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 BVA-2023»

Выполнил студент Бычковская Виктория Александровна

(Ф.И.О.)

Руководитель проекта асс. Мущук Артур Николаевич

(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 асс. Мущук Артур Николаевич

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

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

Нормоконтролер асс. Мущук Артур Николаевич

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

Курсовой проект защищен с оценкой

Минск 2023

Содержание

[Введение 5](#_Toc153810632)

[Глава 1 Спецификация языка программирования 6](#_Toc153810633)

[1.1 Характеристика языка программирования 6](#_Toc153810634)

[1.2 Определение алфавита языка программирования 6](#_Toc153810635)

[1.3 Применяемые сепараторы 7](#_Toc153810636)

[1.4 Применяемые кодировки 7](#_Toc153810637)

[1.5 Типы данных 7](#_Toc153810638)

[1.6 Преобразование типов данных 8](#_Toc153810639)

[1.7 Идентификаторы 8](#_Toc153810640)

[1.8 Литералы 8](#_Toc153810641)

[1.9 Объявления данных 9](#_Toc153810642)

[1.10 Инициализация данных 9](#_Toc153810643)

[1.11 Инструкции языка 9](#_Toc153810644)

[1.12 Операции языка 10](#_Toc153810645)

[1.13 Выражения и их вычисления 11](#_Toc153810646)

[1.14 Конструкции языка 11](#_Toc153810647)

[1.15 Область видимости идентификаторов 12](#_Toc153810648)

[1.16 Семантические проверки 12](#_Toc153810649)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc153810650)

[1.18 Стандартная библиотека и её состав 12](#_Toc153810651)

[1.19 Ввод и вывод данных 13](#_Toc153810652)

[1.20 Точка входа 13](#_Toc153810653)

[1.21 Препроцессор 13](#_Toc153810654)

[1.22 Соглашения о вызовах 13](#_Toc153810655)

[1.23 Объектный код 13](#_Toc153810656)

[1.24 Классификация сообщений транслятора 14](#_Toc153810657)

[1.25 Контрольный пример 14](#_Toc153810658)

[Глава 2 Структура транслятора 15](#_Toc153810659)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc153810660)

[2.2 Перечень входных параметров транслятора 16](#_Toc153810661)

[2.3 Протоколы, формируемые транслятором и их содержимое 16](#_Toc153810662)

[Глава 3 Разработка лексического анализатора 17](#_Toc153810663)

[3.1 Структура лексического анализатора 17](#_Toc153810664)

[3.2 Входные и выходные данные лексического анализатор 17](#_Toc153810665)

[3.3 Параметры лексического анализатора 17](#_Toc153810666)

[3.4 Алгоритм лексического анализа 18](#_Toc153810667)

[3.5 Контроль входных символов 18](#_Toc153810668)

[3.6 Удаление избыточных символов 18](#_Toc153810669)

[3.7 Перечень ключевых слов 19](#_Toc153810670)

[3.8 Основные структуры данных 20](#_Toc153810671)

[3.9 Структура и перечень сообщений лексического анализатора 20](#_Toc153810672)

[3.10 Принцип обработки ошибок 20](#_Toc153810673)

[3.11 Контрольный пример 20](#_Toc153810674)

[Глава 4 Разработка синтаксического анализатора 21](#_Toc153810675)

[4.1 Структура синтаксического анализатора 21](#_Toc153810676)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 21](#_Toc153810677)

[4.3 Построение конечного магазинного автомата 23](#_Toc153810678)

[4.4 Основные структуры данных 24](#_Toc153810679)

[4.5 Описание алгоритма синтаксического разбора 24](#_Toc153810680)

[4.6 Параметры синтаксического анализатора и режимы его работы 25](#_Toc153810681)

[4.7 Структура и перечень сообщений синтаксического анализатора 25](#_Toc153810682)

[4.8 Принцип обработки ошибок 25](#_Toc153810683)

[4.9 Контрольный пример 25](#_Toc153810684)

[Глава 5 Разработка семантического анализатора 26](#_Toc153810685)

[5.1 Структура семантического анализатора 26](#_Toc153810686)

[5.2 Функции семантического анализатора 26](#_Toc153810687)

[5.3 Структура и перечень сообщений семантического анализатора 26](#_Toc153810688)

[5.4 Принцип обработки ошибок 27](#_Toc153810689)

[5.5 Контрольный пример 27](#_Toc153810690)

[Глава 6 Преобразование выражений 28](#_Toc153810691)

[6.1 Выражения, допускаемые языком 28](#_Toc153810692)

[6.2 Польская запись и принцип её построения 28](#_Toc153810693)

[6.3 Программная реализация обработки выражений 29](#_Toc153810694)

[6.4 Контрольный пример 29](#_Toc153810695)

[Глава 7 Генерация кода 30](#_Toc153810696)

[7.1 Структура генератора кода 30](#_Toc153810697)

[7.2 Представление типов данных в оперативной памяти 30](#_Toc153810698)

[7.3 Статическая библиотека 30](#_Toc153810699)

[7.4 Особенности алгоритма генерации кода 31](#_Toc153810700)

[7.5 Параметры, управляющие генерацией кода 33](#_Toc153810701)

[7.6 Контрольный пример 33](#_Toc153810702)

[Глава 8 Тестирование транслятора 34](#_Toc153810703)

[8.1 Общие положения 34](#_Toc153810704)

[8.2 Результаты тестирования 34](#_Toc153810705)

[Заключение 36](#_Toc153810706)

[Список используемых источников 37](#_Toc153810707)

[ПРИЛОЖЕНИЕ А 38](#_Toc153810708)

[ПРИЛОЖЕНИЕ Б 47](#_Toc153810709)

[ПРИЛОЖЕНИЕ В 49](#_Toc153810710)

[ПРИЛОЖЕНИЕ Г 50](#_Toc153810711)

[ПРИЛОЖЕНИЕ Д 53](#_Toc153810712)

[ПРИЛОЖЕНИЕ Е 56](#_Toc153810713)

[ПРИЛОЖЕНИЕ Ж 61](#_Toc153810714)

# **Введение**

Целью курсового проекта является разработка собственного языка программирования и компилятора для него. Язык называется BVA-2023. Написание компилятора будет осуществляться на языке C++.

Компилятор – это программа, задачей которого является перевод программы, написанной на одном из языков программирования в программу на язык ассемблера.

Компиляция состоит из двух частей: анализа и генерации. Анализ – это разбиение исходной программы на составные части и создание ее промежуточного представления. Генерация – конструирование требуемой целевой программы из промежуточного представления. В данном курсовом проекте исходный код транслируется на язык ассемблера.

Для выполнения курсового проекта были поставлены следующие задачи:

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

# **Глава 1 Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

Язык программирования BVA-2023 классифицируется как процедурный, универсальный, строго типизированный, компилируемый.

**1.2 Определение алфавита языка программирования**

В основе алфавита BVA-2023 лежит таблица символов Windows-1251, которая представлена на рисунке 1.1.



Рисунок 1.1 – Таблица кодировки Windows-1251

В языке BVA-2023 могут использоваться символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, спецсимволы, а также непечатные символы пробела, табуляции и перевода строки. Русские символы разрешены только в строковых литералах.

**1.3 Применяемые сепараторы**

Сепараторы необходимы для разделения операций языка. Сепараторы, используемые в языке программирования BVA-2023, приведены в таблице 1.1.

Таблица 1.1 - Применяемые сепараторы

|  |  |  |
| --- | --- | --- |
| Сепаратор | Название | Область применения |
| ‘ ‘ | Пробел | Допускается везде, кроме идентификаторов и ключевых слов |
| ; | Точка с запятой | Разделение конструкций |
| {…} | Фигурные скобки | Заключение программного блока |
| […] | Квадратные кавычки | Блок кода в условных конструкциях |
| (…) | Круглые скобки | Приоритет операций, параметры функции |
| ‘…’ | Одинарные кавычки | Допускается везде, кроме идентификаторов и ключевых слов |
| = | Знак «равно» | Присваивание значения |
| , | Запятая | Разделение параметров |
| +  -  \*  |  &  ~ | Знаки «плюс», «минус», «астерикс», «прямая черта», «амперсанд», «тильда» | Выражения |

**1.4 Применяемые кодировки**

При трансляции исходного кода применяется кодировка Windows-1251. Описание кодировки представлено в пункте 1.2.

**1.5 Типы данных**

Допускается использование фундаментальных типов данных. В языке BVA- 2023 реализованы 2 типа данных: целочисленный и символьный. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка BVA- 2023

|  |  |
| --- | --- |
| Тип данных | Описание |
| Целочисленный тип данных (int) | Фундаментальный тип данных. Предусмотрен для объявления целочисленных данных (4 байта). Инициализация по умолчанию: значение 0. Максимально допустимое значение 231-1. Минимально допустимым является -231-1. Для получения отрицательного числа нужно ввести выражение типа: 0 – i; |
| Символьный тип данных (symb) | Фундаментальный тип данных. Используется для работы с символами, который в памяти занимает 1 байт. Инициализация по умолчанию: символ конца строки «\0». |

Пользовательские типы данных не поддерживаются.

**1.6 Преобразование типов данных**

В языке программирования BVA-2023 преобразование типов данных не поддерживается. Все типы данных определенны однозначно и не могут быть преобразованы в другие.

**1.7 Идентификаторы**

Для именования функций, параметров и переменных используются идентификаторы. Не предусмотрены зарезервированные идентификаторы. Имя идентификаторов не должно совпадать с ключевыми словами языка и с именами функций стандартной библиотеки. В имени идентификатора допускаются только символы латинского алфавита нижнего регистра [a … z]. Максимальная длина имени идентификатора – 15 символов. Максимальная длина имени идентификатора функции – 15 символов.

**1.8 Литералы**

В языке BVA-2023 предусмотрены 2 вида литералов: целочисленные и строковые. Краткое описание литералов приведено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Литерал | Пояснение |
| Целочисленный | Целочисленные неотрицательные литералы, по умолчанию инициализируются 0. Максимально допустимое значение 231-1. Минимально допустимым является -231-1. При выходе за пределы допустимости выводится соответствующая ошибка. |
| Строковый | Используются символы кодировки ASCII. Максимальный размер строки – 255. |

Строковый литерал используется только внутри оператора вывода write. Литералы являются константами и при генерации кода, заносятся в раздел .const.

**1.9 Объявления данных**

В языке BVA-2023 требуется обязательное объявление переменной перед её инициализацией и последующим использованием. Все переменные должны находиться внутри программного блока. Имеется возможность объявления одинаковых переменных в разных блоках, т. к. переменные, объявленные в одной функции, недоступны в другой. Каждая переменная получает префикс – название функции, в которой она объявлена. Недопустимо объявление глобальных переменных.

**1.10 Инициализация данных**

При объявлении переменной не допускается инициализация данных. Краткое описание способов инициализации переменных языка BVA-2023 представлено в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

|  |  |  |
| --- | --- | --- |
| Конструкция | Описание | Пример |
| set <тип данных> <идентификатор>; | Автоматическая инициализация: переменные типа int инициализируются нулём, переменные типа symb – пустым символом. | set int sum;  set symb abc; |
| <идентификатор> = <значение>; | Присваивание переменной значения. Целочисленные значения могут представляться как в десятичном, так и шестнадцатеричном виде. | sum = 15;  sum = 0x1F;  abc = ‘a’; |

Соответствие типов проверяется с помощью семантического анализа.

**1.11 Инструкции языка**

Все возможные инструкции языка программирования BVA-2023 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования BVA-2023

|  |  |
| --- | --- |
| Инструкция | Запись на языке BVA-2023 |
| Объявление переменной | set <тип данных> <идентификатор>; |
| Объявление функции | set <тип данных> func <идентификатор> (<тип данных> <идентификатор>, …) {<блок кода>}; |
| Присваивание | <идентификатор> = <значение>/<идентификатор>; |
| Блок инструкций | main  {  …  } |
| Возврат из подпрограммы | ret <идентификатор> / <литерал>; |
| Условная инструкция | if(<условие>)[<блок кода>]; |
| Вывод данных | write <идентификатор> / <литерал>; |

Условный оператор и функции входа в программу не требуют закрывающую «;».

**1.12 Операции языка**

Язык программирования BVA-2023 может выполнять операции, представленные в таблице 1.6.

Таблица 1.6 – Операции языка программирования BVA-2023

|  |  |  |
| --- | --- | --- |
| Операция | Примечание | Пример |
| = | Присваивание | sum = 15;  symbol = ‘S’; |
| & | Побитовое И | a & b; |
| | | Побитовое ИЛИ | a | b; |
| ~ | Отрицание | a ~ 1 |
| ( | Приоритет операций | (a + b) \* c; |
| ) |

Окончание таблицы 1.6

|  |  |  |
| --- | --- | --- |
| Операция | Примечание | Пример |
| + | Суммирование | a + b; |
| - | Вычитание | a – b; |
| \* | Умножение | a \* b; |

Отрицательные числа задаются с помощью операции «-» (вычитанием большего числа из меньшего).

**1.13 Выражения и их вычисления**

Всякое выражение составляется согласно следующим правилам:

* Допускается использовать скобки для смены приоритета операций;
* Выражение записывается в строку без переносов;
* Использование двух подряд идущих операторов не допускается;
* Допускается использовать в выражении вызов функции, вычисляющей и возвращающей целочисленное значение.

Перед генерацией кода каждое выражение приводится к записи в польской записи для удобства дальнейшего вычисления выражения на языке ассемблера [1].

**1.14 Конструкции языка**

Ключевые программные конструкции языка программирования BVA-2023 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка BVA-2023

|  |  |
| --- | --- |
| Конструкция | Запись на языке BVA-2023 |
| Главная функция (точка входа) | main  {  …  ret <идентификатор> / <литерал>;  }; |
| Функция | <тип> func <идентификатор> (<тип> <идентификатор>, …)  {  …  ret <идентификатор> / <литерал>;  }; |
| Условная конструкция | if(<условие>)  [  …  ]  else(<условие>)  [  …  ] |

В данной таблице представлены различные конструкции программирования и их реализация. Главная функция "main" является основной точкой входа в программу. Внешние функции определяются с помощью ключевого слова "func" со списком параметров. Условные конструкции позволяют выполнять различные кодовые блоки в зависимости от условий истинности или ложности.

**1.15 Область видимости идентификаторов**

В языке BVA-2023 все переменные являются локальными, т.е. имеют функциональную область видимости. Они обязаны находится внутри программного блока функций. Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

**1.16 Семантические проверки**

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

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы не должны повторно объявляться в пределах одной функции. |
| 2 | Тип возвращаемого значения должен совпадать с типом функции при её объявлении или подключении |
| 3 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении или подключении |
| 4 | В функцию должно быть передано то число параметров, сколько ожидается |
| 5 | Тип данных результата выражения должен совпадать с типом данных идентификатора, которому оно присваивается |
| 6 | Проверка условного оператора |

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

**1.17 Распределение оперативной памяти на этапе выполнения**

Все переменные размещаются в куче.

**1.18 Стандартная библиотека и её состав**

Для использования функций стандартной библиотеки BVA-2023, нужно явно подключить необходимую функцию с помощью ключевого слова extern, далее работа с ними производится как с пользовательскими функциями. Функции стандартной библиотеки с описанием представлены в таблице 1.9.

Таблица 1.9 – Состав стандартной библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| long stdcall step(long int1, long int2) | Возведение int1 в степень int2 |
| long stdcall iabs(long aint) | Вычисляет абсолютное значение |
| void stdcall ConsoleWriteInt(long number) | Выводит значение на консоль |
| void stdcall output(const char\* buffer) | Принимает аргумент типа const char\*(buffer) представляющий строку и выводит ее значение на консоль |

void\_stdcall ConsoleWriteInt(long number) необходима для вывода целочисленных литералов.

**1.19 Ввод и вывод данных**

В языке BVA-2023 не реализованы средства ввода данных. Для вывода данных в стандартный поток вывода предусмотрен оператор write, который базируется на приватных функциях стандартной библиотеки.

**1.20 Точка входа**

В языке BVA-2023 каждая программа должна содержать главную функцию, точку входа, с которой начнется последовательное выполнение программы. В программе может быть только одна точка входа.

**1.21 Препроцессор**

Препроцессор в языке программирования BVA-2023 не предусмотрен.

**1.22 Соглашения о вызовах**

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

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

**1.23 Объектный код**

BVA-2023 транслируется в язык ассемблера.

**1.24 Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке BVA-2023 и выявления её транслятором в текущий файл протокола выводится сообщение. Их классификация сообщений приведена в таблице 1.10.

Таблица 1.10 – Классификация сообщений транслятора

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

Компилятор может обрабатывать до 1000 различных ошибок.

**1.25 Контрольный пример**

Контрольный пример представлен в Приложении А.

**Глава 2 Структура транслятора**

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

Транслятор преобразует программу, написанную на языке BVA-2023 в программу на языке ассемблера. Для указания выходных файлов используются входные параметры транслятора, которые описаны в пункте 2.2. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.



Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. Цели лексического анализатора:

− убрать все лишние пробелы;

− выполнить распознавание лексем;

− построить таблицу лексем и таблицу идентификаторов;

− при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

Семантический анализ в свою очередь является проверкой исходной программы BVA-2023 на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода BVA-2023. Для этого используются таблица лексем и идентификаторов. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке BVA-2023, прошедший все предыдущие этапы, в код на языке ассемблера [2].

## **2.2 Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка BVA-2023

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с любым расширением, в котором содержится исходный код на языке BVA-2023. Данный параметр должен быть указан обязательно. В случае если он не будет задан, то выполнение этапа трансляции не начнётся. | Не предусмотрено |
| -log:<имя\_файла> | Файл содержит в себе краткую информацию об исходном коде на языке BVA-2023. | <имя\_файла>.log |
| -out:<имя\_файла> | В этот файл будет записан результат трансляции кода на язык assembler. | <имя\_файла>.asm |

В таблице 2.1 представлены входные параметры транслятора языка BVA-2023, которые используются для формирования файлов с результатами работы лексического, синтаксического и семантического анализаторов**.**

**2.3 Протоколы, формируемые транслятором и их содержимое**

Таблица с перечнем протоколов, формируемых транслятором языка BVA-2023 и их назначением представлена в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка BVA-2023

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, “\*.log ” | Файл содержит в себе краткую информацию об исходном коде на языке BVA-2023. В этот файл выводится протокол работы анализаторов, а также различные ошибки |
| “Out.asm” | Содержит сгенерированный код на языке Ассемблера. |

**Глава 3 Разработка лексического анализатора**

**3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке BVA-2023. Структура лексического анализатора представлена на рисунке 3.1



Рисунок 3.1 – Структура лексического анализатора BVA-2023

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

## **3.2** **Входные и выходные данные лексического анализатор**

Входными параметрами являются данные исходного кода. Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. В качестве выходных параметров выступают таблица лексем и таблица идентификаторов, которые необходимы для продолжения компиляции, в качестве входных данных синтаксического анализатора.

## **3.3 Параметры лексического анализатора**

Входные параметры используются для вывода результата работы лексического анализатора. Они передаются аргументами и представлены в таблице 2.1.

## **3.4 Алгоритм лексического анализа**

Алгоритм работы лексического анализа заключается в последовательном распознавании и разборе цепочек исходного кода и заполнение таблиц идентификаторов и лексем.

 Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов. В случае, если подходящий автомат не был обнаружен, запоминается номер строки, в которой находился этот токен и выводится сообщение об ошибке. Если токен разобран, то дальнейшие действия, которые будут с ним производиться, будут зависеть от того, чем он является.

Регулярные выражения – аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа [3].

Регулярное выражение для ключевого слова set: «set». Граф конечного автомата для этой лексемы представлен на рисунке 3.1. S0 – начальное состояние, S3 – конечное состояние автомата.

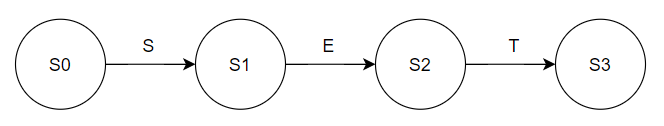


Рисунок 3.1 – Граф переходов для цепочки «set»

**3.5 Контроль входных символов**

Таблица для контроля входных символов представлена в Приложении Ж.

Принцип работы таблицы заключается в соответствии значения каждому элементу в шестнадцатеричной системе счисления значению в таблице Windows- 1251.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, S – пробельный символ, Q – символ одинарной кавычки, I – игнорированный символ, O – операция.

## **3.6 Удаление избыточных символов**

Удаление избыточных символов не предусмотрено, так как после проверки на допустимость символов исходный код на языке программирования BVA-2023 разбивается на токены, которые записываются в очередь.

## **3.7 Перечень ключевых слов**

Лексемы – это символы, соответствующие ключевым словам, символам операций и сепараторам, необходимые для упрощения дальнейшей обработки исходного кода программы [4]. Данное соответствие описано в таблице 3.1.

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

|  |  |  |  |
| --- | --- | --- | --- |
| Тип цепочки | Примечание | Цепочка | Лексема |
| Тип данных | Целочисленный тип данных | int | t |
| Символьный тип данных | symb | t |
| Лексем | Объявление переменной | set | d |
| Оператор вывода | write | p |
| Объявление функции | func | f |
| Возврат значения из функции | ret | r |
| Условный оператор | if | e |
| Блок инструкции цикла | [ | [ |
| ] | ] |
| Блок функции | { | { |
| } | } |
| Изменение приоритетности в выражении и отделение параметров функций | ( | ( |
| ) | ) |
| Сепараторы | ; | ; |
| , | , |
| Оператор присваивания | = | = |
| Оператор | Знаки арифметических операций | + | + |
| - | - |
| \* | \* |
| ~ | ~ |
| & | & |
| | | | |
| Идентификатор | - | [a-z;A-Z]+  [a-z;A-Z;0-9]\* | i |
| Литерал | Целочисленный литерал | [1-9]+[0-9]\* | l |
| Строковый литерал | [a-z;A-Z]+  [a-z;A-Z;0-9]\* | l |
| Точка входа | - | main | m |

В Приложении А находятся конечные автоматы, соответствующие лексемам языка BVA-2023.

## **3.8 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных языка BVA-2023, используемых для хранения, представлены в Приложении А. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде, номер столбца в исходном коде, индекс таблицы идентификаторов (если нет соответствующего идентификатора, то индекс равен -1), а также специальное поле, в котором хранится значение лексемы. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора, его значение, а также бинарное поле для определения внешний ли идентификатор.

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

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 121-131. Перечень сообщений лексического анализатора представлен в листинге 3.1.

|  |
| --- |
| ERROR\_ENTRY(121, "[LA]: Используется необъявленный идентификатор"),  ERROR\_ENTRY(122, "[LA]: Переполнение таблицы идентификаторов"),  ERROR\_ENTRY(123, "[LA]: Переполнение таблицы лексем"),  ERROR\_ENTRY(124, "[LA]: Отсутствует точка входа или их несколько"),  ERROR\_ENTRY(125, "[LA]: Обнаружено несколько точек входа"),  ERROR\_ENTRY(126, "[LA]: У точки входа нет тела"),  ERROR\_ENTRY(127, "[LA]: Несогласованность скобок"),  ERROR\_ENTRY(128, "[LA]: Необъявленный идентификатор"),  ERROR\_ENTRY(129, "[LA]: Переопределение идентификатора"),  ERROR\_ENTRY(131, "[LA]: Слишком длинное имя идентификатора"), |

Листинг 3.1 – Перечень ошибок лексического анализатора

Также сам текст ошибки содержит в себе префикс [LA].

## **3.10 Принцип обработки ошибок**

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

## **3.11 Контрольный пример**

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

**Глава 4 Разработка синтаксического анализатора**

**4.1 Структура синтаксического анализатора**

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

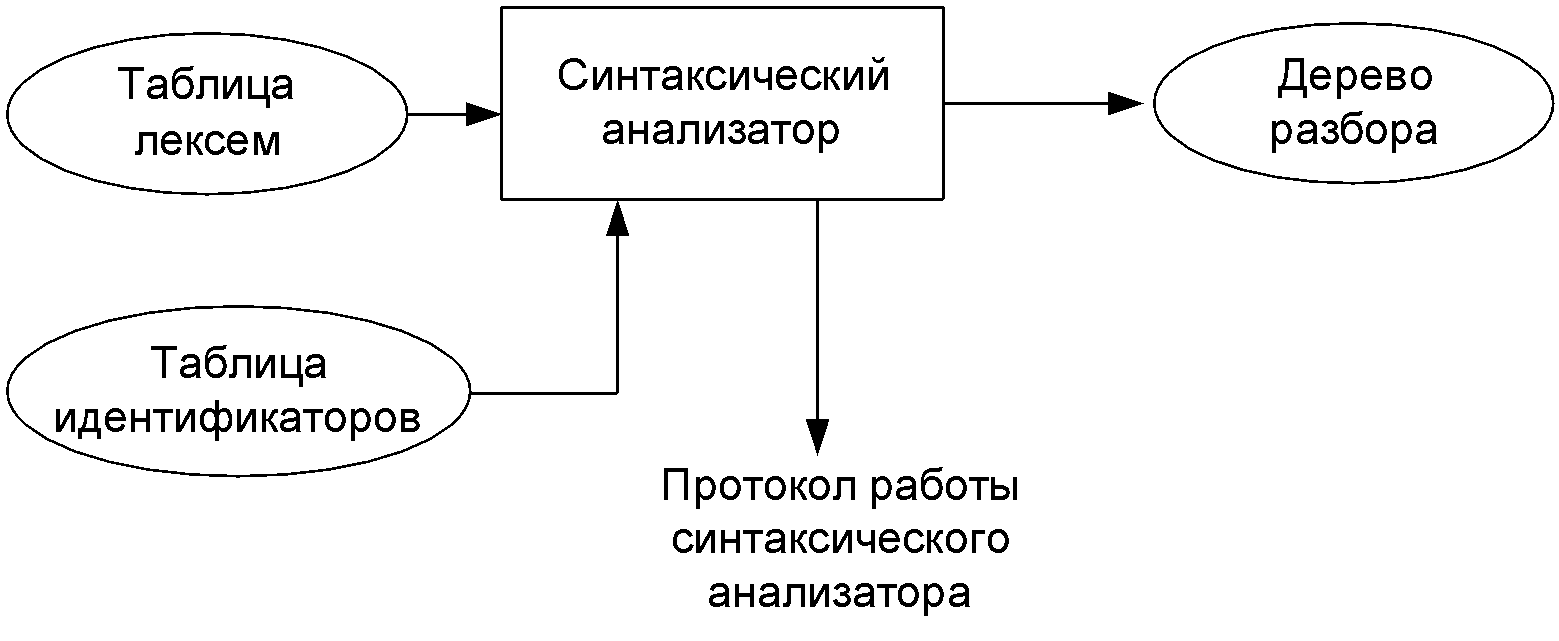


Рисунок 4.1 – Структура синтаксического анализатора

Входом для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные после фазы лексического анализа. Выходом – дерево разбора.

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

В синтаксическом анализаторе транслятора языка BVA-2023 используется контекстно-свободная грамматика , где

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

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

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

S – начальный символ грамматики, являющийся нетерминалом.

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

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

Грамматика языка BVA-2023 представлена в приложении Б.

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

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

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов BVA-2023

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | m{NrE;}  m{rE;}  tfi(F){NrE;}S | Порождает правила, описывающее общую структуру программы |
| N | dti;  dti;N  dtfi(F);  dtfi(F);N  dtfi();  dtfi()N;  i=E;  i=E;N  pi;  pi;N  pl;  pl;N  pi(W);  pi(W);N  T(C)[N]  T(C)[N]N  e(C)[N]  e(C)[N]  rE; | Порождает правила, описывающие конструкции языка |
| E | i  l  (E)  (E)M  i()  i()E  iM  lM  (E)M  i(W)M | Порождает правила, описывающие выражения |
| F | ti  ti,f | Порождает правила, описывающие параметры локальной функции при её объявлении |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| W | i  l  i,W  l,W | Порождает правила, описывающие принимаемые параметры функции |
| C | i  l  i&l  i|i  i&i  i|l | Порождает правила, описывающие условное выражение в операторе цикла |
| M | vE  v(E)  v(E)M  vEM | Порождает правила, описывающие знаки арифметических операций (v – знаки операций: +,-,\*,|,~,&) |

В таблице 4.1 представлено описание нетерминальных символов и соответствующих правил переходов в контекстно-свободной грамматике языка BVA-2023.

## **4.3 Построение конечного магазинного автомата**

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

Таблица 4.2 – Описание компонентов магазинного автомата

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

Продолжение таблицы 4.2

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | 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 (нетерминальный символ А) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

Протокол и ошибки работы синтаксического анализатора выводятся в лог журнал.

## **4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного конечного автомата и структуру грамматики Грейбах, описывающей правила языка BVA-2023. Данные структуры представлены в Приложении В.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата:

* В магазин записывается стартовый символ;
* На основе полученных ранее таблиц формируется входная лента;
* Запускается автомат;
* Выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;
* Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала;
* Если в магазине встретился нетерминал, переходим к пункту 4;
* Если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно и формируется дерево разбора. Иначе генерируется исключение.

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

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

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

Индексы ошибок, обнаруживаемых синтаксическим анализатором, находятся в диапазоне 600-606. Перечень сообщений синтаксического анализатора представлен в листинге 4.1.

|  |
| --- |
| ERROR\_ENTRY(600, "[SA]: Неверная структура программы"),  ERROR\_ENTRY(601, "[SA]: Ошибочный оператор"),  ERROR\_ENTRY(602, "[SA]: Ошибка в выражении"),  ERROR\_ENTRY(603, "[SA]: Ошибка в параметрах функции"),  ERROR\_ENTRY(604, "[SA]: Ошибка в параметрах вызываемой функции"),  ERROR\_ENTRY(605, "[SA]: Ошибка в подвыражении"),  ERROR\_ENTRY(606, "[SA]: Ошибка в if"), |

Листинг 4.1 – Перечень сообщений синтаксического анализатора

Также сам текст ошибки содержит в себе префикс [SА].

## **4.8 Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

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

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке BVA-2023 представлен в Приложении Г. Дерево разбора исходного кода приложен к проекту.

# **Глава 5 Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ происходит при выполнении фазы лексического анализа и реализуется в виде отдельных проверок текущих ситуаций в конкретных случаях: установки флага или нахождении в особом месте программы (оператор выхода из функции, оператор ветвления, вызов функции стандартной библиотеки) [5]. Структура семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1 – структура семантического анализатор

## **5.2 Функции семантического анализатора**

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

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

Все ошибка семантического анализатора имеют идентификатор свыше 700. Сообщения, формируемые семантическим анализатором, представлены в Листинге 5.1.

|  |
| --- |
| ERROR\_ENTRY(700, "[SemA]: Ошибка в возвращаемом значении"),  ERROR\_ENTRY(701, "[SemA]: Ошибка в параметрах функции"),  ERROR\_ENTRY(702, "[SemA]: Ошибка в параметрах функции библиотеки"),  ERROR\_ENTRY(704, "[SemA]: Ошибка в значении для вывода в консоль"),  ERROR\_ENTRY(705, "[SemA]: Несоответствие возвращаемого и присваемого типов данных"),  ERROR\_ENTRY(706, "[SemA]: Несоответсвие присваимаего типа данных"),  ERROR\_ENTRY(707, "[SemA]: Несоответствие типа функции и возвращаемого значения"),  ERROR\_ENTRY(708, "[SemA]: Main Должен возвращать числовое значение"),  ERROR\_ENTRY(709, "[SemA]: Использование имени переменной в качестве функции"),  ERROR\_ENTRY(710, "[SemA]: Ошибка в условном операторе"), |

Листинг 5.1 – Перечень сообщений семантического анализатора

Все ошибки имеют приставку [SemA].

## **5.4 Принцип обработки ошибок**

При обнаружении хотя бы одной ошибки транслятор завершит свою работу c

Записью информации об ошибке в лог файл.

## **5.5 Контрольный пример**

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

# **Глава 6 Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке BVA-2023 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как: +, -, \* и (). А также вызовы функций как операнды арифметических выражений. Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке BVA-2023

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( | 0 |
| ) | 0 |
| , | 1 |
| - | 2 |
| + | 2 |
| \* | 3 |
| | | 3 |
| ~ | 3 |
| & | 3 |

Все операции используются только для целочисленного типа данных.

**6.2 Польская запись и принцип её построения**

Выражения в языке BVA-2023 преобразовываются к обратной польской записи.

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок [2].

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

Алгоритм построения:

– читаем очередной символ;

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

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

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

– исходная строка просматривается слева направо;

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

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

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

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

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись.

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

Все выражения могут быть преобразованы в обратную польскую запись по такому же принципу.

## **6.3 Программная реализация обработки выражений**

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

## **6.4 Контрольный пример**

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

# **Глава 7 Генерация кода**

## **7.1 Структура генератора кода**

Генерация объектного кода – это перевод компилятором внутреннего представления исходной программы BVA-2023 в цепочку символов выходного языка. На вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом.



Рисунок 7.1 — Структура генератора кода

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

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Соответствия между типами данных идентификаторов на языке BVA-2023 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка BVA-2023 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке BVA-2023 | Тип идентификатора на языке ассемблера | Пояснение |
| symb | BYTE | Хранит символьный тип данных |
| int | SDWORD | Хранит целочисленный тип данных. |
| Лексема | BYTE  SDWORD | Литералы:  целочисленные, строковые |

Идентификаторы языка BVA-2023 размещены в сегменте данных (.data). Литералы – в сегменте констант (.const).

**7.3 Статическая библиотека**

В языке BVA-2023 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера.

Таблица 7.2 – Функции статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| long stdcall step(long int1, long int2) | Возведение int1 в степень int2 |
| long stdcall iabs(long aint) | Вычисляет абсолютное значение |
| void stdcall ConsoleWriteInt(long number) | Выводит значение на консоль |
| void stdcall output(const char\* buffer) | Принимает аргумент типа const char\*(buffer) представляющий строку и выводит ее значение на консоль |

В итоге, таблица 7.2 описывает функции, доступные в статической библиотеке языка BVA-2023. Эти функции могут быть полезны при работе с выводом на консоль, а также выполняют математические операции.

## **7.4 Особенности алгоритма генерации кода**

Алгоритм генерации кода выглядит следующим образом:

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

|  |
| --- |
| 1. .586 2. .model flat, stdcall 3. includelib libucrt.lib 4. includelib kernel32.lib 5. includelib ../Debug/BVA\_Lib.lib 6. ExitProcess PROTO : DWORD 7. ConsoleWriteInt PROTO : SDWORD 8. iabs PROTO : SDWORD 9. step PROTO : SDWORD, : SDWORD   10) output PROTO : SDWORD  11) stack 4096 |

Листинг 7.1 – пример заголовочной информации

* Проходит полностью таблицу идентификаторов и заполняет поле .const литералами. Результат представлен в листинге 7.2.

|  |
| --- |
| .const  l0 SDWORD 4  l1 SDWORD 6  l2 SDWORD 1  l3 SDWORD 0  l4 BYTE '... func step ...', 0  l5 SDWORD 2  l6 BYTE '... summ 4 and 6 ...', 0  l7 BYTE '... sub 4 and 6 ...', 0  l8 BYTE '... func iabs ...', 0  l9 BYTE '.. begin ..', 0  l10 BYTE '.. later ..', 0  l11 BYTE '... mul 4 and 6 ...', 0  l12 BYTE '... summ hex ...', 0  l13 SDWORD 10  l14 SDWORD 31  l15 BYTE '... AND 1 and 0 ...', 0  l16 BYTE '... OR 1 and 0 ...', 0  l17 BYTE '... NOT a ...', 0  l18 BYTE ' ... func expression ...', 0  l19 BYTE 'W', 0  l20 BYTE '... Out symb ...', 0  l21 BYTE '... If and else ...', 0  l22 BYTE 'if true', 0  l23 BYTE 'else', 0  l24 BYTE 'if', 0 |

Листинг 7.2 – Пример заполнения поля .const

* Проходим таблицу идентификаторов и объявляем переменные в поле .data. Результат заполнения поля .data представлен в листинге 7.3.

|  |
| --- |
| .data  expressionsm SDWORD 0  maina SDWORD 0  mainb SDWORD 0  mainc SDWORD 0  maind SDWORD 0  mainz SDWORD 0  mainone SDWORD 0  mainzero SDWORD 0  mainsa SDWORD 0 |

Листинг 7.3 – Пример заполнения поля .data

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

|  |
| --- |
| .code  expression PROC b: SDWORD, a: SDWORD  push a  push b  ;\/(MUL)\*\/  pop eax  pop ebx  MUL ebx  push eax  mov eax, expressionsm  ret  expression ENDP |

Листинг 7.4 – Пример функции, полученной в результате генерации

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

**7.5 Параметры, управляющие генерацией кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного кода программы на языке BVA-2023. Результаты работы генератора кода выводятся в файл с расширением .asm.

## **7.6 Контрольный пример**

Генерируемый код записывается в файл «in.txt.asm». Сгенерированный код можно посмотреть в Приложении Е.

# **Глава 8 Тестирование транслятора**

**8.1 Общие положения**

Тестирование должно покрывать как можно больше сценариев использования языка и его конструкций. Все тесты были представлены для типичных ошибок пользователей при использовании языка. Когда компилятор обнаруживает ошибку, он записывает информацию о ней в протокол, содержащий номер ошибки и диагностическое сообщение, помогающее разработчику понять причину ошибки компиляции. После обнаружения ошибки компилятор может продолжить анализ, чтобы найти другие возможные ошибки. Результаты тестирования записываются в файл .log.

**8.2 Результаты тестирования**

В языке BVA-2023 не разрешается использовать запрещённые входным алфавитом символы где-либо кроме строковых или символьных переменных. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| aint | Ошибка 111: Недопустимый символ в исходном файле (-in), позиция 0, строка 0 |

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

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| write f; | f - 21 ---> Ошибка 128 [LA]: Необъявленный идентификатор |
| {} | Ошибка 124 [LA]: Отсутствует точка входа или их несколько в функции |
| main{ main{}  } | Ошибка 124 [LA]: Отсутствует точка входа или их несколько в функции |
| set int a;  set symb a; | a - 10 ---> Ошибка 129 [LA]: Переопределение идентификатора |
| set int qwertyqwertyqwerty; | Ошибка 131 [LA]: Слишком длинное имя идентификатора |

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

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

Таблица 8.3 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| if(flag)  z = a;  ]; | 606: строка 48,[SA]: Ошибка в if |
| c a+b; | 601: строка 26,[SA]: Ошибочный оператор |

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

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| set symb a;  set int b;  c = st(a, b); | Ошибка 706 [SemA]: Несоответсвие присваимаего типа данных, строка 20 |
| int func sum(int a, int b){  set symb f;  ret f;} | Ошибка 707 [SemA]: Несоответствие типа функции и возвращаемого значения, строка 4 |
| set symb f;  f =1; | Ошибка 706 [SemA]: Несоответсвие присваимаего типа данных, строка 17 |
| main{  set symb f;  ret f;  }; | Ошибка 708 [SemA]: Main должен возвращать числовое значение, строка 5 |

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

# **Заключение**

По окончании выполнения всех пунктов, изложенных ранее, получили рабочий транслятор языка программирования BVA-2023 на язык ассемблера.

Таким образом, были выполнены основные задачи данной курсовой работы:

* Сформулирована спецификация языка BVA-2023;
* Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
* Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
* Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
* Осуществлена программная реализация синтаксического анализатора;
* Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
* Разработан транслятор кода на язык ассемблера;
* Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка BVA-2023 включает:

* 2 типа данных;
* Поддержка операторов вывода;
* Возможность вызова функций стандартной библиотеки;
* Наличие 3 арифметических операторов и 3 побитовых операторов для вычисления выражений;
* Поддержка функций, операторов условия;
* Структурированная и классифицированная система для обработки ошибок пользователя.

# **Список используемых источников**

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

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

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

4. Принципы работы транслятора [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/435520/. – Дата доступа: 23.11.2023.

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

# **ПРИЛОЖЕНИЕ А**

|  |
| --- |
| int func expression(int a, int b){  set int sm;  sm = a \* b - (a + b);  ret sm;  }  main{  set int func step(int x, int y);  set int func iabs(int x);  set int a;  set int b;  set int c;  set int d;  set int z;  set int one;  set int zero;  a = 4;  b = 6;  one = 1;  zero = 0;  write '... func step ...';  c = step(a,2);  write c;  write '... summ 4 and 6 ...';  c = a + b;  write c;  write '... sub 4 and 6 ...';  c = a - b;  write c;  write '... func iabs ...';  d = iabs(c);  write '.. begin ..';  write c;  write '.. later ..';  write d;  write '... mul 4 and 6 ...';  c = b \* a;  write c;  write '... summ hex ...';  a = 0x0A;  b = 0x1F;  c = b + a;  write c;  write '... AND 1 and 0 ...';  z = zero & one;  write z;  write '... OR 1 and 0 ...';  z = zero | one;  write z;  write '... NOT a ...';  z = a ~ 4;  write z;  write ' ... func expression ...';  a = expression(a, b);  write a;  set symb sa;  sa = 'W';    write '... Out symb ...';  write sa;  write '... If and else ...';    if(one)  [  write 'if true';  if(one & zero)  [  write 'if';  ]  else(zero | one)  [  write 'else';  ]  ]    ret 0;  } |

Листинг A.1 – Контрольный пример

|  |
| --- |
| +-+-+-+-+-+-+-+- Таблица идентификаторов +-+-+-+-+-+-+-+-  0 ----> - expression - INT - FUNCTION - 2 - 0  1 ----> expression - a - INT - PARAMETER - 5 - 0  2 ----> expression - b - INT - PARAMETER - 8 - 0  3 ----> expression - sm - INT - VARIABLE - 13 - 0  4 ----> - main - INT - FUNCTION - 31 - 0  5 ----> - step - INT - FUNCTION - 36 - 0  6 ----> step - x - INT - PARAMETER - 39 - 0  7 ----> step - y - INT - PARAMETER - 42 - 0  8 ----> - iabs - INT - FUNCTION - 48 - 0  9 ----> iabs - x - INT - PARAMETER - 51 - 0  10 ----> main - a - INT - VARIABLE - 56 - 0  11 ----> main - b - INT - VARIABLE - 60 - 0  12 ----> main - c - INT - VARIABLE - 64 - 0  13 ----> main - d - INT - VARIABLE - 68 - 0  14 ----> main - z - INT - VARIABLE - 72 - 0  15 ----> main - one - INT - VARIABLE - 76 - 0  16 ----> main - zero - INT - VARIABLE - 80 - 0  17 ----> l0 - INT - LITERAL - 84 - 4  18 ----> l1 - INT - LITERAL - 88 - 6  19 ----> l2 - INT - LITERAL - 92 - 1  20 ----> l3 - INT - LITERAL - 96 - 0  21 ----> l4 - STR - LITERAL - 99 - '... func step ...'  22 ----> l5 - INT - LITERAL - 107 - 2  23 ----> l6 - STR - LITERAL - 114 - '... summ 4 and 6 ...'  24 ----> l7 - STR - LITERAL - 126 - '... sub 4 and 6 ...'  25 ----> l8 - STR - LITERAL - 138 - '... func iabs ...'  26 ----> l9 - STR - LITERAL - 148 - '.. begin ..'  27 ----> l10 - STR - LITERAL - 154 - '.. later ..'  28 ----> l11 - STR - LITERAL - 160 - '... mul 4 and 6 ...'  29 ----> l12 - STR - LITERAL - 172 - '... summ hex ...'  30 ----> l13 - INT - LITERAL - 176 - 10  31 ----> l14 - INT - LITERAL - 180 - 31  32 ----> l15 - STR - LITERAL - 192 - '... AND 1 and 0 ...'  33 ----> l16 - STR - LITERAL - 204 - '... OR 1 and 0 ...'  34 ----> l17 - STR - LITERAL - 216 - '... NOT a ...'  35 ----> l18 - STR - LITERAL - 228 - ' ... func expression ...'  36 ----> main - sa - CHAR - VARIABLE - 244 -  37 ----> l19 - CHAR - LITERAL - 248 - 'W'  38 ----> l20 - STR - LITERAL - 251 - '... Out symb ...'  39 ----> l21 - STR - LITERAL - 257 - '... If and else ...'  40 ----> l22 - STR - LITERAL - 265 - 'if true'  41 ----> l23 - STR - LITERAL - 276 - 'else'  42 ----> l24 - STR - LITERAL - 287 - 'if'  +-+-+-+-+-+-+-+- Таблица идентификаторов +-+-+-+-+-+-+-+- |

Листинг A.2 – Таблица идентификаторов

|  |  |
| --- | --- |
| +-+-+-+-+-+-+-+- Таблица лексем +-+-+-+-+-+-+-+-  0 ----> -1 - t - 0  1 ----> -1 - f - 0  2 ----> 0 - i - 0  3 ----> -1 - ( - 0  4 ----> -1 - t - 0  5 ----> 1 - i - 0  6 ----> -1 - , - 0  7 ----> -1 - t - 0  8 ----> 2 - i - 0  9 ----> -1 - ) - 0  10 ----> -1 - { - 0  11 ----> -1 - d - 1  12 ----> -1 - t - 1  13 ----> 3 - i - 1  14 ----> -1 - ; - 1  15 ----> 3 - i - 2  16 ----> -1 - = - 2  17 ----> 1 - i - 2  18 ----> -1 - \* - 2  19 ----> 2 - i - 2  20 ----> -1 - - - 2  21 ----> -1 - ( - 2  22 ----> 1 - i - 2  23 ----> -1 - + - 2  24 ----> 2 - i - 2  25 ----> -1 - ) - 2  26 ----> -1 - ; - 2  27 ----> -1 - r - 3  28 ----> 3 - i - 3  29 ----> -1 - ; - 3  30 ----> -1 - } - 4  31 ----> 4 - m - 5  32 ----> -1 - { - 6  33 ----> -1 - d - 7  34 ----> -1 - t - 7  35 ----> -1 - f - 7  36 ----> 5 - i - 7  37 ----> -1 - ( - 7  38 ----> -1 - t - 7  39 ----> 6 - i - 7  40 ----> -1 - , - 7  41 ----> -1 - t - 7  42 ----> 7 - i - 7  43 ----> -1 - ) - 7  44 ----> -1 - ; - 7  45 ----> -1 - d - 8  46 ----> -1 - t - 8  47 ----> -1 - f - 8  48 ----> 8 - i - 8  49 ----> -1 - ( - 8  50 ----> -1 - t - 8  51 ----> 9 - i - 8  52 ----> -1 - ) - 8  53 ----> -1 - ; - 8  54 ----> -1 - d - 9  55 ----> -1 - t - 9  56 ----> 10 - i - 9  57 ----> -1 - ; - 9  58 ----> -1 - d - 10  59 ----> -1 - t - 10  60 ----> 11 - i - 10  61 ----> -1 - ; - 10  62 ----> -1 - d - 11  63 ----> -1 - t - 11  64 ----> 12 - i - 11  65 ----> -1 - ; - 11  66 ----> -1 - d - 12  67 ----> -1 - t - 12  68 ----> 13 - i - 12  69 ----> -1 - ; - 12  70 ----> -1 - d - 13  71 ----> -1 - t - 13  72 ----> 14 - i - 13  73 ----> -1 - ; - 13  74 ----> -1 - d - 14  75 ----> -1 - t - 14  76 ----> 15 - i - 14  77 ----> -1 - ; - 14  78 ----> -1 - d - 15  79 ----> -1 - t - 15  80 ----> 16 - i - 15  81 ----> -1 - ; - 15  82 ----> 10 - i - 16  83 ----> -1 - = - 17  84 ----> 17 - l - 17  85 ----> -1 - ; - 17  86 ----> 11 - i - 18  87 ----> -1 - = - 18  88 ----> 18 - l - 18  89 ----> -1 - ; - 18  90 ----> 15 - i - 19  91 ----> -1 - = - 19  92 ----> 19 - l - 19  93 ----> -1 - ; - 19  94 ----> 16 - i - 20  95 ----> -1 - = - 20  96 ----> 20 - l - 20  97 ----> -1 - ; - 20  98 ----> -1 - p - 21  99 ----> 21 - l - 22  100 ----> -1 - ; - 22  101 ----> 12 - i - 23  102 ----> -1 - = - 23  103 ----> 5 - i - 23  104 ----> -1 - ( - 23  105 ----> 10 - i - 23  106 ----> -1 - , - 23  107 ----> 22 - l - 23  108 ----> -1 - ) - 23  109 ----> -1 - ; - 23  110 ----> -1 - p - 24  111 ----> 12 - i - 24  112 ----> -1 - ; - 24  113 ----> -1 - p - 25  114 ----> 23 - l - 25  115 ----> -1 - ; - 25  116 ----> 12 - i - 26  117 ----> -1 - = - 26  118 ----> 10 - i - 26  119 ----> -1 - + - 26  120 ----> 11 - i - 26  121 ----> -1 - ; - 26  122 ----> -1 - p - 27  123 ----> 12 - i - 27  124 ----> -1 - ; - 27  125 ----> -1 - p - 28  126 ----> 24 - l - 28  127 ----> -1 - ; - 28  128 ----> 12 - i - 29  129 ----> -1 - = - 29  130 ----> 10 - i - 29  131 ----> -1 - - - 29  132 ----> 11 - i - 29  133 ----> -1 - ; - 29  134 ----> -1 - p - 30  135 ----> 12 - i - 30  136 ----> -1 - ; - 30  137 ----> -1 - p - 31  138 ----> 25 - l - 31  139 ----> -1 - ; - 31  140 ----> 13 - i - 32  141 ----> -1 - = - 32  142 ----> 8 - i - 32  143 ----> -1 - ( - 32  144 ----> 12 - i - 32  145 ----> -1 - ) - 32  146 ----> -1 - ; - 32  147 ----> -1 - p - 33 | 148 ----> 26 - l - 33  149 ----> -1 - ; - 33  150 ----> -1 - p - 34  151 ----> 12 - i - 34  152 ----> -1 - ; - 34  153 ----> -1 - p - 35  154 ----> 27 - l - 35  155 ----> -1 - ; - 35  156 ----> -1 - p - 36  157 ----> 13 - i - 36  158 ----> -1 - ; - 36  159 ----> -1 - p - 37  160 ----> 28 - l - 37  161 ----> -1 - ; - 37  162 ----> 12 - i - 38  163 ----> -1 - = - 38  164 ----> 11 - i - 38  165 ----> -1 - \* - 38  166 ----> 10 - i - 38  167 ----> -1 - ; - 38  168 ----> -1 - p - 39  169 ----> 12 - i - 39  170 ----> -1 - ; - 39  171 ----> -1 - p - 40  172 ----> 29 - l - 40  173 ----> -1 - ; - 40  174 ----> 10 - i - 41  175 ----> -1 - = - 41  176 ----> 30 - l - 41  177 ----> -1 - ; - 41  178 ----> 11 - i - 42  179 ----> -1 - = - 42  180 ----> 31 - l - 42  181 ----> -1 - ; - 42  182 ----> 12 - i - 43  183 ----> -1 - = - 43  184 ----> 11 - i - 43  185 ----> -1 - + - 43  186 ----> 10 - i - 43  187 ----> -1 - ; - 43  188 ----> -1 - p - 44  189 ----> 12 - i - 44  190 ----> -1 - ; - 44  191 ----> -1 - p - 45  192 ----> 32 - l - 46  193 ----> -1 - ; - 46  194 ----> 14 - i - 47  195 ----> -1 - = - 47  196 ----> 15 - i - 47  197 ----> -1 - & - 47  198 ----> 16 - i - 47  199 ----> -1 - ; - 47  200 ----> -1 - p - 48  201 ----> 14 - i - 48  202 ----> -1 - ; - 48  203 ----> -1 - p - 49  204 ----> 33 - l - 49  205 ----> -1 - ; - 49  206 ----> 14 - i - 50  207 ----> -1 - = - 50  208 ----> 15 - i - 50  209 ----> -1 - | - 50  210 ----> 16 - i - 50  211 ----> -1 - ; - 50  212 ----> -1 - p - 51  213 ----> 14 - i - 51  214 ----> -1 - ; - 51  215 ----> -1 - p - 52  216 ----> 34 - l - 52  217 ----> -1 - ; - 52  218 ----> 14 - i - 53  219 ----> -1 - = - 53  220 ----> 10 - i - 53  221 ----> -1 - ~ - 53  222 ----> 17 - l - 53  223 ----> -1 - ; - 53  224 ----> -1 - p - 54  225 ----> 14 - i - 54  226 ----> -1 - ; - 54  227 ----> -1 - p - 55  228 ----> 35 - l - 56  229 ----> -1 - ; - 56  230 ----> 10 - i - 57  231 ----> -1 - = - 57  232 ----> 0 - i - 57  233 ----> -1 - ( - 57  234 ----> 10 - i - 57  235 ----> -1 - , - 57  236 ----> 11 - i - 57  237 ----> -1 - ) - 57  238 ----> -1 - ; - 57  239 ----> -1 - p - 58  240 ----> 10 - i - 58  241 ----> -1 - ; - 58  242 ----> -1 - d - 59  243 ----> -1 - t - 60  244 ----> 36 - i - 60  245 ----> -1 - ; - 60  246 ----> 36 - i - 61  247 ----> -1 - = - 61  248 ----> 37 - l - 61  249 ----> -1 - ; - 61  250 ----> -1 - p - 62  251 ----> 38 - l - 63  252 ----> -1 - ; - 63  253 ----> -1 - p - 64  254 ----> 36 - i - 64  255 ----> -1 - ; - 64  256 ----> -1 - p - 65  257 ----> 39 - l - 66  258 ----> -1 - ; - 66  259 ----> -1 - e - 67  260 ----> -1 - ( - 67  261 ----> 15 - i - 67  262 ----> -1 - ) - 67  263 ----> -1 - [ - 68  264 ----> -1 - p - 69  265 ----> 40 - l - 69  266 ----> -1 - ; - 69  267 ----> -1 - ] - 70  268 ----> -1 - T - 71  269 ----> -1 - ( - 72  270 ----> 11 - i - 72  271 ----> -1 - < - 72  272 ----> 10 - i - 72  273 ----> -1 - ) - 72  274 ----> -1 - [ - 73  275 ----> -1 - p - 74  276 ----> 41 - l - 74  277 ----> -1 - ; - 74  278 ----> -1 - ] - 75  279 ----> -1 - e - 76  280 ----> -1 - ( - 77  281 ----> 11 - i - 77  282 ----> -1 - > - 77  283 ----> 10 - i - 77  284 ----> -1 - ) - 77  285 ----> -1 - [ - 78  286 ----> -1 - p - 79  287 ----> 42 - l - 79  288 ----> -1 - ; - 79  289 ----> -1 - ] - 80  290 ----> -1 - r - 81  291 ----> 20 - l - 82  292 ----> -1 - ; - 82  293 ----> -1 - } - 83 |

Листинг А.3 – Таблица лексем

|  |  |
| --- | --- |
| #define FST\_MAIN FST(5, LEX\_MAIN,\  NODE(1, RELATION('m', 1)),\  NODE(1, RELATION('a', 2)),\  NODE(1, RELATION('i', 3)),\  NODE(1, RELATION('n', 4)),\  NODE())  #define FST\_PRINT FST(6, LEX\_PRINT,\  NODE(1, RELATION('w', 1)),\  NODE(1, RELATION('r', 2)),\  NODE(1, RELATION('i', 3)),\  NODE(1, RELATION('t', 4)),\  NODE(1, RELATION('e', 5)),\  NODE())  #define FST\_RETURN FST(4, LEX\_RETURN,\  NODE(1, RELATION('r', 1)),\  NODE(1, RELATION('e', 2)),\  NODE(1, RELATION('t', 3)),\  NODE())  #define FST\_DECLARE FST(4, LEX\_DECLARE,\  NODE(1, RELATION('s', 1)),\  NODE(1, RELATION('e', 2)),\  NODE(1, RELATION('t', 3)),\  NODE())  #define FST\_INTEGER FST(4, LEX\_INTEGER,\  NODE(1, RELATION('i', 1)),\  NODE(1, RELATION('n', 2)),\  NODE(1, RELATION('t', 3)),\  NODE())  #define FST\_CHAR FST(5, LEX\_CHAR,\  NODE(1, RELATION('s', 1)),\  NODE(1, RELATION('y', 2)),\  NODE(1, RELATION('m', 3)),\  NODE(1, RELATION('b', 4)),\  NODE())  #define FST\_FUNCTION FST(5, LEX\_FUNCTION,\  NODE(1, RELATION('f', 1)),\  NODE(1, RELATION('u', 2)),\  NODE(1, RELATION('n', 3)),\  NODE(1, RELATION('c', 4)),\  NODE())  #define FST\_IF FST(3, LEX\_IF,\  NODE(1, RELATION('i', 1)),\  NODE(1, RELATION('f', 2)),\  NODE())  #define FST\_ELSE FST(5, LEX\_ELSE,\  NODE(1, RELATION('e', 1)),\  NODE(1, RELATION('l', 2)),\  NODE(1, RELATION('s', 3)),\  NODE(1, RELATION('e', 4)),\  NODE())  #pragma endregion  #pragma region SYMBOLS  #define FST\_LEFT\_BRACKET FST(2, LEX\_LEFTHESIS,\  NODE(1, RELATION('(', 1)),\  NODE()\  )  #define FST\_LEFTSQUARE FST(2, LEX\_LEFTSQUARE,\  NODE(1, RELATION('[', 1)),\  NODE()\  )  #define FST\_RIGHTSQUARE FST(2, LEX\_RIGHTSQUARE,\  NODE(1, RELATION(']', 1)),\  NODE()\  )  #define FST\_RIGHT\_BRACKET FST(2, LEX\_RIGHTHESIS,\  NODE(1, RELATION(')', 1)),\  NODE()\  )  #define FST\_LEFT\_BRACE FST(2, LEX\_RIGHTBRACE,\  NODE(1, RELATION('{', 1)),\  NODE()\  )  #define FST\_RIGHT\_BRACE FST(2, LEX\_BRACELET,\  NODE(1, RELATION('}', 1)),\  NODE()\  )  #define FST\_PLUS FST(2, LEX\_PLUS,\  NODE(1, RELATION('+', 1)),\  NODE()\  )  #define FST\_MINUS FST(2, LEX\_MINUS,\  NODE(1, RELATION('-', 1)),\  NODE()\  )  #define FST\_MULTI FST(2, LEX\_STAR,\  NODE(1, RELATION('\*', 1)),\  NODE()\  )  #define FST\_AND FST(2, LEX\_AND,\  NODE(1, RELATION('&', 1)),\  NODE()\  )  #define FST\_OR FST(2, LEX\_OR,\  NODE(1, RELATION('|', 1)),\  NODE()\  )  #define FST\_INVERSE FST(2, LEX\_INVERSE,\  NODE(1, RELATION(':', 1)),\  NODE()\  )  #define FST\_COMMA FST(2, LEX\_COMMA,\  NODE(1, RELATION(',', 1)),\  NODE()\  )  #define FST\_SEMICOLON FST(2, LEX\_SEMICOLON,\  NODE(1, RELATION(';', 1)),\  NODE()\  )  #define FST\_EQUALLY FST(2, LEX\_EQUAL,\  NODE(1, RELATION('=', 1)),\  NODE()\  )  #define FST\_LOWER FST(2, LEX\_LOWER,\  NODE(1, RELATION('<', 1)),\  NODE()\  )  #define FST\_HIGHER FST(2, LEX\_HIGHER,\  NODE(1, RELATION('>', 1)),\  NODE()\  )  #pragma endregion  #pragma region LITERALS  #define FST\_NUMBERS FST(1, LEX\_LITERAL,\  NODE(10, RELATION('0', 0), RELATION('1', 0), RELATION('2', 0), RELATION('3', 0), RELATION('4', 0), RELATION('5', 0),\  RELATION('6', 0), RELATION('7', 0), RELATION('8', 0), RELATION('9', 0)),\  NODE()\  )  #define FST\_ID FST(2, LEX\_ID, \  NODE(124,\  RELATION('a', 0), RELATION('a', 1),\  RELATION('b', 0), RELATION('b', 1),\  RELATION('c', 0), RELATION('c', 1),\  RELATION('d', 0), RELATION('d', 1),\  RELATION('e', 0), RELATION('e', 1),\  RELATION('f', 0), RELATION('f', 1),\  RELATION('g', 0), RELATION('g', 1),\  RELATION('h', 0), RELATION('h', 1),\  RELATION('i', 0), RELATION('i', 1),\  RELATION('j', 0), RELATION('j', 1),\  RELATION('k', 0), RELATION('k', 1),\  RELATION('l', 0), RELATION('l', 1),\  RELATION('m', 0), RELATION('m', 1),\  RELATION('n', 0), RELATION('n', 1),\  RELATION('o', 0), RELATION('o', 1),\  RELATION('s', 0), RELATION('s', 1),\  RELATION('t', 0), RELATION('t', 1),\  RELATION('u', 0), RELATION('u', 1),\  RELATION('v', 0), RELATION('v', 1),\  RELATION('w', 0), RELATION('w', 1),\  RELATION('x', 0), RELATION('x', 1),\  RELATION('y', 0), RELATION('y', 1),\  RELATION('z', 0), RELATION('z', 1),\  RELATION('A', 0), RELATION('A', 1),\ | RELATION('B', 0), RELATION('B', 1),\  RELATION('C', 0), RELATION('C', 1),\  RELATION('D', 0), RELATION('D', 1),\  RELATION('E', 0), RELATION('E', 1),\  RELATION('F', 0), RELATION('F', 1),\  RELATION('G', 0), RELATION('G', 1),\  RELATION('H', 0), RELATION('H', 1),\  RELATION('I', 0), RELATION('I', 1),\  RELATION('J', 0), RELATION('J', 1),\  RELATION('K', 0), RELATION('K', 1),\  RELATION('L', 0), RELATION('L', 1),\  RELATION('M', 0), RELATION('M', 1),\  RELATION('N', 0), RELATION('N', 1),\  RELATION('R', 0), RELATION('R', 1),\  RELATION('S', 0), RELATION('S', 1),\  RELATION('T', 0), RELATION('T', 1),\  RELATION('U', 0), RELATION('U', 1),\  RELATION('V', 0), RELATION('V', 1),\  RELATION('W', 0), RELATION('W', 1),\  RELATION('X', 0), RELATION('X', 1),\  RELATION('Y', 0), RELATION('Y', 1),\  RELATION('Z', 0), RELATION('Z', 1),\  RELATION('1', 0), RELATION('1', 1),\  RELATION('2', 0), RELATION('2', 1),\  RELATION('3', 0), RELATION('3', 1),\  RELATION('4', 0), RELATION('4', 1),\  RELATION('5', 0), RELATION('5', 1),\  RELATION('6', 0), RELATION('6', 1),\  RELATION('7', 0), RELATION('7', 1),\  RELATION('8', 0), RELATION('8', 1),\  RELATION('9', 0), RELATION('9', 1),\  RELATION('0', 0), RELATION('0', 1)\  ),\  NODE())  #define FST\_STRING\_LITERAL FST(3, LEX\_LITERAL,\  NODE(1, RELATION('\'', 1)),\  NODE(135,\  RELATION('A', 1),\  RELATION('B', 1),\  RELATION('C', 1),\  RELATION('D', 1),\  RELATION('E', 1),\  RELATION('F', 1),\  RELATION('G', 1),\  RELATION('H', 1),\  RELATION('I', 1),\  RELATION('J', 1),\  RELATION('K', 1),\  RELATION('L', 1),\  RELATION('M', 1),\  RELATION('N', 1),\  RELATION('O', 1),\  RELATION('P', 1),\  RELATION('Q', 1),\  RELATION('R', 1),\  RELATION('S', 1),\  RELATION('T', 1),\  RELATION('U', 1),\  RELATION('V', 1),\  RELATION('W', 1),\  RELATION('X', 1),\  RELATION('Y', 1),\  RELATION('Z', 1),\  RELATION('a', 1),\  RELATION('b', 1),\  RELATION('c', 1),\  RELATION('d', 1),\  RELATION('e', 1),\  RELATION('f', 1),\  RELATION('g', 1),\  RELATION('h', 1),\  RELATION('i', 1),\  RELATION('j', 1),\  RELATION('k', 1),\  RELATION('l', 1),\  RELATION('m', 1),\  RELATION('n', 1),\  RELATION('o', 1),\  RELATION('p', 1),\  RELATION('q', 1),\  RELATION('r', 1),\  RELATION('s', 1),\  RELATION('t', 1),\  RELATION('u', 1),\  RELATION('v', 1),\  RELATION('w', 1),\  RELATION('x', 1),\  RELATION('y', 1),\  RELATION('z', 1),\  RELATION('А', 1),\  RELATION('Б', 1),\  RELATION('В', 1),\  RELATION('Г', 1),\  RELATION('Д', 1),\  RELATION('Е', 1),\  RELATION('Ж', 1),\  RELATION('З', 1),\  RELATION('И', 1),\  RELATION('Й', 1),\  RELATION('К', 1),\  RELATION('Л', 1),\  RELATION('М', 1),\  RELATION('Н', 1),\  RELATION('О', 1),\  RELATION('П', 1),\  RELATION('Р', 1),\  RELATION('С', 1),\  RELATION('Т', 1),\  RELATION('У', 1),\  RELATION('Ф', 1),\  RELATION('Х', 1),\  RELATION('Ц', 1),\  RELATION('Щ', 1),\  RELATION('Ъ', 1),\  RELATION('Ы', 1),\  RELATION('Ь', 1),\  RELATION('Э', 1),\  RELATION('Ю', 1),\  RELATION('Я', 1),\  RELATION('а', 1),\  RELATION('б', 1),\  RELATION('в', 1),\  RELATION('г', 1),\  RELATION('д', 1),\  RELATION('е', 1),\  RELATION('ж', 1),\  RELATION('з', 1),\  RELATION('и', 1),\  RELATION('й', 1),\  RELATION('к', 1),\  RELATION('л', 1),\  RELATION('м', 1),\  RELATION('н', 1),\  RELATION('о', 1),\  RELATION('п', 1),\  RELATION('р', 1),\  RELATION('с', 1),\  RELATION('т', 1),\  RELATION('у', 1),\  RELATION('ф', 1),\  RELATION('х', 1),\  RELATION('ц', 1),\  RELATION('ч', 1),\  RELATION('ш', 1),\  RELATION('щ', 1),\  RELATION('ъ', 1),\  RELATION('ы', 1),\  RELATION('ь', 1),\  RELATION('э', 1),\  RELATION('ю', 1),\  RELATION('я', 1),\  RELATION('0', 1),\  RELATION('1', 1),\  RELATION('2', 1),\  RELATION('3', 1),\  RELATION('4', 1),\  RELATION('5', 1),\  RELATION('6', 1),\  RELATION('7', 1),\  RELATION('8', 1),\  RELATION('9', 1),\  RELATION(',', 1),\  RELATION('.', 1),\  RELATION('!', 1),\  RELATION('?', 1),\  RELATION('"', 1),\  RELATION('(', 1),\  RELATION(')', 1),\  RELATION(' ', 1),\  RELATION('\'', 2)\  ),\NODE()) |

Листинг A.4 – Конечные автоматы

# **ПРИЛОЖЕНИЕ Б**

|  |
| --- |
| Greibach greibach(NS('S'), TS('$'),  7,  Rule(NS('S'), GRB\_ERROR\_SERIES + 0,  3, //m{NrE;}; m{rE;}; - main функция tfi(F){NrE;}; tfi(F){NrE;};S tfi(){NrE;}; tfi(){NrE;};S - обычные функции(с параметрами и без)  Rule::Chain(8, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}')),  Rule::Chain(7, TS('m'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}')),  Rule::Chain(13, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), NS('S'))  ),  Rule(NS('N'), GRB\_ERROR\_SERIES + 1,  19, //dti; rE; i=E; i=E;N dtfi(F); dti;N rE;N i=E;N dtfi(F);N  Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),  Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),  Rule::Chain(8, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';')),  Rule::Chain(9, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';'), NS('N')),  Rule::Chain(7, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS(';')),  Rule::Chain(8, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS(';'), NS('N')),  Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(3, TS('p'), TS('i'), TS(';')),  Rule::Chain(4, TS('p'), TS('i'), TS(';'), NS('N')),  Rule::Chain(3, TS('p'), TS('l'), TS(';')),  Rule::Chain(4, TS('p'), TS('l'), TS(';'), NS('N')),  Rule::Chain(6, TS('p'), TS('i'), TS('('), NS('W'), TS(')'), TS(';')),  Rule::Chain(7, TS('p'), TS('i'), TS('('), NS('W'), TS(')'), TS(';'), NS('N')),  Rule::Chain(7, TS('T'), TS('('), NS('C'), TS(')'), TS('['), NS('N'), TS(']')),  Rule::Chain(8, TS('T'), TS('('), NS('C'), TS(')'), TS('['), NS('N'), TS(']'), NS('N')),  Rule::Chain(7, TS('e'), TS('('), NS('C'), TS(')'), TS('['), NS('N'), TS(']')),  Rule::Chain(8, TS('e'), TS('('), NS('C'), TS(')'), TS('['), NS('N'), TS(']'), NS('N')),  Rule::Chain(3, TS('r'), NS('E'), TS(';'))  ),  Rule(NS('E'), GRB\_ERROR\_SERIES + 2,  11, //i l (E) i(W) s(W) R(W) c(W) c() iM lM Mi Ml (E)M i(W)M  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('('), NS('E'), TS(')')),  Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),  Rule::Chain(3, TS('i'), TS('('), TS(')')),  Rule::Chain(4, TS('i'), TS('('), TS(')'), NS('E')),  Rule::Chain(2, TS('i'), NS('M')),  Rule::Chain(2, TS('l'), NS('M')),  Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M'))  ),  Rule(NS('M'), GRB\_ERROR\_SERIES + 3,  35, //vE vEM v(E) v(E)M  Rule::Chain(2, TS('+'), NS('E')),  Rule::Chain(4, TS('+'), TS('('), NS('E'), TS(')')),  Rule::Chain(5, TS('+'), TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(3, TS('+'), NS('E'), NS('M')),  Rule::Chain(2, TS('-'), NS('E')),  Rule::Chain(4, TS('-'), TS('('), NS('E'), TS(')')),  Rule::Chain(5, TS('-'), TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(3, TS('-'), NS('E'), NS('M')),  Rule::Chain(2, TS('\*'), NS('E')),  Rule::Chain(4, TS('\*'), TS('('), NS('E'), TS(')')),  Rule::Chain(5, TS('\*'), TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(3, TS('\*'), NS('E'), NS('M')),  Rule::Chain(2, TS('/'), NS('E')),  Rule::Chain(4, TS('/'), TS('('), NS('E'), TS(')')),  Rule::Chain(5, TS('/'), TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(3, TS('/'), NS('E'), NS('M')),  Rule::Chain(1, TS(':')),  Rule::Chain(1, TS('&')),  Rule::Chain(1, TS('|')),  Rule::Chain(2, TS('|'), NS('E')),  Rule::Chain(4, TS('|'), TS('('), NS('E'), TS(')')),  Rule::Chain(5, TS('|'), TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(3, TS('|'), NS('E'), NS('M')),  Rule::Chain(2, TS('&'), NS('E')),  Rule::Chain(4, TS('&'), TS('('), NS('E'), TS(')')),  Rule::Chain(5, TS('&'), TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(3, TS('&'), NS('E'), NS('M'))  ),  Rule(NS('F'), GRB\_ERROR\_SERIES + 4,  2, //ti ti,F  Rule::Chain(2, TS('t'), TS('i')),  Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))  ),  Rule(NS('W'), GRB\_ERROR\_SERIES + 5,  4, // i l i,W l,W  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('i'), TS(','), NS('W')),  Rule::Chain(3, TS('l'), TS(','), NS('W'))  ),  Rule(NS('C'), GRB\_ERROR\_SERIES + 6,  13, // для условий if  Rule::Chain(3, TS('i'), TS('<'), TS('i')),  Rule::Chain(3, TS('l'), TS('<'), TS('l')),  Rule::Chain(3, TS('i'), TS('<'), TS('l')),  Rule::Chain(3, TS('l'), TS('<'), TS('i')),  Rule::Chain(3, TS('i'), TS('>'), TS('i')),  Rule::Chain(3, TS('l'), TS('>'), TS('l')),  Rule::Chain(3, TS('i'), TS('>'), TS('l')),  Rule::Chain(3, TS('l'), TS('>'), TS('i')),  Rule::Chain(3, TS('i'), TS('&'), TS('i')))); |

Листинг Б.1 – Правила, описывающие грамматику языка

# **ПРИЛОЖЕНИЕ В**

|  |
| --- |
| namespace MFST {  struct MfstState {  short lenta\_position, nrule, nrulechain;  MFSTSTSTACK st;  MfstState();  MfstState(short pposition, MFSTSTSTACK pst, short pnrulechain);  MfstState(short pposition, MFSTSTSTACK pst, short pnrule, short pnrulechain);  };  struct Mfst {  enum class RC\_STEP { //! Код возврата функции step  NS\_OK, //! Найдено правило и цепочка, цепочка записана в стек  NS\_NORULE, //! Не найдено правило грамматики (ошибки в грамматике)  NS\_NORULECHAIN, //! Не найдена подходящая цепочка правила (ошибка в исходном коде)  NS\_ERROR, //! Неизвестный нетерминальный символ грамматики  TS\_OK, //! Текущий символ ленты == вершине стека, продвинулась лента, рор стека  TS\_NOK, //! Текущий символ ленты != вершине стека, восстановлено состояние  LENTA\_END, //! Текущая позиция ленты >= lenta\_size  SURPRISE //! Неожиданный код возврата ( ошибка в step)  };  struct MfstDiagnosis {  short lenta\_position, nrule, nrule\_chain;  RC\_STEP rc\_step;  MfstDiagnosis();  MfstDiagnosis(short plenta\_position, RC\_STEP prc\_step, short pnrule, short pnrule\_chain);  }diagnosis[MFST\_DIAGN\_NUMBER];  std::vector<GRBALPHABET> tape;  short lenta\_position, nrule, nrulechain, lenta\_size;  GRB::Greibach grebach;  LT::LexTable lex;  MFSTSTSTACK st;  use\_container<std::stack<MfstState>> storestate;  Mfst();  Mfst(LT::LexTable plex, GRB::Greibach pgrebach);  std::string getCSt();  std::string getCLenta(short pos, short n = 25);  std::string getDiagnosis(short n);  bool savestate(const Log::LOG& log);  bool reststate(const Log::LOG& log);  bool push\_chain(GRB::Rule::Chain chain);  RC\_STEP step(const Log::LOG& log);  bool start(const Log::LOG& log);  bool savediagnois(RC\_STEP pprc\_step);  void printrules(const Log::LOG& log);  static void clearGreibach(Mfst&);  struct Deducation {  short size;  std::vector<short> nrules, nrulechains;  Deducation() : size(0), nrules(0), nrulechains(0) { };  }deducation;  bool savededucation();}; |

Листинг B.1 – Структура магазинного конечного автомата

# **ПРИЛОЖЕНИЕ Г**

|  |
| --- |
| 0 : S->tfi(F){NrE;}S tfi(ti,ti){dti;i=i+i;ri;} S$  1 : SAVESTATE: 1  1 : tfi(ti,ti){dti;i=i+i;ri;} tfi(F){NrE;}S$  2 : fi(ti,ti){dti;i=i+i;ri;}m fi(F){NrE;}S$  3 : i(ti,ti){dti;i=i+i;ri;}m{ i(F){NrE;}S$  4 : (ti,ti){dti;i=i+i;ri;}m{d (F){NrE;}S$  5 : ti,ti){dti;i=i+i;ri;}m{dt F){NrE;}S$  6 : F->ti ti,ti){dti;i=i+i;ri;}m{dt F){NrE;}S$  7 : SAVESTATE: 2  7 : ti,ti){dti;i=i+i;ri;}m{dt ti){NrE;}S$  8 : i,ti){dti;i=i+i;ri;}m{dtf i){NrE;}S$  9 : ,ti){dti;i=i+i;ri;}m{dtfi ){NrE;}S$ |

Рисунок Г.1 – Работа синтаксического анализатора (начало разбора)

|  |
| --- |
| 2011: TS\_NOK/NS\_NORULECHAIN  2011: RESSTATE  2011: pl;]rl;} N]rE;}\_$  2012: N->pl; pl;]rl;} N]rE;}\_$  2013: SAVESTATE: 102  2013: pl;]rl;} pl;]rE;}\_$  2014: l;]rl;} l;]rE;}\_$  2015: ;]rl;} ;]rE;}\_$  2016: ]rl;} ]rE;}\_$  2017: rl;} rE;}\_$  2018: l;} E;}\_$  2019: E->l l;} E;}\_$  2020: SAVESTATE: 103  2020: l;} l;}\_$  2021: ;} ;}\_$  2022: } }\_$  2023: \_$  2024: LENTA\_END  2025: ------>LENTA\_END |

Рисунок Г.2 – Работа синтаксического анализатора (конец разбора)

|  |
| --- |
| 0 : S->tfi(F){NrE;}S  4 : F->ti,F  7 : F->ti  11 : N->dti;N  15 : N->i=E;  17 : E->iM  18 : M->+E  19 : E->i  22 : E->i  25 : S->m{NrE;}\_  27 : N->dtfi(F);N  32 : F->ti,F  35 : F->ti  39 : N->dtfi(F);N  44 : F->ti  48 : N->dti;N  52 : N->dti;N  56 : N->dti;N  60 : N->dti;N  64 : N->dti;N  68 : N->dti;N  72 : N->i=E;N  74 : E->l  76 : N->i=E;N  78 : E->l  80 : N->i=E;N  82 : E->l  84 : N->pl;N  87 : N->i=E;N  89 : E->i(W)  91 : W->i,W  93 : W->i  96 : N->pi;N  99 : N->pl;N  102 : N->i=E;N  104 : E->iM  105 : M->+E  106 : E->i  108 : N->pi;N  111 : N->pl;N  114 : N->i=E;N  116 : E->iM  118 : E->i  120 : N->pi;N  123 : N->pl;N  126 : N->i=E;N  128 : E->i(W)  130 : W->i  133 : N->pl;N  136 : N->pi;N  139 : N->pl;N  142 : N->pi;N  145 : N->pl;N  148 : N->i=E;N  150 : E->iM  151 : M->\*E  152 : E->i  154 : N->pi;N  157 : N->pl;N  160 : N->i=E;N  162 : E->iM  163 : M->&E  164 : E->i  166 : N->pi;N  169 : N->pl;N  172 : N->i=E;N  174 : E->iM  175 : M->|E  176 : E->i  178 : N->pi;N  181 : N->pl;N  184 : N->i=E;N  187 : M->:E  188 : E->l  190 : N->pi;N  193 : N->pl;N  196 : N->i=E;N  198 : E->i(W)  202 : W->i  205 : N->pi;N  208 : N->dti;N  212 : N->dti;N  216 : N->i=E;N  218 : E->l  220 : N->i=E;N  222 : E->l  224 : N->pl;N  227 : N->pi;N  230 : N->pi;N  233 : N->pl;N  236 : N->e(C)[N]N  238 : C->i  241 : N->pl;  245 : N->T(C)[N]N  247 : C->i<i  252 : N->pl;  256 : N->e(C)[N]  258 : C->i<i  263 : N->pl; |

Рисунок Г.3 – Результат работы синтаксического анализатора

# **ПРИЛОЖЕНИЕ Д**

|  |
| --- |
| bool PN::find\_elem(std::stack<char> stack, size\_t size, char elem) {  for (size\_t i = 0; i < size; i++)  if (stack.top() == elem)  return true;  else  stack.pop();  return false;  }  int PN::get\_priority(char lexem) {  std::vector<std::pair<int, char>> priority = { {0, LEX\_LEFTHESIS}, {0, LEX\_RIGHTHESIS},  {1,LEX\_COMMA},  {2,LEX\_PLUS}, {2, LEX\_MINUS},  {3,LEX\_STAR}, {3, LEX\_OR}, {3, LEX\_AND}, {3, LEX\_INVERSE}};  for (size\_t i = 0; i < priority.size(); i++)  if (lexem == priority[i].second)  return priority[i].first;  return 0;  }  void PN::fix\_lextable(LT::LexTable& lextable, const std::string& str, size\_t length, size\_t pos, const std::vector<int>& ids) {  for (size\_t i = 0, q = 0; i < str.size(); i++) {  lextable.table[pos + i].lexema = str[i];  if (lextable.table[pos + i].lexema == LEX\_ID || lextable.table[pos + i].lexema == LEX\_LITERAL) {  lextable.table[pos + i].idxTI = ids[q];  q++;  }  else  lextable.table[pos + i].idxTI = LT\_TI\_NULLIDX;  }  int temp = str.size() + pos;  for (size\_t i = 0; i < length - str.size(); i++) {  lextable.table[temp + i].idxTI = LT\_TI\_NULLIDX;  lextable.table[temp + i].lexema = '!';  lextable.table[temp + i].sn = -1;  }  }  void PN::PolishNotation(LT::LexTable& lextable, IT::IdTable& idtable) {  for (int i = 0; i < lextable.size; i++)  if (lextable.table[i].lexema == LEX\_EQUAL)  if (!Convertation(i + 1, lextable, idtable))  throw ERROR\_THROW(130);  }  bool PN::Convertation(int lextable\_pos, LT::LexTable& lextable, IT::IdTable& idtable)  {  container<std::stack<char>> stack;  std::string PolishString;  std::vector<char> operators = { LEX\_MINUS, LEX\_PLUS, LEX\_STAR, LEX\_AND, LEX\_OR, LEX\_INVERSE };  std::vector<int> ids;  int operators\_count = 0, operands\_count = 0, iterator = 0, right\_counter = 0, left\_counter = 0, params\_counter = 0;  for (int i = lextable\_pos; i < lextable.size; i++, iterator++) {  char lexem = lextable.table[i].lexema;  size\_t stack\_size = stack.size();  if (idtable.table[lextable.table[i].idxTI].idtype == IT::IDTYPE::F) {  stack.push('@');  operands\_count--;  }  if (std::find(operators.begin(), operators.end(), lexem) != operators.end()) {  if (!stack.empty() && stack.top() != LEX\_LEFTHESIS) {  while (!stack.empty() && PN::get\_priority(lexem) <= PN::get\_priority(stack.top())) {  PolishString += stack.top();  stack.pop();  }  }  stack.push(lexem);  operators\_count++;  }  else if (lexem == LEX\_COMMA) {  while (!stack.empty()) {  if (stack.top() == LEX\_LEFTHESIS)  break;  PolishString += stack.top();  stack.pop();  }  operands\_count--;  }  else if (lexem == LEX\_LEFTHESIS) {  left\_counter++;  stack.push(lexem);  }  else if (lexem == LEX\_RIGHTHESIS) {  right\_counter++;  if (!PN::find\_elem(stack, stack\_size, LEX\_LEFTHESIS))  return false;  while (stack.top() != LEX\_LEFTHESIS) {  PolishString += stack.top();  stack.pop();  }  stack.pop();  if (!stack.empty() && stack.top() == '@') {  PolishString += stack.top() + To\_string(params\_counter - 1);  params\_counter = 0;  stack.pop();  }  }  else if (lexem == LEX\_SEMICOLON) {  if (operators\_count != 0 && operands\_count != 0)  if ((!stack.empty() && (stack.top() == LEX\_RIGHTHESIS || stack.top() == LEX\_LEFTHESIS))  || right\_counter != left\_counter || operands\_count - operators\_count != 1)  return false;  while (!stack.empty()) {  PolishString += stack.top();  stack.pop();  }  PN::fix\_lextable(lextable, PolishString, iterator, lextable\_pos, ids);  break;  }  else if (lexem == LEX\_ID || lexem == LEX\_LITERAL) {  if (std::find(stack.c.begin(), stack.c.begin(), '@') != stack.c.end())  params\_counter++;  PolishString += lexem;  if (lextable.table[i].idxTI != LT\_TI\_NULLIDX)  ids.push\_back(lextable.table[i].idxTI);  operands\_count++;  }  }  return true;  } |

Листинг Д.1 – Алгоритма преобразования выражений к польской записи

# **ПРИЛОЖЕНИЕ Е**

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib ../Debug/BVA\_Lib.lib  ExitProcess PROTO : DWORD  \_ConsoleWriteInt PROTO : SDWORD  iabs PROTO : SDWORD  step PROTO : SDWORD, : SDWORD  output PROTO : SDWORD  .stack 4096  .const  l0 SDWORD 4  l1 SDWORD 6  l2 SDWORD 0  l3 BYTE 'вызов функции step', 0  l4 BYTE 'сложение 4 и 6', 0  l5 BYTE 'разность 4 и 6', 0  l6 BYTE 'вызов функции iabs', 0  l7 BYTE 'исходное с', 0  l8 BYTE 'полученное с', 0  l9 BYTE 'произведение 4 и 6', 0  l10 BYTE 'побитовое И 4 и 6', 0  l11 BYTE 'побитовое ИЛИ 4 и 6', 0  l12 BYTE 'побитовое НЕ для 4 ', 0  l13 SDWORD 1  l14 BYTE 'вызов функции sum', 0  l15 BYTE 'г', 0  l16 BYTE 'W', 0  l17 BYTE 'вывод symb и int', 0  l18 BYTE 'сработало if', 0  l19 BYTE 'сработало else', 0  .data  sumsm SDWORD 0  mainb SDWORD 0  mainc SDWORD 0  maind SDWORD 0  maina SDWORD 0  mainz SDWORD 0  mainflag SDWORD 0  mainsa SDWORD 0  mainsc SDWORD 0  .code  sum PROC b: SDWORD, a: SDWORD  push a  push b  ;\/(ADD)+\/  pop eax  pop ebx  ADD eax, ebx  push eax  ;/\(ADD)+/\  pop sumsm  mov eax, sumsm  ret  sum ENDP  main PROC  push l0  pop maina  push l1  pop mainb  push l2  pop mainflag  push offset l3  call output  push maina  push mainb  call step  push eax  pop mainc  push mainc  call \_ConsoleWriteInt  push offset l4  call output  push maina  push mainb  ;\/(ADD)+\/  pop eax  pop ebx  ADD eax, ebx  push eax  ;/\(ADD)+/\  pop mainc  push mainc  call \_ConsoleWriteInt  push offset l5  call output  push maina  push mainb  ;\/(SUB)-\/  pop ebx  pop eax  SUB eax, ebx  push eax  ;/\SUB(-)/\  pop mainc  push mainc  call \_ConsoleWriteInt  push offset l6  call output  push mainc  call iabs  push eax  pop maind  push offset l7  call output  push mainc  call \_ConsoleWriteInt  push offset l8  call output  push maind  call \_ConsoleWriteInt  push offset l9  call output  push mainb  push maina  ;\/(MUL)\*\/  pop eax  pop ebx  MUL ebx  push eax  ;/\(MUL)\*/\  pop mainc  push mainc  call \_ConsoleWriteInt  push offset l10  call output  push maina  push mainb  ;\/AND(&)\/  pop eax  pop edx  AND eax, edx  push eax  ;/\AND(&)/\  pop mainz  push mainz  call \_ConsoleWriteInt  push offset l11  call output  push maina  push mainb  ;\/(OR)|\/  pop eax  pop edx  OR eax, edx  push eax  ;/\(OR)|/\  pop mainz  push mainz  call \_ConsoleWriteInt  push offset l12  call output  push maina  push l13  ;\/(INVERSE(NOT)):\/  NOT eax  push eax  ;/\(INVERSE(NOT)):/\  pop mainz  push mainz  call \_ConsoleWriteInt  push offset l14  call output  push maina  push mainb  call sum  push eax  pop maina  push maina  call \_ConsoleWriteInt  push offset l15  pop mainsa  push offset l16  pop mainsc  push offset l17  call output  push mainsa  call output  push mainsc  call output  push l13  call \_ConsoleWriteInt  .IF mainflag  push offset l18  call output  .ELSEIF maina  push offset l19  call output  .ENDIF  .IF maina  push offset l18  call output  .ENDIF  push l2  call ExitProcess  main ENDP  end main |

Листинг E.1 – Результат генерации ассемблерного кода

# **ПРИЛОЖЕНИЕ Ж**

|  |
| --- |
| #define IN\_CODE\_TABLE {\  IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::I, IN::T, IN::F, IN::F, IN::I, IN::F, IN::F, \  IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \  IN::S, IN::T, IN::F, IN::F, IN::O, IN::O, IN::O, IN::Q, IN::O, IN::O, IN::O, IN::O, IN::O, IN::O, IN::O, IN::O, \  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::O, IN::T, IN::O, IN::T, IN::T, \  IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::O, IN::F, IN::O, IN::F, IN::F, \  IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::O, IN::O, IN::O, IN::F, IN::F, \ \  IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \  IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \  IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \  IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \  IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \  IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \  IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \  IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \  } |

Листинг Ж.1 – Таблица допустимых символов