгоритма синтаксического разбора 2](#_Toc153728987)

[4.6 Параметры синтаксического анализатора 2](#_Toc153728988)

[4.7 Структура и перечень сообщений синтаксического анализатора 2](#_Toc153728989)

[4.8 Принцип при обработке ошибок 2](#_Toc153728990)

[4.9 Контрольный пример 2](#_Toc153728991)

[5. Разработка семантического анализатора 2](#_Toc153728992)

[5.1 Структура семантического анализатора 2](#_Toc153728993)

[5.2 Функции семантического анализатора 2](#_Toc153728994)

[5.3 Структура и перечень сообщений семантического анализатора 2](#_Toc153728995)

[5.4 Принцип обработки ошибок 2](#_Toc153728996)

[5.5 Контрольный пример 2](#_Toc153728997)

[6. Вычисление выражений 2](#_Toc153728998)

[6.1 Выражения, допускаемые языком 2](#_Toc153728999)

[6.2 Польская запись и принцип её построения 2](#_Toc153729000)

[6.3 Программная реализация обработки выражении 2](#_Toc153729001)

[6.4 Контрольный пример 2](#_Toc153729002)

[7 Генерация кода 2](#_Toc153729003)

[7.1 Структура генератора кода 2](#_Toc153729004)

[7.2 Представление типов данных в оперативной памяти 2](#_Toc153729005)

[7.3 Статическая библиотека 2](#_Toc153729006)

[7.4 Особенности генерации кода 2](#_Toc153729007)

[7.5 Параметры, управляющие генераций кода 2](#_Toc153729008)

[7.6 Контрольный пример 2](#_Toc153729009)

[8 Тестирование приложения 2](#_Toc153729010)

[8.1 Общие положения 2](#_Toc153729011)

[8.2 Результаты тестирования 2](#_Toc153729012)

[Заключение 2](#_Toc153729013)

[Список использованных источников 2](#_Toc153729014)

[Приложение A 2](#_Toc153729015)

[Приложение Б 2](#_Toc153729016)

[Приложение В 2](#_Toc153729017)

[Приложение Г 2](#_Toc153729018)

[Приложение Д 2](#_Toc153729019)

[Приложение Е 2](#_Toc153729020)

[Приложение Ж 8](#_Toc153729020)2

Введение

Данный курсовой проект представляет из себя создание собственного языка программирования SES-2023. Этот язык программирования представляет из себя простой учебный язык, который позволяет выполнять простейшие операции, арифметические действия над числами, создавать процедуры и функции и т.д.

Для создания собственного языка программирования требуется разработать собственный компилятор. В задачи компилятора входит: лексический анализ, семантический анализ и синтаксический анализ.

Исходя из раннее определённой цели курсового проекта, были определены следующие задачи:

* разработка спецификации языка программирования;
* разработка структуры транслятора;
* разработка лексического анализатора;
* разработка синтаксического анализатора;
* преобразование выражений;
* генерация кода на язык ассемблер;
* тестирование транслятора.

Далее пояснительная записка содержит решения выше поставленных задач.

****1 Спецификация языка программирования****

****1.1 Характеристика языка программирования****

Язык программирования SES-2023 – стековый, компилируемый, строго типизированный язык.

****1.2 Определение алфавита языка программирования****

**Язык программирования SES-2023 использует стандартную таблицу символов Windows-1251, в которой определены латинские и русские символы, символы-разделители, символы математических операций и специальные символы.**

****1.3 Применяемые сепараторы****

**В языке SES-2023 определён набор символов сепараторов для разделения лексем друг от друга. Все сепараторы приведены ниже в таблице 1.1.**

|  |  |
| --- | --- |
| **Сепаратор** | **Назначение сепаратором** |
| **{}** | **Программный блок** |
| **Пробел** | **Разделитель конструкций и лексем и инструкций** |
| **[]** | **Индекс массива или строки** |

Таблица 1.1 - Применяемые сепараторы

****1.4 Применяемые кодировки****

**Для написания исходного кода программы на языке SES-2023 используются латинские символы и символы разделители кодировки Windows-1251. Русские символы допускается использовать в строковых литералах.**

****1.5 Типы данных****

**В языке программирования предусмотрены следующие типы данных: целочисленный int (4 байта), целочисленный byte (1 байт), string строковый.**

Таблица 1.2 - Фундаментальные типы данных

|  |  |
| --- | --- |
| **Тип** | Описание |
| **int** | Фундаментальный тип данных. Автоматическая инициализация 0;  Возможные операции:   * арифметические(+, -, \*,/); * операции сравнения(<, >,>=,<=,==)   Возможные значения : -2^31 до 2^31 включительно. |

Окончание таблицы 1.2

|  |  |
| --- | --- |
| **Тип** | Описание |
| **byte** | Фундаментальный тип данных. Автоматическая инициализация 0;  Возможные операции:   * арифметические(+, -,\*,/); * операции сравнения(<, >,>=,<=,==);   Возможные значения : -2^7 до 2^7 включительно. |
| string | Фундаментальный тип данных. Предусмотрен для объявления строк. (1 символ – 1 байт). Автоматическая инициализация строкой нулевой длины. Применяется кодировка Windows-1251. |
| bool | Фундаментальный тип данных. Занимает 1 байт, значение 0 или false или любое отличное от 0 или true, автоматическая инициализация 0 . |
| array int | Структура данных представляющая собой последовательность элементов типа int длиной n. При объявлении количество элементов n должно находится в стеке. Начиная с верхнего элемента стека после извлеченного n значения присваиваются элементам массива по порядку. Если в стеке недостаточно элементов автоматически каждый элемент массива инициализируется 0. |
| array byte | Структура данных представляющая собой последовательность элементов типа int длиной n. При объявлении количество элементов n должно находится в стеке. Начиная с верхнего элемента стека после извлеченного n значения присваиваются элементам массива по порядку. Если в стеке недостаточно элементов автоматически каждый элемент массива инициализируется 0. |

****1.6 Преобразование типов данных****

**В языке SES-2023 доступно преобразование из byte в int, но не наоборот.** Все остальные типы данных определенны однозначно и не могут быть преобразованы в другие.

****1.7**** Идентификаторы

В языке SES-2023 идентификаторы являются именами для различных переменных, функций и параметров для этих функций. Не предусмотрено дублирование идентификаторов в одной области видимости. Так же имя идентификатора не может совпадать с каким-либо ключевым словом языка. Может состоять только латинских букв, цифр и знака нижнего подчёркивания. Первой в имени идентификатора не может быть цифра.

Правило записи имени идентификатора можно задать регулярным выражением: [a-zA-Z\_][a-zA-Z0-9\_]\*

Примеры правильного идентификаторов: idenf, idenf\_123

****1****.8 Литералы

Литерал в языке программирования — это константное значение, напрямую вписанное в исходный код программы.

Литералы могут быть представлены в исходном коде программы различными способами. Например:

* целочисленные литералы: 42,042;
* строковые литералы: ‘Hello, world!’, 'a','123';
* булевые литералы: true, false;

Численные литералы записываются как набор цифр для десятичного представления. Для записи восьмеричного целочисленного литерала необходимо указать префикс 0 в начале литерала.

****1.9 Объявление данных****

**Для использования переменной она должна быть объявлена до этого момента. Переменная может быть объявлена в любом программном блоке, если она уже не была объявлена в данной области видимости. Переменная не может иметь глобальную область видимости. Если будет объявлена вне любого программного блока, то транслятором это будет распознано как синтаксическая ошибка.**

**Правила объявления переменной:**

<тип данных> <идентификатор>

1.10 Инициализация данных

При объявлении различных типов данных им сразу присваивается значение по умолчанию. Например, целочисленные по умолчанию имеют значение 0, булевые имеют false, строковые полностью заполняются -1. Если попытаться вывести на консоль незаполненный элемент строки, то будет выведено: “я”.

Каждой переменной должно быть явно присвоено какое-либо значение пользователем. В случае массивов или строк, то достаточно присвоить одному из элементов какое-либо значение. В противном случае транслятор расценит это как ошибку, так как переменная была объявлена, но не используется, потому что пользователь сам туда ничего не положил.

1.11 Инструкции языка

Все инструкции, существующие в языке SES-2023 представлены на таблице 1.3 ниже. Стоит отметить, что при применении инструкций языка нужно убедится, что на момент вызова в стеке находится необходимое количество значений для выполнения данной инструкции. Если в стеке недостаточно элементов, то об этом возможно узнать только во время самого исполнения программы. После выполнения инструкции те значения, которые использовались для вычислений будут извлечены и на их место будет помещен результат выполнения инструкции. Данные правила так же распространяются для вызова функций.

Таблица 1.3 — Инструкции языка

|  |  |
| --- | --- |
| Инструкция языка | Синтаксис |
| Вызов функций | <идентификатор  функции> |
| Объявление переменной | <тип данных> <идентификатор> |
| Присваивание | <идентификатор> $ |
| Вывод данных | write  writeline |
| Объявление внешней функции | function <тип\_данных> <идентификатор> (<тип данных> <идентификатор>, …) {<тело\_функции>}  <тело\_функции>::=<инструкция><набор\_инструкций>|<инструкция> |
| Дублирование верхнего элемента стека | dup |
| Удаление верхнего элемента стека | drop |
| Обмен местами двух верхних элементов стека | swap |
| Дублирование предпоследнего элемента стека на его вершину | over |

Окончание таблицы 1.3

|  |  |
| --- | --- |
| Инструкция языка | Синтаксис |
| Получение глубины стека в верх стека | deapth |
| Очистка всего стека | clear |
| Возврат из функции | return |

****1.12 Операции языка****

**Язык SES-2023 может выполнять различные арифметические операции и операции сравнения. Все операции представлены в таблице 1.4.**

**Таблица 1.4— Операции языка**

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Арифметическое назначение | Приоритетность операции | Свойства |
| () | Приоритетность операций | 0 | Ассоциативность, коммутативность, дистрибутивность |
| + | Сложение | 1 | Коммутативность, ассоциативность |
| - | Вычитание | 1 | Антикоммутативность,  ассоциативность |
| \* | Умножение | 2 | Коммутативность,  ассоциативность |
| / | Деление | 2 | Антикоммутативность, ассоциативность |
| >,>= | Сравнение чисел | 3 | Антикоммутативность |
| <,<= | Сравнение чисел | 3 | Антикоммутативность |
| = | Сравнение чисел | 3 | Коммутативность |

**Порядок выполнения операций SES-2023 одинаковым приоритетом следует левоассоциативности.**

****1.13 Выражения и их вычисления****

Предусмотрены следующие правила составления выражений:

* выражение может содержать вызов функций;
* рассматриваются слева направо;

1.14 Конструкции языка

Языка SES-2023 имеет всего 4 конструкции: главная функция, представляет собой набор инструкций и операторов, функция, тело которой определяет сам пользователь, условный и цикл. Стоит отметить, что функции, определенные пользователем некоторые черты функциональных языков программирования: функции не имеют побочных эффектов, а возвращают в стек только единственный результат вычисленный на основе параметров. Функция имеет свой собственный стек, и она не может класть или извлекать элементы из стека другого блока кода, из которой она была вызвана. Нельзя определить функцию, которая не имела бы параметров. В конце тела функции обязательно должен стоять оператор возврата, несмотря на остальные в том же теле.

Ключевые программные конструкции языка программирования SES-2023 представлены в таблице 1.5.

**Таблица 1.5 — Конструкции языка**

|  |  |
| --- | --- |
| Конструкция | Представление в языке |
| Главная функция | main  {  …  }; |
| Функция | <тип данных> function <идентификатор> (<тип\_данных> <идентификатор>, …)  {  …  <идентификатор\литерал> return  } |
| Условный оператор | if (<условие>) {<инструкции языка>}else {<инструкции языка>} |
| Цикл | while(<условие>){<инструкции языка>} |

****1.15 Область видимости идентификаторов****

**Все идентификаторы, объявленные внутри блока, являются локальными. Идентификаторы переменные внутри какой-либо функции, доступны только лишь в ней. Это же распространяется и для параметров.** Объявление пользовательских областей видимости не предусмотрено.

****1.16 Семантические проверки****

Перечень семантических проверок, предусмотренных языком, приведен в таблице 1.6.

Таблица 1.6 - Перечень семантических проверок

|  |  |
| --- | --- |
| **№** | **Правило** |
| **1** | **Наличие main** |
| **2** | **Идентификаторы усекаются до 5 символов** |
| **3** | **Не допускается совпадения названий идентификаторов с ключевыми словами** |
| **4** | **Все функции должны быть объявлены перед main** |
| **5** | **Все идентификаторы должны быть обвялены в той же области видимости перед использованием** |
| **6** | **Соответствие типов формальных и фактических параметров при вызове**  **функций** |
| **7** | **Соответствие типов формальных и фактических параметров при вызове**  **операторов** |
| **8** | **Все переменные должны инициализироваться** |
| **9** | **Инструкции if и while должны содержать логический оператор** |
| **10** | **При обращении к массиву с помощью оператора индексации, передаваемый индекс не должен выходить за пределы массива** |
| **11** | **Возвращаемое значение функции должно совпадать с типом функции** |
| **12** | **При присваивании значения переменной, переменная должна быть явно указана перед оператором присваивания** |
| **13** | **При объявлении массива или строки требуется явно указать количество элементов. (При объявлении строки без количества символов, то это переменная сможет содержать только один символ)** |
| **14** | **Проверка на корректность скобок при выделении блоков кода(функции, блоки условных операторов и циклов)** |

1.17 Распределение оперативной памяти на этапе выполнения

Транслированный код использует два вида области памяти: куча и стек. В куче размещаются переменные и литералы. Стек используется для размещения туда переменных и литералов для последующего использования из операторами и функциями.

1.18 Стандартная библиотеки и её состав

В языке SES-2023 присутствуют стандартные библиотеки, которые необходимо вручную подключать с помощью директивы препроцессора #include. Содержимое библиотек и описание функций представлено в таблице 1.7.

Таблица 1.7 - Стандартная библиотека языка SES-2023

|  |  |
| --- | --- |
| Функция(Библиотека) | Описание |
| factor(math.lib) | Математическая функция. Принимает в качестве параметра целое число из стека. Возвращает произведение всех чисел от 1 до заданного числа. |
| power(math.lib) | Математическая функция. Принимает в качестве параметров два целочисленных значения из стека. Возвращает целочисленное число, значение, которого является первый параметр base возведённый в степень exponent, который является вторым параметром. |
| length(string.lib) | Строковая функция. Снимает строку со стека и помещает её длину (количество символов) обратно в стек |
| compare(string.lib) | Строковая функция. Снимает две строки со стека и помещает результат сравнения (например, false для разны и true для одинаковых) обратно в стек. |
| print(string.lib) | Строковая функция, предназначенная для вывода полной строки с переводом каретки. Может принимать строку для полного вывода либо n-ый элемент для вывода начиная с него и до конца строки. |

****1.19 Ввод и вывод данных****

Стандартный вывод данных осуществляется с помощью ключевого слова “write” или “writeline” для вывода строка с переводом каретки. Есть возможность выводить значение переменных и строковых литералов, которые будут заключены в двойные кавычки. При использовании стандартного вывода для строк или массива возможно вывести только отдельный элемент структуры данных. Важно отметить, что выводится то, что находится в вершине стека, и что после вывода значение будет удалено из стека.

Для ввода данных необходимо использовать ключевое слово read перед, которым нужно явно указать переменную, которую будет осуществлен ввод. В случае ввода некорректного значения, например если пользователь попытается поместить в переменную типа byte число, выходящее за пределы допустимых значений, то будет выведено соответствующее сообщение на экран, но программа продолжит своё исполнение, а в переменную будет помещён 0 или false, в зависимости от типа.

****1.20 Точка входа****

Точкой входа в программе является ключевое слово “main”. Точка входа не может отсутствовать или быть переопределена.

1.21 Препроцессор

Препроцессор в языке программирования SES-2023 имеет директиву #include, целью которой является подключения встроенных библиотек языка программирования SES-2023. После самой директивы необходимо написать обязательно через один пробел название библиотеки.

Пример: #include math.lib

Ещё препроцессор имеет директиву #polish для преобразования выражения, записанного в инфиксной форме. Подробное описание данной директивы находится в главе 6.

После обработки препроцессором исходный код и набор подключенных библиотек передаётся лексическому анализатору.

Обработанный код препроцессором дублируется в текстовый файл под названием AfterPreproc.txt.

1.22 Соглашения о вызовах

Все параметры заноситься в стек вручную как для функций, так и для выполнения инструкций. В функцию параметры передаются, начиная с вершины стека, то есть верхняя вершина стека будет первым параметром и так далее.

1.23 Объектный код

Язык SES-2023 транслируется в исходный код для языка Assembler.

1.24 Классификация сообщений транслятора

Обрабатываются ошибки на всех этапах обработки исходного кода, то есть во время прохождения различных этапов анализа.

Классификация ошибок представлена в таблице 1.9.

Таблица 1.8 - Классификация ошибок

|  |  |
| --- | --- |
| Номера ошибок | Характеристика |
| 0 – 1 | Системные ошибки |
| 2-3 | Ошибка препроцессора |
| 100 – 112 | Ошибки входных параметров |
| 113-119 | Ошибки лексического анализа |
| 120-129 | Ошибки семантического анализа |
| 600-603 | Ошибки синтаксического анализа |

1.25 Контрольный пример

Для того показать наглядно возможности языка SES-2023 составлен контрольный пример, который находится в приложении A. Чтобы показать пригодность языка SES-2023 для решения прикладных задач было создано решения для популярной задачи «Баланс скобок», состоящая в проверке правильности расставления круглых скобок: если скобка открылась, то она обязательно должна закрываться. Решение данной задачи представлено в Приложении Ж.

**2. Структура транслятора**

****2.1 Компоненты транслятора их назначение и принципы взаимо-действия****

**В языке SES-2023 исходный код транслируется в код языка Assembler. Но перед этим транслятор должен пройти несколько фаз транслятора, после которых будут составлены таблицы идентификаторов и лексем для последующей генерации кода.**

**Таблицы создаются на фазе лексического анализа. Одной из задач данной фазы трансляции является максимальное упрощение кода путём замены его на набор лексем и удалением лишних символов. В данной реализации лексический анализатор будет выполнять частично роль семантического анализатор, проверяя в некоторых местах корректность написанного кода.**

Синтаксический анализатор – фаза транслятора, выполняющая проверку исходного кода на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Результатом работы является дерево разбора

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть проверку исходного кода на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Входными данными являются таблица лексем и идентификаторов.

**Генератор кода** – фаза транслятора, преобразующая промежуточное представление кода в код на языке Assembler. На вход генератора подаются таблица лексем и таблица идентификаторов. В результате получается файл типа .asm в котором находится представление исходного кода.

Полная схема транслятора представлена на рисунке 2.1.

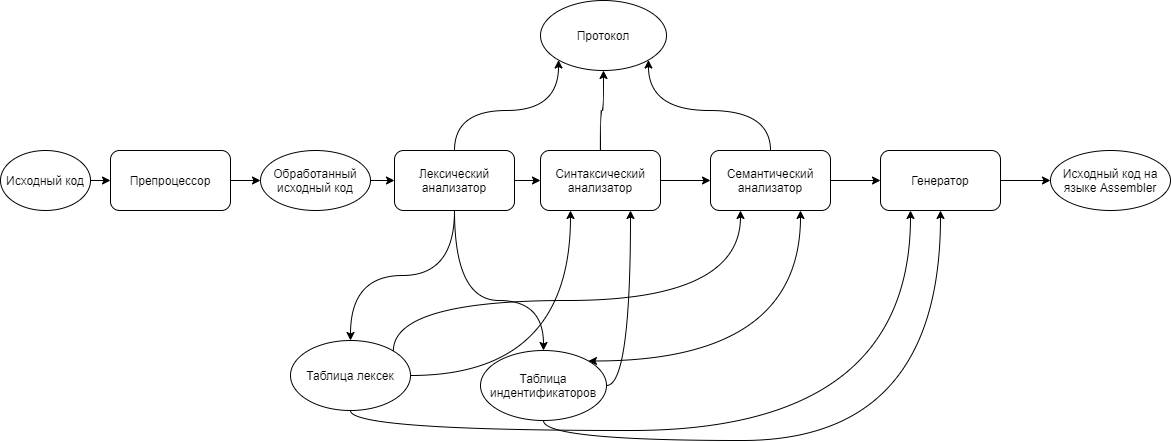


Рисунок 2.1­— Схема транслятора

****2.2 Перечень параметров транслятора****

Для формирования файлов с результатами работы лексического, синтаксического и семантического анализаторов используются входные параметры транслятора, которые приведены в таблице 2.2.

Таблица 2.1 - Входные параметры транслятора языка SES-2023

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке SES-2023 , имеющий расширение .txt | Не предусмотрено |
| -log:<путь к log-файлу> | Файл журнала для вывода протоколов работы программы. | Значение по умолчанию:  <имя in-файла>.log |
| -out:<путь к out-файлу> | Выходной файл – результат работы транслятора. Содержит исходный код на языке ассемблера. | Значение по умолчанию:  <имя in-файла>.asm |

****2.3 Протоколы, формируемые транслятором****

В ходе работы программы формируются протоколы работы лексического, синтаксического и семантического анализаторов, которые содержат в себе перечень протоколов работы. Все файлы создаются в корневом каталоге. В таблице 2.2 приведены протоколы, формируемые транслятором и их содержимое.

Таблица 2.2 - Протоколы, формируемые транслятором языка SES-2023

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования SES-2023. Содержит информацию про входные параметры, общем количестве символов и строк(исходные данные), протокол работы синтаксического анализатора, полученный на этапе синтаксического анализа. В случае ошибки текст, строка и позиция ошибки будет записан в файл. |
| Выходной файл с названием "IT.txt" | Файл содержит таблицу идентификаторов, сформированную во время лексического анализа. |
| Выходной файл с названием "LT.txt" | Файл содержит таблицу лексем, сформированную во время лексического анализа. |

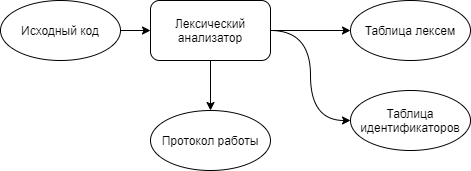
Продолжение таблицы 2.2

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Выходной файл, заданный параметром "-out:" c расширением .asm | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

****3. Разработка лексического анализатора****

****3.1 Структура лексического анализатора****

**Лексический анализатор – это фаза транслятора выполняющая лексический анализ исходного кода. Схема работы лексического анализатора представлена на рисунке 3.1.**

****

**Рисунок 3.1 – Схема лексического анализатора**

**Лексический анализатор принимает на вход исходный код, обработанный препроцессором. На выходе формируется таблица идентификаторов и лексем.**

****3.2 Входные и выходные данные****

**Входными данными является исходный код, который предварительно обработан препроцессором: код разделён на лексемы, удалены лишние символы и обработаны директивы.**

**Выходными данными являются таблица лексем, в которой содержится ещё более упрощенный код для последующего его понимания другими модулями транслятора, и таблица идентификаторов в которой содержаться все имена переменных, их тип, начальное значение, а также литералы. Обе таблицы связаны друг с другом ссылкой лексемы идентификатора или литерала на советующую запись в таблице идентификаторов.**

**Две таблицы применяются потом для последующих проверок семантики и синтаксиса языка, а также для генерации кода на языке Assembler.**

****3.3 Параметры лексического анализатора****

**Лексический анализатор не имеет параметров. Он все время работает в одинаковом и одном режиме.**

****3.4 Алгоритм лексического анализатора****

**Алгоритм работы лексического анализатора:**

1. **Исходный код читается посимвольно. Каждый символ записывается в строковый буфер, пока не будет достигнут символ-сепаратор**;
2. **Строка передаётся различным конечным автоматам. Если какой­–либо из автоматов разберет строку успешно, то он вернёт лексическому анализатору одну символьную лексему, которая будет записана в таблицу идентификаторов. Если лексема является идентификатором или литералом, то далее идут пункт 3. В противном случае алгоритм начинается сначала**;
3. **В зависимости от прошлых лексем, которые означают тип идентификатора. Идентификатору присваивается тип данных**;
4. **Если идентификатор только объявляется с явным указанием типа, то в частично заполненной таблице идентификаторов происходит поиск идентификатора с таким именем, предварительно будет записан тип и область видимости из стека. Если поиск успешен, то лексический анализ останавливается и выводится ошибка, в противном случае идентификатор добавляется в таблицу идентификаторов, а также в таблицу лексем добавляется лексема идентификатора с номер индекса таблицы идентификаторов, где он записан. Переходим в пункт 1**;
5. **Если уже объявленный идентификатор просто используется по ходу программы, то происходит его поиск в частично заполненной таблице. Если идентификатор не найден, то будет выведена соответствующая ошибка и лексический анализ будет завершён. В противном случае лексеме присваивается соответствующий индекс таблицы, и она записывается в таблицу лексем. Переходим в пункт 1**;
6. **Если лексема является литералом, то сначала будет выявлен тип литерала и значение. Если такой же литерал записан уже в таблицу идентификаторов, запись в неё производится не будет, в противном случае наоборот. В таблицу лексем записывается лексема с ссылкой на таблицу идентификаторов. Переходим в пункт 1;**
7. **Если идентификатор является функцией, то она будет записан в таблицу идентификаторов с соответствующими типом возвращаемого значений. Последующие идентификаторы в круглых скобках будут записаны как параметры. В стек помещается функция для отметки о области видимости последующих идентификаторов. Функция там останется пока не буде закончено объявление этой же функции. Переходим в пункт 1**
8. **Если не пройден весь исходный код, то переходим в пункт 1.**

****3.5 Контроль входных символов****

Таблица входных символов представлена на рисунке 3.2.

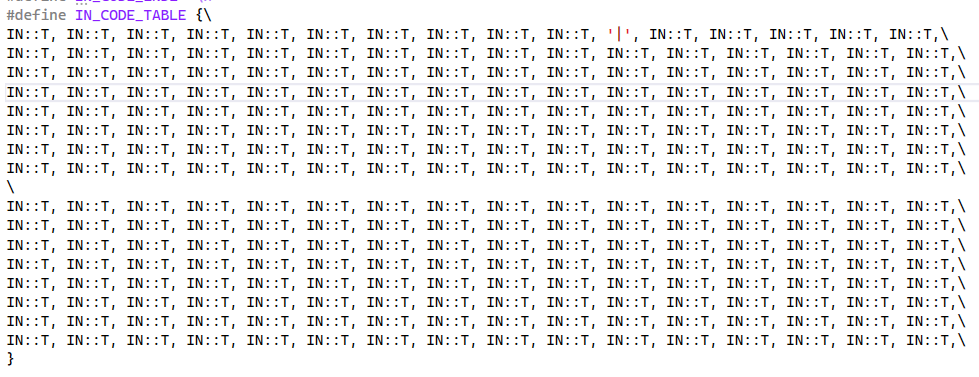


Рисунок 3.2 ­– Таблица контроля входных символов

**Индекс элемента соответствует его коду в таблице символов Window-1251.**

**T­ – символ разрешён. F – символ запрещён. I – символ игнорируется. Если на место символа записан другой, то будет происходить замена на записанный.**

****3.6 Удаление избыточных символов****

**Избыточным символами являются символы пробела и символ перевода каретки. Символы табуляции заменены препроцессором на пробелы. Пробел также является символом–сепаратором.**

**Важно помнить, что при записи отрицательных литералов или операций, состоящих из нескольких символов пробел недопустим, иначе выражение будет разобрано не так, как подразумевается пользователем. Например, следующая строчка кода «- 1» , будет распознана транслятором, как применение операции «-» и помещение литерала в стек. Если бы пробел отсутствовал, то данная строчка кода бы значила помещение отрицательного целочисленного литерала в стек.**

**При работе лексического анализатора он читает исходный код посимвольно и при встрече сепаратора прекращает запись в буфер и начинает проверку записанной текущей лексемы. При встрече символа перевода каретки увеличивается счётчик строк на единицу.**

****3.7 Перечень ключевых слов****

**Полный перечень ключевых слов приведён в таблице 3.1.**

**Таблица 3.1 ­­– Перечень ключевых слов**

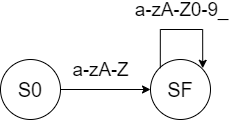
|  |  |  |
| --- | --- | --- |
| **Тип цепочки** | **Цепочка** | **Лексема** |
| **Ключевые слова** | **int** | **t** |
| **byte** | **t** |
| **bool** | **t** |
| **string** | **t** |
| **array** | **a** |
| **function** | **f** |
| **main** | **m** |
| **while** | **h** |

Окончание таблицы 3.1

|  |  |  |
| --- | --- | --- |
| **Тип цепочки** | **Цепочка** | **Лексема** |
| **Ключевые слова** | **if** | **z** |
| **true** | **l** |
| **false** | **l** |
| **Операторы** | **+** | **+** |
| **-** | **-** |
| **\*** | **\*** |
| **/** | **/** |
| **dup** | **d** |
| **drop** | **r** |
| **swap** | **s** |
| **over** | **o** |
| **depth** | **e** |
| **clearк** | **c** |
| **return** | **q** |
| **write** | **w** |
| **writeline** | **x** |
| **read** | **n** |
| **$** | **$** |
| **>** | **>** |
| **<** | **<** |
| **=** | **=** |
| **>=** | **p** |
| **<=** | **k** |
| **!=** | **j** |
| **Сепараторы** | **[]** | **@** |
| **{** | **{** |
| **}** | **}** |

**Для распознавания вышеперечисленных цепочек используется механизм конечного автомата, в которых цепочки записаны в виде регулярных выражений. Пример записи регулярного выражения для идентификатора:** **[a-zA-Z\_][a-zA-Z0-9\_]\*.**

**Два графа переходов для конечных автоматов для цепочки идентификатора и целочисленного типа представлены на рисунке 3.3, 3.4.**

****

**Рисунок 3.3 – Граф для идентификатора**

**C:\Users\HP\Desktop\int.png**

**Рисунок 3.4 – Граф для целочисленного типа int**

**Код, который реализует все цепочки разбора представлены в приложении Б.**

****3.8 Основные структуры данных****

Основные структуры таблиц лексем и идентификаторов данных языка SES- 2023, используемых для хранения, представлены в приложении Б. В таблице лексем содержится лексема, которая имеет номер строки в исходном коде, индекс таблицы идентификатора, если данная лексема литерал или идентификатор, то имеется номер соответствующего элемента в таблице идентификаторов. В таблице идентификаторов содержится имя идентификатора, тип данных, смысловой тип идентификатора, область видимости и его значение.

3.9 Структура и перечень сообщений лексического анализатора

После обнаружения ошибки лексический анализатор немедленно прекращает работу, выводит ошибку на консоль и записывает её в протокол.

Полный перечень возможных ошибок при фазе лексического анализа перечислен в таблице 3.2.

Таблица 3.2– Перечень ошибок при лексическом анализе

|  |  |
| --- | --- |
| Код | Сообщение |
| 113 | Недопустимый идентификатор |
| 114 | Дублирование идентификатора |
| 115 | Идентификатор не определён |
| 116 | Некорректное использование индекса |
| 117 | Недопустимый размер таблицы идентификаторов |
| 118 | Превышен размер таблицы идентификаторов |
| 119 | Превышена длина лексемы |

3.10 Принцип обработки ошибок

На консоль выводится изначально номер ошибки и само сообщение. Далее выводится строка и позиция, в которой допущена ошибка.

Так же ошибка записывается в протокол в таком же формате.

****3.11 Контрольный пример****

**Результат работы лексического анализатора, а именно таблица лексем и идентификаторов, представлены в приложении В.**

****4. Разработка синтаксического анализатора****

****4.1 Структура синтаксического анализатора****

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания заранее определённых синтаксических конструкций. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора

Входными данными для синтаксического анализатора являются таблица лексем и идентификаторов. Выходными данными являются дерево разбора и ошибки, которые выводятся на консоль и в протокол работы транслятора.

4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка SES-2023 используется контекстно-свободная грамматика G=<T, N, P, S>, где

T – множество терминальных символов(набор лексем, перечисленных в таблице 3.3),

N – множество нетерминальных символов (первый столбец таблицы 4.2),

P – множество правил языка (второй столбец таблицы 4.2),

S – начальный символ грамматики, являющийся нетерминалом.

Эта грамматика имеет нормальную форму Грейбах, т.к. она не леворекурсивная (не содержит леворекурсивных правил).

Полный перечень синтаксических правил в форме Грэйбах перечислены в таблице 4.1.

Таблица 4.1 – Синтаксические правила

|  |  |  |
| --- | --- | --- |
| **Нетерминал** | **Цепочка** | **Описание** |
| **S** | **m{N}** | **Стартовый символ порождает всю структуру исходного кода.** |
| **fti(W){Niq}S** |
| **fti(W){iq}S** |
| **fti(W){Nlq}S** |
| **fti(W){lq}S** |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| **Нетерминал** | **Цепочка** | **Описание** |
| **S** | **fati(W){Nir}S** |  |
| **N** | **i** | **Данный символ порождает правила, описывающие корректную запись операторов языка SES-2023, которые составляют почти весь исходный код.** |
| **iN** |
| **l** |
| **lN** |
| **ti** |
| **tiN** |
| **atil@** |
| **atil@N** |
| **til@** |
| **til@N** |
| **z(N){N}** |
| **z(N){N}N** |
| **h(N){N}** |
| **h(N){N}N** |
| **i$** |
| **i$N** |
| **ii@$** |
| **ii@$N** |
| **il@$** |
| **il@$N** |
| **w** |
| **wN** |
| **x** |
| **xN** |
| **d** |
| **dN** |
| **r** |
| **rN** |
| **s** |
| **sN** |
| **o** |
| **oN** |
| **in** |
| **inN** |
| **e** |
| **eN** |
| **c** |
| **cN** |
| **+** |
| **+N** |

Окончание таблицы 4.1

|  |  |  |
| --- | --- | --- |
| **Нетерминал** | **Цепочка** | **Описание** |
| **N** | **-** |  |
| **S** | **-N** |  |
| **\*** |  |
| **\*N** |  |
| **/** |
| **/N** |  |
| **=** |
| **=N** |
| **>** |
| **>N** |
| **<** |
| **<N** |
| **p** |
| **pN** |
| **k** |
| **kN** |
| **j** |
| **jN** |
| **ii@** |
| **ii@N** |
| **iL@** |
| **iL@N** |
| **W** | **ti** | **Данный символ порождает, правила для описания набора параметров при объявлении функции, которые она будет принимать при вызове.** |
| **ti,W** |

****4.3 Построение конечного магазинного автомата****

Конечный автомат с магазинной памятью представляет собой семерку. Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

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. |

Окончание таблицы 4.2

|  |  |  |
| --- | --- | --- |
|  | Алфавит входных символов | Алфавит представляет из себя множества терминальных и нетерминальных символов, описание которых содержится в таблице 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $). |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики. |
|  | 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. |

****4.4 Основные структуры данных****

Структуры Rule и Greibach описывают элементы грамматики Грэйбах. Вот их описание:

Структура Rule (правило в грамматике Грэйбах):

* nn: Нетерминалы (левый символ правила) < 0 (GRBALPHABET).
* iderror: Идентификатор диагностического сообщения (int).
* size: Количество цепочек - правых частей правила (short).
* chains: Массив цепочек - правых частей правила (struct Chain\*).

Структура Chain (цепочка, правая часть правила):

* size: Длина цепочки (short).
* nt: Цепочка символов, которые могут быть терминалами (>0) или нетерминалами (<0) (GRBALPHABET\*).

Структура Rule содержит также некоторые вспомогательные функции:

* getCChain(char\* b): Получает правую сторону правила в виде строки (char\*).
* T(char t): Статическая функция, возвращающая символ-терминал (GRBALPHABET) на основе символа (char).
* N(char n): Статическая функция, возвращающая символ-нетерминал (GRBALPHABET) на основе символа (char).
* isT(GRBALPHABET s): Статическая функция, определяющая, является ли символ терминалом (bool).
* isN(GRBALPHABET s): Статическая функция, определяющая, является ли символ нетерминалом (bool).
* alphabet\_to\_char(GRBALPHABET s): Статическая функция, преобразующая GRBALPHABET в символ (char).

Структура Greibach (грамматика Грэйбах):

* size: Количество правил (short).
* startN: Стартовый символ (GRBALPHABET).
* stbottomT: Дно стека (GRBALPHABET).
* rules: Массив правил (struct Rule\*).

Структура Greibach содержит некоторые функции:

* getRule(GRBALPHABET pnn, Rule& prule): Получает правило по заданному нетерминалу (GRBALPHABET) и сохраняет его в переданном объекте prule. Возвращает номер правила (short).
* getRule(short n): Возвращает правило (Rule) по заданному номеру (short).

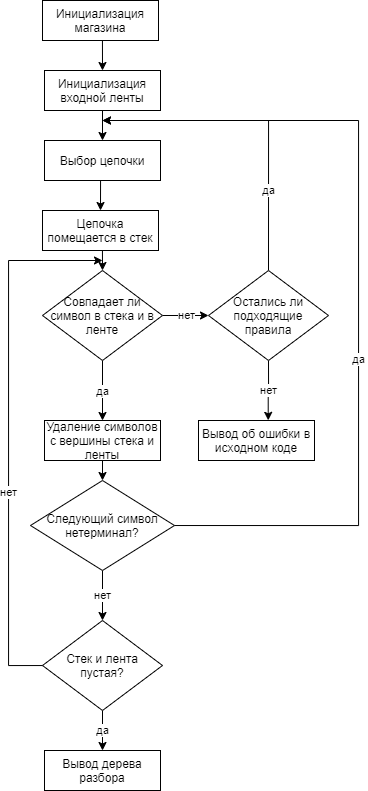
**Полный код функций и данных структур грамматики Грэйбах вынесен в Приложение Г.**

****4.5 Описание алгоритма синтаксического разбора****

**Алгоритм работы синтаксического разбора описан ниже.**

1. Инициализируется магазин (стек) с помещением стартового символа;
2. Создается входная лента на основе предварительно полученной таблицы лексем;
3. Выбирается цепочка, соответствующая текущему нетерминальному символу. Эта цепочка записывается в магазин в обратном порядке;
4. Если терминалы в ленте и на вершине стека совпадают, то данный терминал удаляется из ленты и стека. В противном случае происходит возврат к предыдущему сохраненному состоянию и выбор другой цепочки нетерминала;
5. Если в магазине встретился нетерминал, выполняется переход к пункту 3;
6. Если символ достигает дна стека и в этот момент лента пуста, то синтаксический анализ выполнен успешно. В противном случае генерируется исключение.

**Визуализация алгоритма в виде блок-схемы представлена на рисунке 4.2.**

****

**Рисунок 4.2 – Блок-схема алгоритма синтаксического разбора**

**Данный алгоритм представляет обобщённую модель работы магазинного автомата.**

****4.6 Параметры синтаксического анализатора****

**Синтаксический анализатор не имеет параметров, управляющих режимом работы синтаксического анализатора, поэтому он всегда работает в одном одинаковом режиме.**

****4.7 Структура и перечень сообщений синтаксического анализатора****

**Перечень ошибок возможных при работе синтаксического анализатора представлен в таблице 4.3.**

**Таблица 4.3 – Перечень ошибок синтаксического анализатора**

|  |  |
| --- | --- |
| **Код** | **Сообщение** |
| **600** | **Неверная структура программы** |
| **601** | **Ошибочный оператор** |
| **602** | **Ошибка в параметрах объявляемой функции** |

****4.8 Принцип при обработке 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.Все ошибки записываются в общую структуру ошибок.

4.В случае нахождения ошибки, после всей процедуры трассировки в протокол будет выведено диагностическое сообщение.

****4.9 Контрольный пример****

**Распечатка протокола синтаксического разбора представлена в приложении Г.**

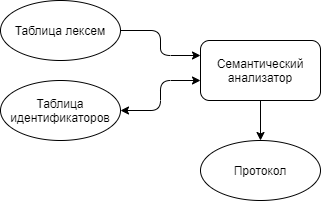
**В протоколе представлена трассировка синтаксического анализатора, которая содержит текущий набор символов в ленте и содержимое стека автомата для каждой итерации. В конце после успешного синтаксического разбора представлено дерево разбора для данного контрольного примера.**

****5. Разработка семантического анализатора****

****5.1 Структура семантического анализатора****

Семантический анализатора предназначен для анализа смысловой структуры программы и проверки ее семантической корректности. Он выполняет более высокоуровневую проверку программы, чем лексический и синтаксический анализ, и обнаруживает ошибки, связанные с типами данных, областями видимости, правильностью использования операторов и функций, а также другие семантические нарушения.

Графическое представление структуры семантического анализатора представлено ниже на рисунке 5.1



**Рисунок 5.1 – Структура семантического анализатора**

**Семантический анализатор принимает на вход таблицу лексем и таблицу идентификаторов. Также он может изменять таблицу идентификаторов при проверке правил для массивов.**

****5.2 Функции семантического анализатора****

**Некоторые проверки на различные правила семантического анализатора были распределены на различные фазы транслятора. Это сделано с целью упростить и ускорить работу самого транслятора. Например, при обращении к переменной нужно проверить, объявлена ли она. Это легко сделать на фазе лексического анализа, когда на данный момент имеется частично заполненная таблица идентификаторов, в которой записаны только объявленные переменные до обращения к текущей.**

**Некоторые проверки невозможно сделать на этапе трансляции, так как нельзя предугадать, что будет находится на том или ином этапе выполнения исходного кода. Поэтому некоторые вынесены на этап исполнения кода. Например, при выполнении того или иного оператора проверяются типы или индес и длина массива и так далее.**

**Полный перечень семантических проверок с указанием их фаз приведён ниже в таблице 5.1.**

**Таблица 5.1 – Перечень семантических правил**

|  |  |  |
| --- | --- | --- |
| **№** | **Правило** | **Фаза** |
| **1** | **Наличие main** | **Семантический** |
| **2** | **Идентификаторы усекаются до 5 символов** | **Лексический** |
| **3** | **Не допускается совпадения названий идентификаторов с ключевыми словами** | **Лексический** |
| **4** | **Все функции должны быть объявлены перед main** | **Синтаксический** |
| **5** | **Все идентификаторы должны быть обвялены в той же области видимости перед использованием** | **Лексический** |
| **6** | **Соответствие типов формальных и фактических параметров при вызове**  **функций** | **Исполнение** |
| **7** | **Соответствие типов формальных и фактических параметров при вызове**  **операторов** | **Исполнение** |
| **8** | **Все переменные должны инициализироваться** | **Семантический** |
| **9** | **Инструкции if и while должны содержать логический оператор** | **Семантический** |
| **10** | **При обращении к массиву с помощью оператора индексации, передаваемый индекс не должен выходить за пределы массива** | **Семантический (если индекс литерал)**  **Синтаксический**  **Исполнение(если индекс переменная )** |
| **11** | **Возвращаемое значение функции должно совпадать с типом функции** | **Семантический** |
| **12** | **При присваивании значения переменной, переменная должна быть явно указана перед оператором присваивания** | **Синтаксический** |
| **13** | **При объявлении массива или строки требуется явно указать количество элементов. (При объявлении строки без количества символов, то это переменная сможет содержать только один символ)** | **Синтаксический** |
| **14** | **Проверка на корректность скобок при выделении блоков кода(функции, блоки условных операторов и циклов)** | **Семантический** |
| **15** | **Маскимальный размер строки или массива 128 элементов** | **Семантический** |
| **16** | **В стеке не может находится более 32 элементов** | **Исполнение** |

**Семантический анализатор содержит набор функций, которые отвечают за проверку соответствующих правил семантики языка SES-2023 или заполие некоторой структуры для последующей её проверки на соответствие семантическим правилам.**

**Перечень функций с их описанием и полным прототипом приведены в таблице 5.2.**

**Таблица 5.2 – Перечень функции семантического анализатора**

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| **void addVars(**int i, LT::LexTable& lextable, IT::IdTable& idtable**)** | **Помечает определённую переменную в векторе инициализированной.** |
| **int checkIfWhile(int i, LT::LexTable& lextable, IT::IdTable& idtable)** | **Проверяет на наличие булевых операторов в условном и операторе цикла.** |
| **void addSizeToArray(int i, LT::LexTable& lextable, IT::IdTable& idtable)** | **Добавляет в таблицу информацию о количестве переменных массива** |
| **checkSizeofArray(int i, LT::LexTable& lextable, IT::IdTable& idtable)** | **Проверяет, не выходит ли индекс за пределы массива или строки** |
| **void checkVars()** | **Проверяет на наличие не проинициализированных переменных** |

**Алгоритм работы семантического анализа:**

1. Таблица лексем считывается посимвольно. Считывается таблица идентификаторов для последующей проверки, инициализируются ли каждая из них в коде;
2. При встрече лексемы оператора присваивания для какого-либо идентификатора, он помечается, как проинициализированный;
3. При встрече объявления массива или строки в таблицу идентификаторов добавляется размер данного массива или строки;
4. При встрече лексемы оператора взятия элемента по индексу проверяется, не выходит ли этот индекс за размер массива или строки;
5. При встрече оператора возраста из функции проверяется соответствие типов возвращаемого значения и типа функции;
6. При встрече if или while проверяется наличие булевых операторов.
7. При встрече main возводится флаг о его наличии. При повторной встрече будет выведена ошибка.

5.3 Структура и перечень сообщений семантического анализатора

При работе семантического анализатора могут возникать различные ошибки. Полный перечень кодов ошибок и их сообщения приведены ниже в таблице 5.3.

Таблица 5.3 – Перечень ошибок семантического анализа

|  |  |
| --- | --- |
| Код | Сообщение |
| 121 | Возможна только точка входа main |
| 122 | Выражение должно иметь тип bool |

Продолжение таблицы 5.3

|  |  |
| --- | --- |
| 123 | <набор\_перменных> Вышеперечисленный переменные должны инициализироваться |
| 124 | Выход за пределы массива |
| 125 | Несоответствие типа возвращаемого значения |
| 126 | Некорректный блок кода |
| 127 | Присвоить значение можно только переменной |
| 128 | Чтение доступно только для переменных |
| 129 | Максимальный размер строки/массива 128(не меньше нуля) |

****5.4 Принцип обработки ошибок****

**Принцип обработки ошибок точно такой же как на этапе лексического анализа (см 3.10).**

****5.5 Контрольный пример****

**В таблице 5.4 приведены примеры кода содержащих семантические ошибки и сообщения полученных**

**Таблица 5.4 – Примеры диагностических ошибок**

|  |  |
| --- | --- |
| **Код** | **Сообщение** |
| main  {  a 2 +  } | **Ошибка106:Идентификатор не определён**  **Строка:3** |
| main  {  int a  a writeline  } | **a**  **Ошибка123:Вышеперечисленные переменные не проициницилизированы** |
| main  {  array int arr[5]  arr[6]  } | **Ошибка124:Выход за пределы массива**  **Строка:4** |

****6. Вычисление выражений****

****6.1 Выражения, допускаемые языком****

**Для языка SES-2023 допустимыми являются выражения, записанные в виде обратной польской нотации. В выражениях могут использоваться целочисленные и строковые типы данных. В выражениях допускается использование функций с передачей параметров согласно stdcall. Записать длинное и сложное выражение в форме обратной польской нотации является сложной задачей, поэтому предусмотрена специальная директива препроцессора #polish <выражение\_в\_инфиксной\_форме>. Не допускается знак перевод каретки при записи выражения в инфиксной форме. Приоритет всех возможных операций представлен в таблице 6.1.**

Таблица 6.1 - Приоритеты операций

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| / | 4 |
| \* | 4 |
| - | 3 |
| + | 3 |
| ( ) | 1 |

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

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 работы семантического анализа:**

1. Исходное выражение делится на отдельные токены;
2. Все операнды переносятся в результирующую строку;
3. Операция записывается в стек, если он пуст;
4. Если встречается вызов функции то для записи все передаваемые параметры записываются в стек и начиная с вершины стека параметры записываются в результат. Это позволяет передавать параметры согласно соглашению stdcall;
5. Открывающая скобка помещается в стек;
6. **Закрывающая скобка выталкивает все операции до открывающей скобки;**
7. **Когда входная строка заканчивается, все оставшиеся операции выталкиваются из стека;**

****6.3 Программная реализация обработки выражении****

**Программная реализация обработки выражений представлена в Приложении Д.**

****6.4 Контрольный пример****

**В контрольном примере имеется несколько директив для преобразования, а также для этого раздела были сформированы сложные выражения для наглядного представления работы обработчика выражений. Они и результат обработки представлен в таблице 6.2.**

**Таблица 6.2 – Преобразование выражений**

|  |  |
| --- | --- |
| **Выражение** | **Результат** |
| #polish var1+var2 | **var1 var2 +** |
| #polish t-1 | **t 1 -** |
| #polish var1+var2/3+example(4)\*2\*(10-5) | **var1 var2 3 / 4 example 2 10 5 - \* \* + +** |

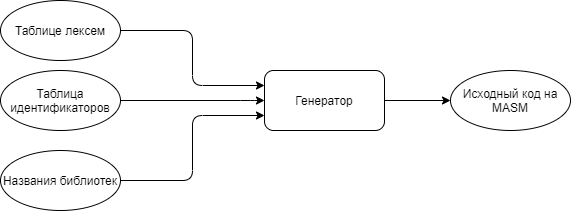
****7 Генерация кода****

****7.1 Структура генератора кода****

**Целевым языком для трансляции кода был выбран Microsoft Macro Assembler. Принцип генерации основывается на том, что любой оператор представляет собой определённый набор действий, который будет использоваться при каждом использовании оператора с передачей самих операндов из стека. Работа операторов цикла и условный оператора, основано на переходе по меткам и перехода по ним. Функции на языке SES-2023 транслируются на отдельные процедуры на языке Assembler, возврат значений которых происходит через регистры.**

**Генератор принимает на вход таблицу лексем, таблицу идентификаторов и набор названий подключённых библиотек.**

**Графическое представление генератора находится на рисунке 7.1.**

****

**Рисунок 7.1 – Структура транслятора**

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

**Любой тип данных представляет из себя байтовый(db) массив, первым элементом которого является тип данных, а остальные байты хранят само значение. Количество байт выделяемых под значение зависит от самого типа. Подробные пояснения о каждом типе данных языка SES-2023 представлены в таблице 7.1.**

**Таблица 7.1 – Типы данных**

|  |  |  |
| --- | --- | --- |
| **Тип данных** | **Представление на языке Assembler** | **Назначение** |
| **byte** | **db 1,0** | **Хранит целочисленное знаковое значение размером один байт.** |
| **int** | **db 2,4 dup(0)** | **Хранит целочисленное знаковое значение размером 4 байта** |
| **bool** | **db 3,0** | **Хранит логическое значение true, false.** |

Окончание таблицы 7.1

|  |  |  |
| --- | --- | --- |
| **Тип данных** | **Представление на языке Assembler** | **Назначение** |
| **string <id>[n]** | **db 7,n dup(-1)** | **Хранит набор символов длиной n. При объявлении заполняется -1.** |

****7.3 Статическая библиотека****

**Язык SES-2023 предусматривает несколько статических библиотек, которые содержат функции. Библиотеки написаны на языке С++. Чтобы подключить библиотеку нужно воспользоваться командой препроцессора #include и указать название библиотеки. В случае если название неправильное, то директива просто игнорируется и удаляется со всеми остальными директивами из исходного кода.**

**Полный перечень функций стандартных библиотек представлен в 1.18.**

**Так же статические библиотеки на языке С++ использовались для реализации ошибок во время исполнения программы и для ввода и вывода данных. Полная информация по данным библиотекам предоставлена в таблице 7.2.**

**Таблица 7.2 ­– Функции статических библиотек**

|  |  |  |
| --- | --- | --- |
| **Библиотека** | **Функция** | **Описание** |
| **CallErrors.lib** | void ErrorType(char\* str) | **Принимает номер строки, где произошла ошибка и выводит окно, о том что типы параметров или операндов несоответствуют допустимым.** |
| void ErrorStackEmpty(char\* str) | **Принимает номер строки, где произошла ошибка и выводит окно, о том что в стеке недостаточно параметров для оператора или вызова функции.** |
| void ErrorStackOverflow(char\* str) | **Принимает номер строки, где произошла ошибка и выводит окно, о том что стек переполнен.** |
| void ErrorIncorInd(char\* str) | **Принимает номер строки, где произошла ошибка и выводит окно, о том что произошёл выход за пределы массива или строки.** |

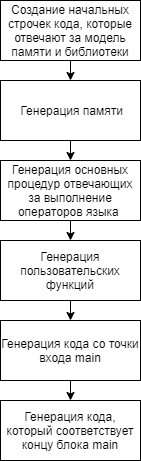
Продолжение таблицы 7.2

|  |  |  |
| --- | --- | --- |
| **Библиотека** | **Функция** | **Описание** |
| **write.lib** | void writeString(char\* type, char\* str) | **Принимает тип и значение, и выводит его в соответствии с типом на консоль без перехода на новую строку.** |
| void writeLineString(char\* type, char\* str) | **Принимает тип и значение, и выводит его в соответствии с типом на консоль с переходом на новую строку.** |
| void readCon(char\*type,char\* value) | **Принимает тип и указатель на область памяти, куда будет произведена запись. Если введённое значение соответствует переданному типу, то происходит запись в данную область, в противном случае выводится окно о несоответствии значения типу.** |

****7.4 Особенности генерации кода****

**Особенностью генератора кода языка SES-2023 является проверка некоторых семантических правил на этапе исполнения. Например, при выполнении кода, когда вызываются операторы или функции происходит проверка типов. В случае отрицательного результата программа прекращает выполнения и выводит соответствующее сообщение. Также проверка выполняется на присутствие достаточного количества элементов в стеке, а также на переполнение стека.**

**Обобщённый алгоритм генератора кода представлен на рисунке 7.2.**

****

**Рисунок 7.2 – Обобщённый алгоритм генерации кода**

**В модуле генератора присутвует только две функции**Первая функция **generate memory(IT::IdTable\* idtable)** предназначена для генерации памяти для переменных и литералов, а вторая для генерации основного исходного кода.

Вторая функция **:,** void Generator (LT::LexTable& lextable, IT::IdTable& idtable, Parm::PARM parms, char\*\* libraries) предназначена для генерации основного исоплняемого кода гланой и пользовательских функций.

В генеарторе определено множество функций, написанных на языке ассемблер, которые предназначены для выполнения различных арифмитических операций. Например, для сложения была написана функция PLUS, исходный код которой можно увидет в результате генерации контрольного примера в приложении Е.

****7.5 Параметры, управляющие генераций кода****

**У генератора отсутствуют параметры, которые управляли бы самим процессом генерации. Данный модуль транслятора работает всё время в одном и том же режиме.**

****7.6 Контрольный пример****

**Результат генерации кода в Приложении Е.**

****8Тестирование приложения****

****8.1 Общие положения****

При возникновении ошибки на каком-либо этапе трансляции, она обрабатывается в главном файле программ: ошибка выводится на консоль и записывается в протокол работы. Запись в протокол не осуществляется на этапе исполнения.

8.2 Результаты тестирования

В таблице 8.1 приведены ошибки, генерируемые в процессе считывания входного файла, а также в процессе лексического, синтаксического и семантического анализов.

Таблица 8.1 – Результаты тестирования транслятора

|  |  |  |
| --- | --- | --- |
| Исходный код | Ошибка | Этап |
| main{  #polish (a+b)/(a-b  } | Ошибка2:Ошибочное выражние  Строка:3 Позиция:18 | Препроцессор |
| #include srting.lib  main  {  hello  } | Ошибка3:Неизвестная библиотека  Строка:1 Позиция:10 | Препроцессор |
| main  {  int 1q  1 1q $  } | Ошибка113:Недопустимый идентификатор  Строка:3 Позиция:7 | Лексический анализ |
| function int fun(int a)  {  a writeline  a return  }  main{  id  'hello' writeline  } | Ошибка115:Идентификатор не определён  Строка:8 Позиция:3 | Лексический анализ |

Продолжение таблицы 8.1

|  |  |  |
| --- | --- | --- |
| Исходный код | Ошибка | Этап |
| main{  int id  25 id $  byte id  'hello' writeline} | Ошибка114:Дублирование идентификатора  Строка:4 Позиция:8 | Лексический анализ |
| main{  int id  id[3]  } | Ошибка116:Некорректное использование индекса  Строка:3 Позиция:3 | Лексический анализ |
| function int fun(int a, int q, int w, int e, int r, int t, int y, int u, int i)  {  int b  5 b $  b a +  b $  b return  }  main  {  1 1 1 1 1 1 1 1 1 fun  write  } | Ошибка120:Превышено максимальное допустимое количество параметров у функции  Строка:1 Позиция:79 | Лексический анализ |
| main{  int n  10 n $  array int id[n]  } | 601: строка 10,Ошибочный оператор | Синтаксический анализ |
| int fun(int a)  {a writeline  a return  }{  'hello' writeline  } | 600: строка 1,Неверная структура программы  Ошибка605:Синтаксически анализ завершён досрочно | Синтаксический анализ |

Продолжение таблицы 8.1

|  |  |  |
| --- | --- | --- |
| Исходный код | Ошибка | Этап |
| main{  int n  10 n $  if()  {  'hello' writeline  }  } | 601: строка 4,Ошибочный оператор | Синтаксический анализ |
| function int examp (function int a)  {  a return  }  main{  int n  10 n $  if(10 2 +)  {  'hello' writeline  }} | 603: строка 1,Ошибка в параметрах функции  Ошибка605:Синтаксически анализ завершён досрочно | Синтаксический анализ |
| main{  int n  10 n $  if(10 2 +)  {  'hello' writeline  }} | Ошибка122:Выражение должно иметь тип bool  Строка:4 | Семантический анализ |
| main  {  int a  a writeline} | **a**  **Ошибка123:Вышеперечисленные переменные не проициницилизированы** | Семантический анализ |
| main{  array int arr[3]  arr[3] write} | Ошибка124:Выход за пределы массива  Строка:3 | Семантический анализ |
| function int examp (int a)  {string t  10 t $  t return}  main{  examp write} | Ошибка125:Несоответствие типа возвращаемого значения  Строка:5 | Семантический анализ |

Продолжение таблицы 8.1

|  |  |  |
| --- | --- | --- |
| Исходный код | Ошибка | Этап |
| main{  int w  10 w $  if(15 w>)  {if(true)  {  'hello' write  } | Ошибка126:Некорректный блок кода | Семантический анализ |
| function int fun(int a)  {  a writeline  0 return  }  main  {  10 fun $  } | Ошибка127:Присвоить можно значение можно только переменной  Строка:8 | Семантический анализ |
| function int fun(int a)  {  a writeline  0 return  }  main  {  10 fun $  } | Ошибка128:Чтение доступно только для переменной  Строка:8 | Семантически анализ |
| main  {  string str[-1]  str read  } | Ошибка129:Максимальный размер строки/массива 128(не меньше нуля)  Строка:3 | Семантический анализ |
| main  {  '1' 2 +  writeline  } |  | Исполнение |

Окончание таблицы 8.1

|  |  |  |
| --- | --- | --- |
| Исходный код | Ошибка | Этап |
| main  {  1 +  writeline  } |  | Исполнение |
| main  {  while(true)  {  1 1 +  }  } |  | Исполнение |

****Заключение****

В ходе проделанной работы был разработан транслятор для языка программирования SES-2023.

В пояснительной записке описана реализация поставленных в рамках курсового проекта ряда задач:

* реализованы 3 арифметических оператора и 5 операторов сравнения;
* реализованы четыре типа данных;
* поддерживается операторы вывода и ввода;
* присутствует 2 подключаемых стандартных библиотек;
* реализована функция лексического сравнения строк;
* реализован условный оператор;
* реализован оператор цикла;
* реализовано преобразование типов для из 1 байтового целочисленного в 4 байтовое целочисленное.

Таким образом была достигнута поставленная цель по разработке компилятора SES-2023, были учтены все требования, все задачи курсового проекта выполнены. Также были в проекте реализованы функции, которые не предусматриваются начальными задачами.

Список использованных источников

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

2. Ахо, А. Теория синтаксического анализа, перевода и компиляции /А. Ахо, Дж. Ульман. – Москва : Мир, 1998. – Т. 2 : Компиляция. - 487 с.

3. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.

4. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.

5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с

Приложение A

**Листинг 1 – Контрольный пример на языке SES-2023**

|  |
| --- |
| #include string.lib  #include math.lib  function int example(int a)  {  int b  5 b $  b a +  b $  b return  }  main  {  'hello world' print  int t  3 factor  t $  2 t power  dup writeline  t $  11  example t $  if(15 t <)  {  't=' print  t writeline  }  array int arr[6]  while (0 t <)  {  #polish t-1  t $  t writeline  if(t 6 <)  {  t arr[t] $  }  }  byte ind  0 ind $  'Вывод массива:' print  while (ind 6 <)  {  arr[ind] writeline  ind 1 +  ind $  }  string str[10]  'Введите строку (не больше 10 может поместиться)' print  str read  'Длина строки' print  str length writeline  'Сумма чисел' print  'Первая переменаня' print  int var1  var1 read  'Вторая переменная' print  int var2  var2 read  #polish var1+var2  'Результат' print  print  'Результат выражения var1+var2/3+example(4)\*2\*(10-5)' print  #polish var1+var2/3+example(4)\*2\*(10-5)  print  } |

****Приложение Б****

**Структуры таблиц лексем и идентификаторов и цепочки разборов лексического анализатора**

|  |
| --- |
| #define FST\_INT FST::FST \_int(str,\  4,\  FST::NODE(1, FST::RELATION('i', 1)),\  FST::NODE(1, FST::RELATION('n', 2)),\  FST::NODE(1, FST::RELATION('t', 3)),\  FST::NODE()\  );  #define FST\_BYTE FST::FST \_byte(str,\  5,\  FST::NODE(1,FST::RELATION('b',1)),\  FST::NODE(1,FST::RELATION('y',2)),\  FST::NODE(1,FST::RELATION('t',3)),\  FST::NODE(1,FST::RELATION('e',4)),\  FST::NODE()\  );  #define FST\_BOOL FST::FST \_bool(str,\  5,\  FST::NODE(1,FST::RELATION('b',1)),\  FST::NODE(1,FST::RELATION('o',2)),\  FST::NODE(1,FST::RELATION('o',3)),\  FST::NODE(1,FST::RELATION('l',4)),\  FST::NODE()\  );  #define FST\_STRING FST::FST \_string(str,\  7,\  FST::NODE(1, FST::RELATION('s', 1)),\  FST::NODE(1, FST::RELATION('t', 2)),\  FST::NODE(1, FST::RELATION('r', 3)),\  FST::NODE(1, FST::RELATION('i', 4)),\  FST::NODE(1, FST::RELATION('n', 5)),\  FST::NODE(1, FST::RELATION('g', 6)),\  FST::NODE()\  );  #define FST\_ARRAY FST::FST \_array(str,\  6,\  FST::NODE(1,FST::RELATION('a',1)),\  FST::NODE(1,FST::RELATION('r',2)),\  FST::NODE(1,FST::RELATION('r',3)),\  FST::NODE(1,FST::RELATION('a',4)),\  FST::NODE(1,FST::RELATION('y',5)),\  FST::NODE()\  );  #define FST\_FUNC FST::FST \_function(str,\  9,\  FST::NODE(1, FST::RELATION('f', 1)),\  FST::NODE(1, FST::RELATION('u', 2)),\  FST::NODE(1, FST::RELATION('n', 3)),\  FST::NODE(1, FST::RELATION('c', 4)),\  FST::NODE(1, FST::RELATION('t', 5)),\  FST::NODE(1, FST::RELATION('i', 6)),\  FST::NODE(1, FST::RELATION('o', 7)),\  FST::NODE(1, FST::RELATION('n', 8)),\  FST::NODE()\  );  #define FST\_RETURN FST::FST \_return(str,\  7,\  FST::NODE(1,FST::RELATION('r',1)),\  FST::NODE(1,FST::RELATION('e',2)),\  FST::NODE(1,FST::RELATION('t',3)),\  FST::NODE(1,FST::RELATION('u',4)),\  FST::NODE(1,FST::RELATION('r',5)),\  FST::NODE(1,FST::RELATION('n',6)),\  FST::NODE()\  );  #define FST\_MAIN FST::FST \_main(str,\  5,\  FST::NODE(1, FST::RELATION('m', 1)),\  FST::NODE(1, FST::RELATION('a', 2)),\  FST::NODE(1, FST::RELATION('i', 3)),\  FST::NODE(1, FST::RELATION('n', 4)),\  FST::NODE()\  );  #define FST\_WRITE FST::FST \_write(str,\  6,\  FST::NODE(1,FST::RELATION('w',1)),\  FST::NODE(1,FST::RELATION('r',2)),\  FST::NODE(1,FST::RELATION('i',3)),\  FST::NODE(1,FST::RELATION('t',4)),\  FST::NODE(1,FST::RELATION('e',5)),\  FST::NODE()\  );  #define FST\_WRITELINE FST::FST \_writeline(str,\  10,\  FST::NODE(1,FST::RELATION('w',1)),\  FST::NODE(1,FST::RELATION('r',2)),\  FST::NODE(1,FST::RELATION('i',3)),\  FST::NODE(1,FST::RELATION('t',4)),\  FST::NODE(1,FST::RELATION('e',5)),\  FST::NODE(1,FST::RELATION('l',6)),\  FST::NODE(1,FST::RELATION('i',7)),\  FST::NODE(1,FST::RELATION('n',8)),\  FST::NODE(1,FST::RELATION('e',9)),\  FST::NODE()\  );  #define FST\_READ FST::FST \_read(str,\  5,\  FST::NODE(1,FST::RELATION('r',1)),\  FST::NODE(1,FST::RELATION('e',2)),\  FST::NODE(1,FST::RELATION('a',3)),\  FST::NODE(1,FST::RELATION('d',4)),\  FST::NODE()\  );  #define FST\_TRUE FST::FST \_true(str,\  5,\  FST::NODE(1,FST::RELATION('t',1)),\  FST::NODE(1,FST::RELATION('r',2)),\  FST::NODE(1,FST::RELATION('u',3)),\  FST::NODE(1,FST::RELATION('e',4)),\  FST::NODE()\  );  #define FST\_FALSE FST::FST \_false(str,\  6,\  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('e',5)),\  FST::NODE()\  );  #define FST\_WHILE FST::FST \_while(str, \  6, \  FST::NODE(1, FST::RELATION('w', 1)), \  FST::NODE(1, FST::RELATION('h', 2)), \  FST::NODE(1, FST::RELATION('i', 3)), \  FST::NODE(1, FST::RELATION('l', 4)), \  FST::NODE(1, FST::RELATION('e', 5)), \  FST::NODE() \  );  #define FST\_IF FST::FST \_if(str, \  3, \  FST::NODE(1, FST::RELATION('i', 1)), \  FST::NODE(1, FST::RELATION('f', 2)), \  FST::NODE() \  );  #define FST\_DUP FST::FST \_dup(str, \  4, \  FST::NODE(1, FST::RELATION('d', 1)), \  FST::NODE(1, FST::RELATION('u', 2)), \  FST::NODE(1, FST::RELATION('p', 3)), \  FST::NODE() \  );  #define FST\_DROP FST::FST \_drop(str, \  5, \  FST::NODE(1, FST::RELATION('d', 1)), \  FST::NODE(1, FST::RELATION('r', 2)), \  FST::NODE(1, FST::RELATION('o', 3)), \  FST::NODE(1, FST::RELATION('p', 4)), \  FST::NODE() \  );  #define FST\_SWAP FST::FST \_swap(str, \  5, \  FST::NODE(1, FST::RELATION('s', 1)), \  FST::NODE(1, FST::RELATION('w', 2)), \  FST::NODE(1, FST::RELATION('a', 3)), \  FST::NODE(1, FST::RELATION('p', 4)), \  FST::NODE() \  );  #define FST\_OVER FST::FST \_over(str, \  5, \  FST::NODE(1, FST::RELATION('o', 1)), \  FST::NODE(1, FST::RELATION('v', 2)), \  FST::NODE(1, FST::RELATION('e', 3)), \  FST::NODE(1, FST::RELATION('r', 4)), \  FST::NODE() \  );  #define FST\_DEPTH FST::FST \_depth(str, \  6, \  FST::NODE(1, FST::RELATION('d', 1)), \  FST::NODE(1, FST::RELATION('e', 2)), \  FST::NODE(1, FST::RELATION('p', 3)), \  FST::NODE(1, FST::RELATION('t', 4)), \  FST::NODE(1, FST::RELATION('h', 5)), \  FST::NODE() \  );  #define FST\_CLEAR FST::FST \_clear(str, \  6, \  FST::NODE(1, FST::RELATION('c', 1)), \  FST::NODE(1, FST::RELATION('l', 2)), \  FST::NODE(1, FST::RELATION('e', 3)), \  FST::NODE(1, FST::RELATION('a', 4)), \  FST::NODE(1, FST::RELATION('r', 5)), \  FST::NODE() \  );  #define FST\_LITERAL FST::FST literal\_int(str,\  2,\  FST::NODE(20,\  FST::RELATION('0', 0), 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('8', 0),\  FST::RELATION('9', 0), FST::RELATION('0', 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::NODE()\  );  #define FST\_IDENF FST::FST idenf(str,\  2,\  FST::NODE(118,\  FST::RELATION('a', 1), FST::RELATION('a', 0), FST::RELATION('b', 1), FST::RELATION('b', 0),\  FST::RELATION('c', 1), FST::RELATION('c', 0), FST::RELATION('d', 1), FST::RELATION('d', 0), FST::RELATION('e', 1), FST::RELATION('e', 0),\  FST::RELATION('f', 1), FST::RELATION('f', 0), FST::RELATION('g', 1), FST::RELATION('g', 0), FST::RELATION('h', 0), FST::RELATION('h', 1), FST::RELATION('i', 0), FST::RELATION('i', 1),\  FST::RELATION('j', 0), FST::RELATION('j', 1), FST::RELATION('k', 0), FST::RELATION('k', 1), FST::RELATION('l', 0), FST::RELATION('l', 1),\  FST::RELATION('m', 0), FST::RELATION('m', 1), FST::RELATION('n', 0), FST::RELATION('n', 1), FST::RELATION('o', 0), FST::RELATION('o', 1),\  FST::RELATION('p', 0), FST::RELATION('p', 1), FST::RELATION('q', 0), FST::RELATION('q', 1), FST::RELATION('r', 0), FST::RELATION('r', 1),\  FST::RELATION('s', 0), FST::RELATION('s', 1), FST::RELATION('t', 0), FST::RELATION('t', 1), FST::RELATION('u', 0), FST::RELATION('u', 1),\  FST::RELATION('v', 0), FST::RELATION('v', 1), FST::RELATION('w', 0), FST::RELATION('w', 1), FST::RELATION('x', 0), FST::RELATION('x', 1),\  FST::RELATION('y', 0), FST::RELATION('y', 1), FST::RELATION('z', 0), FST::RELATION('z', 1),FST::RELATION('A', 1), FST::RELATION('A', 0), FST::RELATION('B', 1), FST::RELATION('B', 0),\  FST::RELATION('C', 1), FST::RELATION('C', 0), FST::RELATION('D', 1), FST::RELATION('D', 0), FST::RELATION('E', 1), FST::RELATION('E', 0),\  FST::RELATION('F', 1), FST::RELATION('F', 0), FST::RELATION('G', 1), FST::RELATION('G', 0), FST::RELATION('H', 0), FST::RELATION('H', 1), FST::RELATION('I', 0), FST::RELATION('I', 1),\  FST::RELATION('J', 0), FST::RELATION('J', 1), FST::RELATION('K', 0), FST::RELATION('K', 1), FST::RELATION('L', 0), FST::RELATION('L', 1),\  FST::RELATION('M', 0), FST::RELATION('M', 1), FST::RELATION('N', 0), FST::RELATION('N', 1), FST::RELATION('O', 0), FST::RELATION('O', 1),\  FST::RELATION('P', 0), FST::RELATION('P', 1), FST::RELATION('Q', 0), FST::RELATION('Q', 1), FST::RELATION('R', 0), FST::RELATION('R', 1),\  FST::RELATION('S', 0), FST::RELATION('S', 1), FST::RELATION('T', 0), FST::RELATION('T', 1), FST::RELATION('U', 0), FST::RELATION('U', 1),\  FST::RELATION('V', 0), FST::RELATION('V', 1), FST::RELATION('W', 0), FST::RELATION('W', 1), FST::RELATION('X', 0), FST::RELATION('X', 1),\  FST::RELATION('Y', 0), FST::RELATION('Y', 1), FST::RELATION('Z', 0), FST::RELATION('Z', 1), FST::RELATION('0', 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::NODE()\  ); |

**Листинг 1 – Цепочки разбора**

|  |
| --- |
| struct Entry  {  char lexema[LEXEMA\_FIXSIZE];  int sn;//номер строки в исходном коде  int idxTI;// строки в IT  };  struct LexTable  {  int maxsize;  int size;  Entry\* table;    }; |

**Листинг 1 – Структуры таблицы идентификаторов**

|  |
| --- |
| struct Entry // строка таблицы идентификаторов  {  int idxfirstLE; // индекс первой строки в таблице лексем  char id[ID\_MAXSIZE]; // идентификатор (автоматически усекается до ID\_MAXSIZE)  IDDATATYPE iddatatype; // тип данных  IDTYPE idtype; // тип идентификатора  Entry\* scope;  union  {  int vint; // значение integer  struct  {  int len; // кол-во символов в string  char str[TI\_STR\_MAXSIZE - 1]; // символы string  } vstr[TI\_STR\_MAXSIZE]; // значение string  } value; // значение идентификатора  struct IdTable // экземпляр таблицы идентификаторов  {  int maxsize; // емкость таблицы идентификаторов < TI\_MAXSIZE  int size; // текущий размер таблицы идентификаторов < maxsize  Entry\* table; // массив строк таблицы идентификаторов  }; |

**Листинг 2 – Структуры таблицы идентификаторов**

****Приложение В****

**Таблица лексем**

|  |
| --- |
| 1  3 fti(ti)  4 {  5 ti  6 li$  7 ii+  8 i$  9 iq  10 }  12 m  13 {  14 li  15 ti  16 li  17 i$  18 lii  19 dx  20 i$  21 l  22 ii$  23 z(li<)  24 {  25 li  26 ix  27 }  28 atil@  29 h(li<)  30 {  31 il-  32 i$  33 ix  34 z(il<)  35 {  36 iii@$  37 }  38 }  39 ti  40 li$  41 li  42 h(il<)  43 {  44 ii@x  45 il+  46 i$  47 z(l)  48 {  49 li  50 }  51 }  52 til@  53 li  54 in  55 li  56 iix  57 li  58 li  59 ti  60 in  61 li  62 ti  63 in  64 ii+  65 li  66 i  67 li  68 iil/lilll-\*\*++  69 i  70 } |

**Листинг 2 – Таблица лексем**

|  |
| --- |
| id datatype idtype Line value Scope idTi  length INT F 0 1 0  compare BOOL F 0 2 0  compS BOOL F 0 2 0  print F 0 0  power INT F 0 2 0  factor INT F 0 1 0  example INT F 3 0 3  a INT P 3 0 example 3  b INT V 5 0 example 5  L0 BYTE L 6 5 example 6  main INT F 12 0 12  L2 STR L 14 'hello world' main 14  t INT V 15 0 main 15  L3 BYTE L 16 3 main 16  L4 BYTE L 18 2 main 18  L5 BYTE L 21 11 main 21  L6 BYTE L 23 15 main 23  L8 STR L 25 't=' main 25  arr INT A 28 0 main 28  L9 BYTE L 28 6 main 28  L10 BYTE L 29 0 main 29  L11 BYTE L 31 1 main 31  ind BYTE V 39 0 main 39  L13 STR L 41 'Вывод массива:' main 41  true BOOL L 47 1 main 47  L16 STR L 49 'hello' main 49  str STR V 52 main 52  L17 BYTE L 52 10 main 52  L19 STR L 53 'Введите строку(не больше 10 может поместиться)' main 53  L21 STR L 55 'Длина строки' main 55  L23 STR L 57 'Деление чисел' main 57  L25 STR L 58 'Первая переменаня' main 58  var1 INT V 59 0 main 59  L27 STR L 61 'Вторая переменная' main 61  var2 INT V 62 0 main 62  L29 STR L 65 'Результат' main 65  L31 STR L 67 'Результат выражения var1+var2/3+example(4)\*2\*(10-5)' main 67  L32 BYTE L 68 4 main 68 |

**Листинг 3 – Таблица идентификторов**

****Приложение Г****

**Структуры и функции синтаксического анализатора. Протокол работы синтаксического анализатора**

|  |
| --- |
| struct Rule //правило в грамматике Грейбах  {  GRBALPHABET nn; //нетерминалы (левый символ правила) < 0  int iderror; //идентификатор диагностического сообщения  short size; //количество цепочек - правых частей правила  struct Chain //цепочка(правая часть правила)  {  short size; //длина цепочки  GRBALPHABET\* nt; //цепочка терминалов (>0) и нетермеминалов (<0)  Chain()  {  this->size = 0; //количество символов в цепочке  this->nt = 0; //символы (терминал или нетерминал)  };  Chain(short psize, GRBALPHABET s, ...); //кол-во символов в цепочке/символы(терминалы и нетерминалы)  char\* getCChain(char\* b); //получить правую сторону правила  static GRBALPHABET T(char t) { return GRBALPHABET(t); }; //терминал  static GRBALPHABET N(char n) { return -GRBALPHABET(n); }; //не терминал  static bool isT(GRBALPHABET s) { return s > 0; }; //терминал?  static bool isN(GRBALPHABET s) { return !isT(s); }; //не терминал?  static char alphabet\_to\_char(GRBALPHABET s) //GRBALPHABET->char (обращение по указателю к полю символа)  {  return isT(s) ? char(s) : char(-s);  };  }\*chains; //массив цепочек - правых частей правила  Rule()  {  this->nn = 0x00;  this->size = 0;  }  Rule(GRBALPHABET pnn, int iderroe, short psize, Chain c, ...);  //(нетерминал(< 0); идентификатор диагностического сообщ - я; кол - во цепочек - правых частей правила; множество цепочек - правых частей правила)  char\* getCRule(char\* b, short nchain); //получить парвило в виде N->цепочка (для распечатки) (буфер;номер цепочки(правой части) в правиле)  short getNextChain(GRBALPHABET t, Rule::Chain& pchain, short j); //получить следующую за j подходящую цепочку, вернуть её номер или -1 (первый символ цепочки;возвращаемая цепочка; номер цепочки)  };  struct Greibach //грамматика Грейбах  {  short size; //количество правил  GRBALPHABET startN; //стартовый символ  GRBALPHABET stbottomT; //дно стека  Rule\* rules; //множество правил  Greibach() { this->size = 0; this->startN = 0; this->stbottomT = 0; this->rules = 0; };  Greibach(GRBALPHABET pstartN, GRBALPHABET pstbottomT, short psize, Rule r, ...);  short getRule(GRBALPHABET pnn, Rule& prule);  Rule getRule(short n);  };  struct MfstState //состояние автомата для сохранения  {  short lenta\_position; //состояние автомата для сохранения  short nrule; //номер текущего правила  short nrulechain; //номер текущей цепчки, текущего правила  MFSTSTSTACK st; //стек автомата  MfstState();  MfstState(short pposition, MFSTSTSTACK pst, short pnrulechain); //(позиция на ленте;стек автомата; номер текущей цепочки текущего правила)  MfstState(short pposition, MFSTSTSTACK pst, short pnrule, short pnrulechain);//(позиция на ленте;стек автомата; номер текущего правила; номер текущей цепочки текущего правила)  };  class my\_stack\_MfstState :public std::stack<MfstState> {  public:  using std::stack<MfstState>::c;  };  struct Mfst //магазинный автомат  {  enum RC\_STEP { //код вовзрата 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, //неизвествный нетерминальный символ грамматики  TS\_OK, //тек. символ ленты == вершине стека, продвинулась лента, pop стека  TS\_NOK, //тек. символ ленты != вершине стека, продвинулась лента, pop стека  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);  } diagnosis[MFST\_DIAGN\_NUMBER]; // последние самые глубокие сообщения  GRBALPHABET\* lenta; //перекодированныя (TN/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::Greibach grebach; //грамматика Грейбах  LT::LexTable lex; //результат работы лексического анализатора  MFSTSTSTACK st; //стек автомата  my\_stack\_MfstState storestate; //стек для хранения состояний  Mfst();  Mfst(LT::LexTable& plex, GRB::Greibach pgrebach);  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-ю строку диагностики или 0х00  bool savestate(Log::LOG& log); //сохранить состояние автомата  bool resetstate(Log::LOG& log); //восстановить состояние автомата  bool push\_chain(GRB::Rule::Chain chain); //поместить цепочку правила в стек  RC\_STEP step(Log::LOG& log); //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 послдеовательность правил  struct Deducation  {  short size;  short\* nrules;  short\* nrulechains;  Deducation()  {  size = 0;  nrules = 0;  nrulechains = 0;  };  }deducation;  bool savededucation();  }; |

**Листинг 1– Структуры синтаксического анализатора**

|  |
| --- |
| MfstState::MfstState()  {  lenta\_position = 0;  nrule = -1;  nrulechain = -1;  };  MfstState::MfstState(short pposition, MFSTSTSTACK pst, short pnrulechain)  {  lenta\_position = pposition;  st = pst;  nrulechain = pnrulechain;  };  MfstState::MfstState(short pposition, MFSTSTSTACK pst, short pnrule, short pnrulechain)  {  lenta\_position = pposition;  st = pst;  nrule = pnrule;  nrulechain = pnrulechain;  };  Mfst::MfstDiagnosis::MfstDiagnosis()  {  lenta\_position = -1;  rc\_step = SURPRISE;  nrule = -1;  nrule\_chain = -1;  };  Mfst::MfstDiagnosis::MfstDiagnosis(short plenta\_position, RC\_STEP prc\_step, short pnrule, short pnrule\_chain)  {  lenta\_position = plenta\_position;  rc\_step = prc\_step;  nrule = pnrule;  nrule\_chain = pnrule\_chain;  };  Mfst::Mfst() { lenta = 0; lenta\_size = lenta\_position = 0; };  Mfst::Mfst(LT::LexTable& lextable, GRB::Greibach pgrebach)  {  grebach = pgrebach;  lex = lextable;  lenta = new short[lenta\_size = lex.size];  for (int k = 0; k < lenta\_size; k++)  lenta[k] = GRB::Rule::Chain::T(lex.table[k].lexema[0]);  lenta\_position = 0;  st.push(grebach.stbottomT);  st.push(grebach.startN);  nrulechain = -1;  }  Mfst::RC\_STEP Mfst::step()  {  RC\_STEP rc = SURPRISE;  if (lenta\_position < lenta\_size)  {  if (GRB::Rule::Chain::isN(st.top()))  {  GRB::Rule rule;  if ((nrule = grebach.getRule(st.top(), rule)) >= 0)  {  GRB::Rule::Chain chain;  if ((nrulechain = rule.getNextChain(lenta[lenta\_position], chain, nrulechain + 1)) >= 0)  {  MFST\_TRACE1  savestate(); st.pop(); push\_chain(chain); rc = NS\_OK;  MFST\_TRACE2  }  else  {  MFST\_TRACE4("TNS\_NORULECHAIN/NS\_NORULE")  savediagnosis(NS\_NORULECHAIN); rc = resetstate() ? NS\_NORULECHAIN : NS\_NORULE;  };  }  else rc = NS\_ERROR;  }  else if ((st.top() == lenta[lenta\_position]))  {  lenta\_position++; st.pop(); nrulechain = -1; rc = TS\_OK;  MFST\_TRACE3  }  else { MFST\_TRACE4(TS\_NOK / NS\_NORULECHAIN) rc = resetstate() ? TS\_NOK : NS\_NORULECHAIN; };  }  else  {  rc = LENTA\_END;  MFST\_TRACE4(LENTA\_END);  };  return rc;  };  bool Mfst::push\_chain(GRB::Rule::Chain chain)  {  for (int k = chain.size - 1; k >= 0; k--) st.push(chain.nt[k]);  return true;  };  bool Mfst::savestate()  {  storestate.push(MfstState(lenta\_position, st, nrule, nrulechain));  MFST\_TRACE6("SAVESTATE:", storestate.size());  return true;  };  bool Mfst::resetstate()  {  bool rc = false;  MfstState state;  if (rc = (storestate.size() > 0))  {  state = storestate.top();  lenta\_position = state.lenta\_position;  st = state.st;  nrule = state.nrule;  nrulechain = state.nrulechain;  storestate.pop();  MFST\_TRACE5("RESSTATE")  MFST\_TRACE2  };  return rc;  };  bool Mfst::savediagnosis(RC\_STEP prc\_step)  {  bool rc = false;  short k = 0;  while (k < MFST\_DIAGN\_NUMBER && lenta\_position <= diagnosis[k].lenta\_position)  k++;  if (rc = (k < MFST\_DIAGN\_NUMBER))  {  diagnosis[k] = MfstDiagnosis(lenta\_position, prc\_step, nrule, nrulechain);  for (int i = k + 1; i < MFST\_DIAGN\_NUMBER; i++)  diagnosis[i].lenta\_position = -1;  }  return rc;  };  bool Mfst::start(Log::LOG& log)  {  bool rc = false;  RC\_STEP rc\_step = SURPRISE;  char buf[MFST\_DIAGN\_MAXSIZE]{};  rc\_step = step();  while (rc\_step == NS\_OK || rc\_step == NS\_NORULECHAIN || rc\_step == TS\_OK || rc\_step == TS\_NOK)  rc\_step = step();  switch (rc\_step)  {  case LENTA\_END:  {  MFST\_TRACE4("------>LENTA\_END")  std::cout << "------------------------------------------------------------------------------------------ ------" << std::endl;  sprintf\_s(buf, MFST\_DIAGN\_MAXSIZE, "%d: всего строк %d, синтаксический анализ выполнен без ошибок", 0, lex.table[lex.size - 1].sn);  std::cout << std::setw(4) << std::left << 0 << "всего строк " << lex.table[lex.size - 1].sn << ", синтаксический анализ выполнен без ошибок" << std::endl;  rc = true;  break;  }  case NS\_NORULE:  {  MFST\_TRACE4("------>NS\_NORULE")  std::cout << "------------------------------------------------------------------------------------------ ------" << std::endl;  std::cout << getDiagnosis(0, buf) << std::endl;  \*log.stream << getDiagnosis(0, buf) << std::endl;  std::cout << getDiagnosis(1, buf) << std::endl;  \*log.stream << getDiagnosis(1, buf) << std::endl;  std::cout << getDiagnosis(2, buf) << std::endl;  \*log.stream << getDiagnosis(2, buf) << std::endl;  throw ERROR\_THROW(605);  break;  }  case NS\_NORULECHAIN: MFST\_TRACE4("------>NS\_NORULECHAIN") break;  case NS\_ERROR: MFST\_TRACE4("------>NS\_ERROR") break;  case SURPRISE: MFST\_TRACE4("------>NS\_SURPRISE") break;  }  return rc;  };  char\* Mfst::getCSt(char\* buf)  {  short p;  for (int k = (signed)st.size() - 1; k >= 0; --k)  {  p = st.c[k];  buf[st.size() - 1 - k] = GRB::Rule::Chain::alphabet\_to\_char(p);  }  buf[st.size()] = '\0';  return buf;  }  char\* Mfst::getCLenta(char\* buf, short pos, short n)  {  short i, k = (pos + n < lenta\_size) ? pos + n : lenta\_size;  for (i = pos; i < k; i++) buf[i - pos] = GRB::Rule::Chain::alphabet\_to\_char(lenta[i]);  buf[i - pos] = 0x00;  return buf;  }  char\* Mfst::getDiagnosis(short n, char\* buf)  {  char\* rc = new char[200] {};  int errid = 0;  int lpos = -1;  if (n < MFST\_DIAGN\_NUMBER && (lpos = diagnosis[n].lenta\_position) >= 0)  {  errid = grebach.getRule(diagnosis[n].nrule).iderror;  Error::ERROR err = Error::geterror(errid);  sprintf\_s(buf, MFST\_DIAGN\_MAXSIZE, "%d: строка %d,%s", err.id, lex.table[lpos].sn, err.message);  rc = buf;  }  return rc;  }  void Mfst::printrules()  {  MfstState state;  GRB::Rule rule;  for (unsigned short i = 0; i < storestate.size(); i++)  {  state = storestate.c[i];  rule = grebach.getRule(state.nrule);  MFST\_TRACE7  };  };  bool Mfst::savededucation()  {  MfstState state;  GRB::Rule rule;  deducation.nrules = new short[deducation.size = storestate.size()];  deducation.nrulechains = new short[deducation.size];  for (unsigned short i = 0; i < storestate.size(); i++)  {  state = storestate.c[i];  deducation.nrules[i] = state.nrule;  deducation.nrulechains[i] = state.nrulechain;  }  return true;} |

**Листинг 2 – Функции синтаксического анализатора**

|  |
| --- |
| **Шаг : Правило Входная лента Стек**  **0 : S->fti(W){Niq}S fti(ti){tili$ii+i$iq}m{li S!**  **0 : SAVESTATE: 1**  **0 : fti(ti){tili$ii+i$iq}m{li fti(W){Niq}S!**  **1 : ti(ti){tili$ii+i$iq}m{lit ti(W){Niq}S!**  **2 : i(ti){tili$ii+i$iq}m{liti i(W){Niq}S!**  **3 : (ti){tili$ii+i$iq}m{litil (W){Niq}S!**  **4 : ti){tili$ii+i$iq}m{litili W){Niq}S!**  **5 : W->ti ti){tili$ii+i$iq}m{litili W){Niq}S!**  **5 : SAVESTATE: 2**  **5 : ti){tili$ii+i$iq}m{litili ti){Niq}S!**  **6 : i){tili$ii+i$iq}m{litilii i){Niq}S!**  **7 : ){tili$ii+i$iq}m{litilii$ ){Niq}S!**  **8 : {tili$ii+i$iq}m{litilii$l {Niq}S!**  **9 : tili$ii+i$iq}m{litilii$li Niq}S!**  **10 : N->ti tili$ii+i$iq}m{litilii$li Niq}S!**  **10 : SAVESTATE: 3**  **10 : tili$ii+i$iq}m{litilii$li tiiq}S!**  **11 : ili$ii+i$iq}m{litilii$lii iiq}S!**  **12 : li$ii+i$iq}m{litilii$liid iq}S!**  **13 : 2**  **13 : RESSTATE**  **…**  **2351: N->\*N \*++i} N}!**  **2351: SAVESTATE: 108**  **2351: \*++i} \*N}!**  **2352: ++i} N}!**  **2353: N->+ ++i} N}!**  **2353: SAVESTATE: 109**  **2353: ++i} +}!**  **2354: +i} }!**  **2355: 2**  **2355: RESSTATE**  **2355: ++i} N}!**  **2356: N->+N ++i} N}!**  **2356: SAVESTATE: 109**  **2356: ++i} +N}!**  **2357: +i} N}!**  **2358: N->+ +i} N}!**  **2358: SAVESTATE: 110**  **2358: +i} +}!**  **2359: i} }!**  **2360: 2**  **2360: RESSTATE**  **2360: +i} N}!**  **2361: N->+N +i} N}!**  **2361: SAVESTATE: 110**  **2361: +i} +N}!**  **2362: i} N}!**  **2363: N->i i} N}!**  **2363: SAVESTATE: 111**  **2363: i} i}!**  **2364: } }!**  **2365: !**  **2366: 6**  **2367: ------>LENTA\_END**  **0 : S->fti(W){Niq}S**  **4 : W->ti**  **8 : N->tiN**  **10 : N->lN**  **11 : N->i$N**  **13 : N->iN**  **14 : N->iN**  **15 : N->+N**  **16 : N->i$**  **21 : S->m{N}**  **23 : N->lN**  **24 : N->iN**  **25 : N->tiN**  **27 : N->lN**  **28 : N->iN**  **29 : N->i$N**  **31 : N->lN**  **32 : N->iN**  **33 : N->iN**  **34 : N->dN**  **35 : N->xN**  **36 : N->i$N**  **38 : N->lN**  **39 : N->iN**  **40 : N->i$N**  **42 : N->z(N){N}N**  **44 : N->lN**  **45 : N->iN**  **46 : N-><**  **49 : N->lN**  **50 : N->iN**  **51 : N->iN**  **52 : N->x**  **54 : N->atil@N**  **59 : N->h(N){N}N**  **61 : N->lN**  **62 : N->iN**  **63 : N-><**  **66 : N->iN**  **67 : N->lN**  **68 : N->-N**  **69 : N->i$N**  **71 : N->iN**  **72 : N->xN**  **73 : N->z(N){N}**  **75 : N->iN**  **76 : N->lN**  **77 : N-><**  **80 : N->iN**  **81 : N->ii@$**  **87 : N->tiN**  **89 : N->lN**  **90 : N->i$N**  **92 : N->lN**  **93 : N->iN**  **94 : N->h(N){N}N**  **96 : N->iN**  **97 : N->lN**  **98 : N-><**  **101 : N->ii@N**  **104 : N->xN**  **105 : N->iN**  **106 : N->lN**  **107 : N->+N**  **108 : N->i$N**  **110 : N->z(N){N}**  **112 : N->l**  **115 : N->lN**  **116 : N->i**  **119 : N->til@N**  **123 : N->lN**  **124 : N->iN**  **125 : N->inN**  **127 : N->lN**  **128 : N->iN**  **129 : N->iN**  **130 : N->iN**  **131 : N->xN**  **132 : N->lN**  **133 : N->iN**  **134 : N->lN**  **135 : N->iN**  **136 : N->tiN**  **138 : N->inN**  **140 : N->lN**  **141 : N->iN**  **142 : N->tiN**  **144 : N->inN**  **146 : N->iN**  **147 : N->iN**  **148 : N->+N**  **149 : N->lN**  **150 : N->iN**  **151 : N->iN**  **152 : N->lN**  **153 : N->iN**  **154 : N->iN**  **155 : N->iN**  **156 : N->lN**  **157 : N->/N**  **158 : N->lN**  **159 : N->iN**  **160 : N->lN**  **161 : N->lN**  **162 : N->lN**  **163 : N->-N**  **164 : N->\*N**  **165 : N->\*N**  **166 : N->+N**  **167 : N->+N**  **168 : N->i**  **168 : N->i** |

**Листинг 3 – Протокол работы синтаксического анализатора**

****Приложение Д****

**Программная реализация обработки выражений**

|  |
| --- |
| string infixToPostfix(vector<string> tokens)  {  stack<string> stack;  string currentToken;  std::string result;  for (int i = 0; i < tokens.size(); i++)  {  currentToken = tokens[i];  if (find(operators.begin(), operators.end(), currentToken)!=operators.end())  {  if (currentToken == "(")  {  stack.push(currentToken);  continue;  }  if (currentToken == ")")  {  while (stack.top() != "(")  {  result += stack.top();  result += " ";  stack.pop();  if(stack.empty())  throw ERROR\_THROW\_IN(2,str,col);  }  stack.pop();  continue;  }  if (stack.empty() || stack.top() == "(")  {    stack.push(currentToken);  }  else if (getPriority(currentToken) < getPriority(stack.top()))  {  while ((getPriority(currentToken) < getPriority(stack.top())))  {  result += stack.top();  result += " ";  stack.pop();  if (stack.empty())  break;  }  stack.push(currentToken);  }  else  {  stack.push(currentToken);  }    }  else {if (currentToken.find('@') != std::string::npos)  {  stack.push(currentToken);  i++;  //пропускаем открывающеюся скобку  i++;  currentToken = tokens[i];  while(currentToken!=")")  {  stack.push(currentToken);  i++;  currentToken = tokens[i];  }  while (stack.top().find('@') == std::string::npos)  {  result += stack.top();  result += " ";  stack.pop();  }  stack.top().pop\_back();  stack.top().pop\_back();  result += stack.top();  result += " ";  stack.pop();  //убрать оставшиеся скобку  currentToken = "";  }  result += currentToken;  result += " ";  }  }  while (!stack.empty())  {  if (find(operators.begin(), operators.end(), stack.top()) == operators.end() || stack.top()=="("||stack.top()==")")  {  throw ERROR\_THROW\_IN(2,str,col);  }  result += stack.top();  result += " ";  stack.pop();  }return result;} |

**Листинг 1 – Программная реализация обработки выражений**

****Приложение Е****

**Результат генерации и функия сложения**

|  |  |
| --- | --- |
| PLUS PROC type1 : ptr byte, var1 : ptr byte, type2 : ptr byte, var2 : ptr byte  mov eax, type1  mov al, [eax]; Тип первой переменной  mov ebx, type2  mov bl, [ebx]; Тип второй переменной  cmp al, 2  je var1Int  cmp al,1  jne callErrorType  mov eax, 0  mov ecx, var1  movsx eax, byte ptr [ecx]  jmp second\_var  var1Int :  mov eax, 0  mov ecx, var1  mov eax, [ecx]  second\_var :  cmp bl, 2  je var2Int  cmp bl,1  jne callErrorType  mov ebx, 0  mov ecx, var2  movsx ebx, byte ptr [ecx]  jmp done  var2Int :  mov ebx, 0  mov ecx, var2  mov ebx, [ecx]  done :  add eax, ebx  cmp eax, 127; Сравнение с 127 | jg greater; Переход, если значение больше 127  cmp eax, -128; Сравнение с –128 jl less; Переход, если значение меньше - 128  ; Значение в EAX находится в диапазоне от - 128 до 127  mov ebx, 1; Сохраняем 1 в EBX  mov ecx, ptrBuffer  mov[byte ptr[ecx]], bl; Переходим к завершению  mov[byte ptr[ecx + 1]], al  add ptrBuffer,2  mov eax, ecx  inc ecx  mov ebx, ecx  ret  greater :  ; Значение в EAX больше 127  mov ebx, 2; Сохраняем 2 в EBX  mov ecx, ptrBuffer  mov[byte ptr[ecx]], bl; Переходим к завершению  mov[dword ptr[ecx + 1]], eax  add ptrBuffer,5  mov eax, ecx  inc ecx  mov ebx, ecx  ret  less :  ; Значение в EAX меньше - 128  mov ebx, 2; Сохраняем 2 в EBX  mov ecx, ptrBuffer  mov[byte ptr[ecx]], bl; Переходим к завершению  mov[dword ptr[ecx + 1]], eax  add ptrBuffer,5  mov eax, ecx  inc ecx  mov ebx, ecx  ret  PLUS ENDP |

**Листинг 1 – Функция сложения**

|  |  |
| --- | --- |
| .586  .MODEL FLAT, STDCALL  includelib kernel32.lib  includelib user32.lib  includelib libucrt.lib  includelib ../Debug/write.lib  includelib ../Debug/callErrors.lib  ExitProcess PROTO: DWORD  MessageBoxA PROTO : DWORD, : DWORD, : DWORD, : DWORD  SetConsoleTitleA PROTO :DWORD  EXTERN writeString: PROC  EXTERN writeLineString: PROC  EXTERN readCon:PROC  EXTERN waitPause: PROC  EXTERN ErrorType: PROC  EXTERN ErrorStackEmpty: PROC  EXTERN ErrorStackOverflow: PROC  EXTERN ErrorIncorInd:PROC  includelib ../Debug/Math.lib  EXTERN factor\_fun: PROC  EXTERN power\_fun: PROC  includelib ../Debug/String.lib  EXTERN length\_fun: PROC  EXTERN compare\_fun: PROC  EXTERN print\_fun:PROC  EXTERN compS\_fun: PROC  includelib ../Debug/Math.lib  EXTERN factor\_fun: PROC  EXTERN power\_fun: PROC  includelib ../Debug/String.lib  EXTERN length\_fun: PROC  EXTERN compare\_fun: PROC  EXTERN print\_fun:PROC  EXTERN compS\_fun: PROC  .STACK 4096  .DATA  MB\_OK EQU 0  message db 'Произошла ошибка во время выполнения!'  caption db 'Ошибка в строке:'  MB\_ICONERROR EQU 16  current\_string DWORD 0  buffer db 4096 dup(0)  ptrBuffer dword OFFSET buffer  consoletitle db 'in',0  ;VARS  stklength db 1, 0,4 dup(0)  stkcompare db 1, 0,4 dup(0)  stkcompS db 1, 0,4 dup(0)  stkprint db 1, 0,4 dup(0)  stkpower db 1, 0,4 dup(0)  stkfactor db 1, 0,4 dup(0)  stkexample db 1, 0,4 dup(0)  b\_example db 2,4 dup(0)  stkmain db 1, 0,4 dup(0)  t\_main db 2,4 dup(0)  arr\_main db 2,6\*4 dup(0)  ind\_main db 1,0  str\_main db 7,10 dup(-1),0  var1\_main db 2,4 dup(0)  var2\_main db 2,4 dup(0)  .CONST  L0 db 1,5  L2 db 7,'hello world',0  L3 db 1,3  L4 db 1,2  L5 db 1,11  L6 db 1,15  L8 db 7,'t=',0  L9 db 1,6  L10 db 1,0  L11 db 1,1  L13 db 7,'Вывод массива:',0  true db 3,1  L16 db 7,'hello',0  L17 db 1,10  L19 db 7,'Введите строку(не больше 10 может поместиться)',0  L21 db 7,'Длина строки',0  L23 db 7,'Сумма чисел',0  L25 db 7,'Первая переменаня',0  L27 db 7,'Вторая переменная',0  L29 db 7,'Результат',0  L31 db 7,'Результат выражения var1+var2/3+example(4)\*2\*(10-5)',0  L32 db 1,4  false db 3,0  example\_fun PROC a\_example\_type: ptr byte,a\_example: ptr byte  START:  mov eax, a\_example\_type  mov al, [eax]  cmp al, 2  je go\_next  cmp al, 1  jne CallErrorType  go\_next:  mov dword ptr stkexample+2, esp  mov current\_string,4  mov current\_string,5  mov current\_string,6  add stkexample +1,1  cmp stkexample+1,32  jg callErrorOver  push offset L0+1  push offset L0  add stkexample +1,1  cmp stkexample+1,32  jg callErrorOver  push offset b\_example+1  push offset b\_example  sub stkexample +1,2  cmp stkexample+1, 0  jl callErrorEmpty  call Assign  mov current\_string,7  add stkexample +1,1  cmp stkexample+1,32  jg callErrorOver  push offset b\_example+1  push offset b\_example  add stkexample +1,1  cmp stkexample+1,32  jg callErrorOver  push a\_example  push a\_example\_type  sub stkexample +1,2  cmp stkexample+1, 0  jl callErrorEmpty  call PLUS  push ebx  push eax  add stkexample +1,1  cmp stkexample+1,32  jg callErrorOver  mov current\_string,8  add stkexample +1,1  cmp stkexample+1,32  jg callErrorOver  push offset b\_example+1  push offset b\_example  sub stkexample +1,2  cmp stkexample+1, 0  jl callErrorEmpty  call Assign  mov current\_string,9  add stkexample +1,1  cmp stkexample+1,32  jg callErrorOver  push offset b\_example+1  push offset b\_example  pop eax  pop ebx  ret  mov current\_string,10  mov esp,dword ptr stkexample+2  ret  example\_fun ENDP  mov current\_string,12  main PROC  START :  push OFFSET consoletitle  call SetConsoleTitleA  mov current\_string,12  mov dword ptr stkmain+2, esp  mov current\_string,13  mov current\_string,14  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L2+1  push offset L2  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call print\_fun  pop edx  pop edx  mov current\_string,15  mov current\_string,16  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L3+1  push offset L3  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call factor\_fun  pop edx  pop edx  add eax, 1  push eax  sub eax,1  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  mov current\_string,17  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset t\_main+1  push offset t\_main  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call Assign  mov current\_string,18  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L4+1  push offset L4  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset t\_main+1  push offset t\_main  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call power\_fun  pop edx  pop edx  pop edx  pop edx  add eax, 1  push eax  sub eax,1  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  mov current\_string,19  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  pop eax  pop ebx  push ebx  push eax  push ebx  push eax  mov eax,0  mov ebx,0  cmp stkmain+1, 1  jl callErrorEmpty  call writeLineString  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  pop eax  pop eax  mov eax,0  mov current\_string,20  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset t\_main+1  push offset t\_main  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call Assign  mov current\_string,21  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L5+1  push offset L5  mov current\_string,22  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call example\_fun  add eax, 1  push eax  sub eax,1  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset t\_main+1  push offset t\_main  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call Assign  mov current\_string,23  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L6+1  push offset L6  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset t\_main+1  push offset t\_main  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call LESSFUN  push ebx  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  pop eax  pop eax  mov al, [eax]  cmp al,[byte ptr true+1]  jne goIf1  mov current\_string,25  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L8+1  push offset L8  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call print\_fun  pop edx  pop edx  mov current\_string,26  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset t\_main+1  push offset t\_main  cmp stkmain+1, 1  jl callErrorEmpty  call writeLineString  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  pop eax  pop eax  mov eax,0  mov current\_string,27  goIf1:  mov current\_string,28  mov current\_string,29  while2:  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L10+1  push offset L10  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset t\_main+1  push offset t\_main  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call LESSFUN  push ebx  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  pop eax  pop eax  mov al, [eax]  cmp al,[byte ptr true+1]  jne endWhile2  mov current\_string,31  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset t\_main+1  push offset t\_main  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L11+1  push offset L11  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call MINUS  push ebx  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  mov current\_string,32  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset t\_main+1  push offset t\_main  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call Assign  mov current\_string,33  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset t\_main+1  push offset t\_main  cmp stkmain+1, 1  jl callErrorEmpty  call writeLineString  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  pop eax  pop eax  mov eax,0  mov current\_string,34  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset t\_main+1  push offset t\_main  add stkmain+1, 1 | cmp stkmain+1, 32  jg callErrorOver  push offset L9+1  push offset L9  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call LESSFUN  push ebx  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  pop eax  pop eax  mov al, [eax]  cmp al,[byte ptr true+1]  jg callErrorOver  push offset ind\_main+1  push offset ind\_main  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L9+1  push offset L9  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call LESSFUN  push ebx  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  pop eax  pop eax  mov al, [eax]  cmp al,[byte ptr true+1]  jne endWhile4  mov current\_string,44  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  mov eax, OFFSET arr\_main  add eax,1  mov ebx, OFFSET ind\_main  add ebx,1  mov edx,0  mov dl, [ebx]  shl edx, 2  add eax, edx  push eax  shr edx,2  cmp edx, 6  je callErrorIncorInd  jg callErrorIncorInd  push OFFSET arr\_main  cmp stkmain+1, 1  jl callErrorEmpty  call writeLineString  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  pop eax  pop eax  mov eax,0  mov current\_string,45  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset ind\_main+1  push offset ind\_main  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L11+1  push offset L11  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call PLUS  push ebx  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  mov current\_string,46  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset ind\_main+1  push offset ind\_main  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call Assign  mov current\_string,47  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset true+1  push offset true  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  pop eax  pop eax  mov al, [eax]  cmp al,[byte ptr true+1]  jne goIf5  mov current\_string,49  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L16+1  push offset L16  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call print\_fun  pop edx  pop edx  mov current\_string,50  goIf5:  mov current\_string,51  jmp while4  endWhile4:  mov current\_string,52  mov current\_string,53  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L19+1  push offset L19  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call print\_fun  pop edx  pop edx  mov current\_string,54  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset str\_main+1  push offset str\_main  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  INVOKE int\_to\_char, OFFSET caption, current\_string  push OFFSET caption  call readCon  pop eax  pop eax  pop eax  mov eax,0  mov current\_string,55  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L21+1  push offset L21  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call print\_fun  pop edx  pop edx  mov current\_string,56  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset str\_main+1  push offset str\_main  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call length\_fun  pop edx  pop edx  add eax, 1  push eax  sub eax,1  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  cmp stkmain+1, 1  jl callErrorEmpty  call writeLineString  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  pop eax  pop eax  mov eax,0  mov current\_string,57  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L23+1  push offset L23  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call print\_fun  pop edx  pop edx  mov current\_string,58  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L25+1  push offset L25  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call print\_fun  pop edx  pop edx  mov current\_string,59  mov current\_string,60  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset var1\_main+1  push offset var1\_main  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  INVOKE int\_to\_char, OFFSET caption, current\_string  push OFFSET caption  call readCon  pop eax  pop eax  pop eax  mov eax,0  mov current\_string,61  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L27+1  push offset L27  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call print\_fun  pop edx  pop edx  mov current\_string,62  mov current\_string,63  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset var2\_main+1  push offset var2\_main  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  INVOKE int\_to\_char, OFFSET caption, current\_string  push OFFSET caption  call readCon  pop eax  pop eax  pop eax  mov eax,0  mov current\_string,64  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset var1\_main+1  push offset var1\_main  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset var2\_main+1  push offset var2\_main  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call PLUS  push ebx  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  mov current\_string,65  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L29+1  push offset L29  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call print\_fun  pop edx  pop edx  mov current\_string,66  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call print\_fun  pop edx  pop edx  mov current\_string,67  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L31+1  push offset L31  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call print\_fun  pop edx  pop edx  mov current\_string,68  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset var1\_main+1  push offset var1\_main  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset var2\_main+1  push offset var2\_main  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L3+1  push offset L3  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call DIVIDE  push ebx  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L32+1  push offset L32  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call example\_fun  add eax, 1  push eax  sub eax,1  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L4+1  push offset L4  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L17+1  push offset L17  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  push offset L0+1  push offset L0  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call MINUS  push ebx  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call MULT  push ebx  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call MULT  push ebx  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call PLUS  push ebx  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  sub stkmain+1, 2  cmp stkmain+1,0  jl callErrorEmpty  call PLUS  push ebx  push eax  add stkmain+1, 1  cmp stkmain+1, 32  jg callErrorOver  mov current\_string,69  sub stkmain+1, 1  cmp stkmain+1,0  jl callErrorEmpty  call print\_fun  pop edx  pop edx  mov current\_string,70  call waitPause  mov esp,dword ptr stkmain+2  push - 1  call ExitProcess  main ENDP  end main |

**Листинг 2 – Результат генерации кода**

**Приложение Ж**

**Задача «Баланс скобок»**

|  |
| --- |
| **#include string.lib**  **main**  **{**  **'Введите строку для проверки(макс длина 20 символов)' print**  **string str[20]**  **str read**  **int len**  **int i**  **0 i $**  **str length**  **len $**  **while(i len <)**  **{**  **if('{' str[i] compS)**  **{**  **'{'**  **}**  **if('}' str[i] compS)**  **{**  **if(depth 1 - 0 =)**  **{**  **'неправильная строка' print**  **0 return**  **}**  **drop**  **}**  **i 1 +**  **i $**    **}**  **if(depth 1 - 0 >)**  **{**  **'неправильная строка2' print**  **0 return**  **}**  **}** |