МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 6-05-0612-01 Программная инженерия

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора SGV – 2024»

Выполнил студент Статько Герман Вячеславович

(Ф.И.О.)

Руководитель проекта асс. Гончар Е. А.

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

Заведующий кафедрой Смелов В. В.

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

Курсовой проект защищен с оценкой

Минск 2024

**Содержание**

[Введение 5](#_Toc185179803)

[1 Спецификация языка программирования 6](#_Toc185179804)

[1.1 Характеристика языка программирования 6](#_Toc185179805)

[1.2 Определение алфавит языка программирования 6](#_Toc185179806)

[1.3 Применяемые сепараторы 6](#_Toc185179807)

[1.4 Применяемые кодировки 7](#_Toc185179808)

[1.5 Типы данных 7](#_Toc185179809)

[1.6 Преобразование типов данных 8](#_Toc185179810)

[1.7 Идентификаторы 8](#_Toc185179811)

[1.8 Литералы 8](#_Toc185179812)

[1.9 Область видимости идентификаторов 9](#_Toc185179813)

[1.10 Инициализация данных 10](#_Toc185179814)

[1.11 Инструкции языка 10](#_Toc185179815)

[1.12 Операции языка 11](#_Toc185179816)

[1.13 Выражения и их вычисления 11](#_Toc185179817)

[1.14 Программные конструкции языка 12](#_Toc185179818)

[1.15 Область видимости 12](#_Toc185179819)

[1.16 Семантические проверки 12](#_Toc185179820)

[1.17 Распределение оперативной памяти на этапе выполнения 13](#_Toc185179821)

[1.18 Стандартная библиотека и её состав 13](#_Toc185179822)

[1.19 Ввод и вывод данных 13](#_Toc185179823)

[1.20 Точка входа 13](#_Toc185179824)

[1.21 Препроцессор 14](#_Toc185179825)

[1.22 Соглашения о вызовах 14](#_Toc185179826)

[1.23 Объектный код 14](#_Toc185179827)

[1.24 Классификация сообщений транслятора 14](#_Toc185179828)

[1.25 Контрольный пример 14](#_Toc185179829)

[2 Структура транслятора 15](#_Toc185179830)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc185179831)

[2.2 Перечень параметров транслятора 16](#_Toc185179832)

[3 Разработка лексического анализатора 17](#_Toc185179833)

[3.1 Структура лексического анализатора 17](#_Toc185179834)

[3.2 Входные и выходные данные лексического анализатора 18](#_Toc185179835)

[3.3 Параметры лексического анализатора 18](#_Toc185179836)

[3.4 Алгоритм лексического анализа 18](#_Toc185179837)

[3.5 Перечень ключевых слов языка 19](#_Toc185179838)

[3.5 Основные структуры данных 21](#_Toc185179839)

[3.6 Перечень сообщений лексического анализатора 22](#_Toc185179840)

[3.7 Принцип обработки ошибок 23](#_Toc185179841)

[3.8 Параметры лексического анализатора 23](#_Toc185179842)

[3.9 Алгоритм лексического анализа 23](#_Toc185179843)

[3.10 Контрольный пример 23](#_Toc185179844)

[4. Разработка синтаксического анализатора 24](#_Toc185179845)

[4.1 Структура синтаксического анализатора 24](#_Toc185179846)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 24](#_Toc185179847)

[4.3 Построение конечного магазинного автомата 26](#_Toc185179848)

[4.4 Основные структуры данных 27](#_Toc185179849)

[4.5 Описание алгоритма синтаксического анализа 27](#_Toc185179850)

[4.6 Структура и перечень сообщений синтаксического анализа 28](#_Toc185179851)

[4.7 Параметры синтаксического анализатора и режимы его работы 28](#_Toc185179852)

[4.8 Принцип обработки ошибок 29](#_Toc185179853)

[4.9 Контрольный пример 29](#_Toc185179854)

[5. Разработка семантического анализатора 30](#_Toc185179855)

[5.1 Структура семантического анализатора 30](#_Toc185179856)

[5.2 Функции семантического анализатора 30](#_Toc185179857)

[5.3 Перечень сообщений семантического анализатора 30](#_Toc185179858)

[5.4 Принцип обработки ошибок 31](#_Toc185179859)

[5.5 Контрольный пример 31](#_Toc185179860)

[6. Вычисление выражений 32](#_Toc185179861)

[6.1 Выражения, допускаемые языком 32](#_Toc185179862)

[6.2 Польская запись и примеры ее построения 32](#_Toc185179863)

[6.3 Программная реализация обработки выражений 33](#_Toc185179864)

[6.4 Контрольный пример 33](#_Toc185179865)

[7. Генерация кода 34](#_Toc185179866)

[7.1 Структура генератора кода 34](#_Toc185179867)

[7.2 Представление типов данных в оперативной памяти 34](#_Toc185179868)

[7.3 Статическая библиотека 35](#_Toc185179869)

[7.4 Особенности генерации кода 35](#_Toc185179870)

[7.5 Входные параметры генератора кода 35](#_Toc185179871)

[7.6 Контрольный пример 36](#_Toc185179872)

[8. Тестирование транслятора 37](#_Toc185179873)

[8.1 Общие положения 37](#_Toc185179874)

[8.2 Результаты тестирования 37](#_Toc185179875)

[Заключение 38](#_Toc185179876)

[Список использованных литературных источников 39](#_Toc185179877)

[ПРИЛОЖЕНИЕ А 40](#_Toc185179878)

[ПРИЛОЖЕНИЕ Б 42](#_Toc185179879)

[ПРИЛОЖЕНИЕ В 55](#_Toc185179880)

[ПРИЛОЖЕНИЕ Г 60](#_Toc185179881)

[ПРИЛОЖЕНИЕ Д 62](#_Toc185179882)

[ПРИЛОЖЕНИЕ Е 63](#_Toc185179883)

[ПРИЛОЖЕНИЕ Ж 67](#_Toc185179884)

# **Введение**

Задачей данного курсового проекта является разработка транслятора для своего языка программирования: SGV-2024.

Написание транслятора будет осуществляться на языке C++, при этом код на языке SGV-2024 будет транслироваться в язык Assembler.

Транслятор SGV-2024 состоит из следующих частей:

– семантический анализатор;

– синтаксический анализатор;

– лексический анализатор;

– генератор исходного кода на языке Assembler.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разбработка структуры транслятора;

– разработка лексический и семантического анализаторов;

– разработка синтаксического анализатора;

– преобразование выражений;

– генерация кода на язык Assembler;

– тестирование транслятора.

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

# **1 Спецификация языка программирования**

## **Характеристика языка программирования**

Язык программирования SGV-2024 – это универсальный язык высокого уровня. Он является процедурным, компилируемым, не объектно-ориентированным. Язык строго типизируемый, что говорит о невозможности преобразования типов, транслируемым языком программирования.

## **Определение алфавит языка программирования**

Совокупность символов, используемых в языке, называется алфавитом языка.

На этапе выполнения могут использоваться символы латинского алфавита и кириллица для символьных типов, цифры десятичной системы счисления от 0 до 9, спецсимволы, а также символы пробела.

## **Применяемые сепараторы**

Символы-сепараторы служат в качестве разделителей операций языка. Сепараторы, используемые в языке программирования SGV-2024, приведены в таблице 1.1.

|  |  |  |
| --- | --- | --- |
| Сепаратор | Название | Область применения |
| ‘ ‘ | Пробел | Допускается везде, кроме идентификаторов и ключевых слов |
| ; | Точка с запятой | Разделение конструкций |
| {…} | Фигурные скобки | Заключение программного блока |
| (…) | Круглые скобки | Приоритет операций, параметры функции, параметры условий |
| ‘…’ | Одинарные кавычки | Строковый и символьный литералы |
| = | Знак «равно» | Присваивание значения |
| , | Запятая | Разделение параметров |
| /  +  -  \* | Знаки «косая черта», «плюс», «минсу», «звезда», | Выражения |
| Сепаратор | Название | Область применения |
| ==  !=  <=  >=  >  < | «двойное равно», «не равно», «меньше или равно», «больше или равно», знаки «больше» и «меньше» | Выражения в условном блоке |

Таблица 1.1 – Сепараторы

## **Применяемые кодировки**

Для написания исходного кода на языке программирования SGV-2024 используется кодировка Windows-1251. Таблица показана на рисунке 1.2.

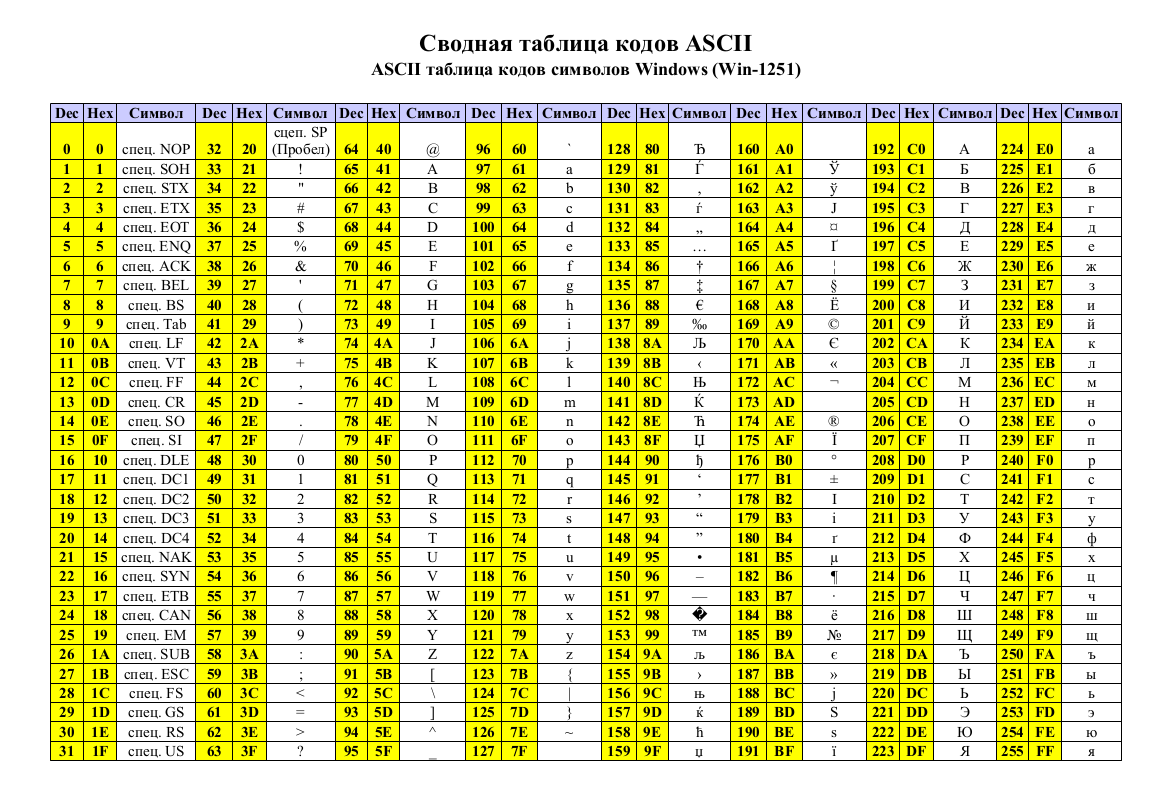


Рисунок 1.2 – Таблица W-1251

## **Типы данных**

В языке SGV-2024 реализованы четыре типа данных. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.3.

Таблица 1.3 – Типы данных языка SGV-2024

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Целочисленный тип данных int | Фундаментальный тип данных. Используется для работы с целочисленными положительными значениями.  При попытке инициализации значением больше максимального, инициализируется максимальным. При попытке инициализации значением меньше минимального, инициализируется минимальным. В памяти занимает 4 байта.  Максимальное значение: 255. Минимальное значение: 0.  Инициализация по умолчанию: значение 0. |

Продолжение таблицы 1.3

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Символьный тип данных char | Фундаментальный тип данных. Используется для работы с символом. В памяти занимает 1 байт. Инициализация по умолчанию: 0x00. |
| Строковый тип данных string | Фундаментальный тип данных. Используется для работы со строками символов. В памяти занимает до 12 байт.  По умолчанию инициализируется пустой строкой. |
| Логический тип bool | Фундаментальный тип данных. Используется для работы с логическими выражениями. В памяти занимает 1 байт.  По умолчанию инициализируется false. |

Пользовательские типы данных не поддерживаются.

## **Преобразование типов данных**

Преобразование типов данных не поддерживается, так как язык является строго типизированным.

## **Идентификаторы**

В имени идентификатора допускаются только символы латинского алфавита и знак «\_». Максимальная длина имени идентификатора – 8 символов. Максимальная длина имени идентификатора функции – 5 символов. При вводе идентификатора длиной более разрешенного количества символов, транслятор выдает ошибку. Имя идентификатора не может совпадать с именем функции, уже содержащаяся в стандартной библиотеке.

## **Литералы**

С помощью литералов осуществляется инициализация переменных. В языке существует три типа литералов. Краткое описание литералов языка SGV-2024 представлено в таблице 1.4

Таблица 1.4 – Краткое описание литералов языка

|  |  |  |  |
| --- | --- | --- | --- |
| Тип литерала | Регулярное выражение | Описание | Пример |
| Целочисленный литерал | [1-9]+[0-9]\* | Целочисленные литералы. | declare integer sum = 123;  123 – целочисленный литерал. |
| Символьный литерал | [a-z|A-Z|А-Я|а-я|0-9|!-/] | Символ, заключённый в ‘’ одинарные кавычки. | declare char symbol = ‘Л’;  Л – символьный литерал. |

Продолжение таблицы 1.4

|  |  |  |  |
| --- | --- | --- | --- |
| Тип литерала | Регулярное выражение | Описание | Пример |
| Строковый литерал | [a-z|A-Z|А-Я|а-я|0-9|!-/]\* | Строковые литералы, максимальная длина строки 11 символов | \_printStr(‘Hello world’) |
| Логический литерал | [true+false] | Логический литерал | declare bool k = true;  true – логический литерал. |

Литералы являются константами и при генерации кода объявляются один раз.

## **Область видимости идентификаторов**

Область видимости переменных в языке программирования SGV-2024 построена по принципу «сверху вниз», аналогично тому, как это реализовано в языке С++. Это означает, что видимость переменной ограничивается текущим программным блоком, а доступ к переменным возможен только после их объявления. Для работы с переменными в SGV-2024 существует строгое требование: перед инициализацией и дальнейшим использованием переменная обязательно должна быть предварительно объявлена.

Все переменные должны быть размещены внутри определенного программного блока, что исключает возможность их глобального использования за пределами этих блоков. Кроме того, допускается создание переменных с одинаковыми именами в различных программных блоках. Это возможно благодаря тому, что переменные, объявленные в одном блоке, недоступны за его пределами, включая другие функции. Для идентификации переменных каждой из них автоматически присваивается уникальный префикс, содержащий название функции, в которой она была объявлена. Таким образом, это упрощает работу с локальными переменными и снижает вероятность конфликта имен.

Функции стандартной библиотеки языка SGV-2024 можно объявлять в произвольных частях кода. Это дает разработчику большую гибкость в структурировании программ и организации их кода.

## **Инициализация данных**

При объявлении переменной не допускается инициализация. Описание способов инициализации переменных языка SGV-2024 представлено в таблице 1.5.

Таблица 1.5 - Способы инициализации переменных

|  |  |  |
| --- | --- | --- |
| Конструкция | Описание | Пример |
| declare <тип данных> <идентификатор>; | Объявление переменной. | declare int num;  declare char symb; |
| <идентификатор> = <значение>; | Присваивание переменной значения. | num = 12345;  symb = ‘Л’; |

Соответствие типов проверяется на семантическом анализе.

## **Инструкции языка**

Все инструкции языка SGV-2022 представлены в общем виде в таблице 1.6.

Таблица 1.6 – Инструкции языка программирования SGV-2024

|  |  |
| --- | --- |
| Инструкция | Запись на языке SGV-2024 |
| Объявление переменной | declare <тип данных> <идентификатор>; |
| Объявление функции | <тип данных> function <идентификатор> (<тип данных> <идентификатор>, …) {<блок кода>}; |
| Присваивание | <идентификатор> = <значение>/<идентификатор>; |
| Блок инструкций | main  {  …  } |
| Возврат из подпрограммы | return <идентификатор> / <литерал>; |
| Вывод данных | \_printStr(<идентификатор> / <литерал>);  \_printInt(<идентификатор> / <литерал>);  \_printBool(<идентификатор> / <литерал>);  \_printChar(<идентификатор> / <литерал>); |
| Условный оператор | if(<условие>)  {  <блок кода>;  }  else  {  <блок кода>;  }; |

Инструкции (кроме функции входа в программу) требуют закрывающую «;».

* 1. **Операции языка**

Язык программирования SGV-2024 может выполнять операции, представленные в таблице 1.7.

Таблица 1.7 – Операции языка программирования SGV-2024

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Примечание | Типы данных | Пример |
| ( | Приоритет операций | - | num = (x \* (x - y))\*y |
| ) |
| + | Суммирование | (int, int) | num = x + y; |
| - | Вычитание | (int, int) | num = x – y; |
| \* | Умножение | (int, int) | num = x \* y; |
| : | Деление | (int, int) | num = x / y; |
| = | Присваивание | (int, int)  (char, char)  (string, string)  (bool, bool) | num = 12345;  symb = ‘Л’; |
| ==, != | Знаки «равенство», «неравенство» для условной инструкции | (int, int) | if(x == y) {…}; |
| <, > | Операторы «меньше», «больше» для условной инструкции | (int, int) | if(x < y) {…} |
| <=, >= | Операторы «меньше или равно», «больше или равно» для условной инструкции | (int, int) | if(x <= y) {…}; |

Т.к. отрицательные числа не поддерживаются, если результат операции меньше нуля, он ставится в 0.

## **Выражения и их вычисления**

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

## **Программные конструкции языка**

Ключевые программные конструкции языка программирования SGV-2024 представлены в таблице 1.8.

Таблица 1.8 – Программные конструкции языка SGV-2024

|  |  |
| --- | --- |
| Конструкция | Запись на языке SGV-2024 |
| Главная функция (точка входа) | main  {  …  return <идентификатор> / <литерал>;  } |
| Функция | <тип данных> function <идентификатор> (<тип> <идентификатор>, …)  {… return <идентификатор> / <литерал>; }; |
| Условный оператор | if(5 > 4){…}; |

Программные конструкции языка SGV-2024 представляют собой базовый функционал для выполнения различных операций, что делает возможным решать задачи различного уровня.

## **Область видимости**

В языке SGV-2024 все переменные являются локальными, т.е. имеют функциональную область видимости. Они обязаны находится внутри программного блока функций (по принципу С++). Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

## **Семантические проверки**

Проверка смысловой правильности конструкций языка программирования. Таблица проверок, предусмотренных языком, приведена в таблице 1.9.

Таблица 1.9 - 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. |
| 4 | В функцию должно быть передано то число параметров, сколько ожидается |

Продолжение таблицы 1.9

|  |  |
| --- | --- |
| Номер | Правило |
| 5 | Тип данных результата выражения должен совпадать с типом данных идентификатора, которому оно присваивается |
| 6 | Типы данных операндов выражения должны быть одинаковыми |
| 7 | В программе должен присутствовать main |

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

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

Все переменные размещаются в куче.

## **Стандартная библиотека и её состав**

Стандартная библиотека SGV-2024 написана на языке программирования C++. Функции стандартной библиотеки с описанием представлены в таблице 1.10.

|  |  |  |
| --- | --- | --- |
| Функция(C++) | Возвращаемое значение | Описание |
| int \_pow(int, int) | int | Возвращает число в степени |
| int \_exp(int) | int | Возвращает округленное значение экспоненты в степени |
| void \_printInt(int) | Нет | Вывод значение int в консоль |
| void \_printStr(string) | Нет | Вывод значение string или char в консоль |
| void \_printBool(bool) | Нет | Вывод значение bool в консоль |
| void \_printChar(char) | Нет | Вывод значение char в консоль |
| void \_pause() | Нет | Приостанавливает выполнение программы |

Таблица 1.10 – Функции стандартной библиотеки

## **Ввод и вывод данных**

В языке SGV-2024 не реализованы средства ввода данных.

Для вывода данных в стандартный поток вывода предусмотрены такие функции как: \_printStr, \_printBool, \_printInt, \_printChar.

## **Точка входа**

В языке SGV каждая программа должна содержать главную функцию main, т. е. точку входа, с которой начнется последовательное выполнение программы.

## **Препроцессор**

Препроцессор в языке программирования SGV-2024 не предусмотрен.

## **Соглашения о вызовах**

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

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## **Объектный код**

SGV-2024 транслируется в язык Assembler.

## **Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке SGV-202 и выявления её транслятором в текущий файл протокола выводится сообщение. Классификация сообщений приведена в таблице 1.10.

Таблица 1.10. – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-89 | Системные ошибки |
| 90-99 | Ошибки параметров |
| 100-599 | Ошибки лексического анализа |
| 600-699 | Ошибки синтаксического анализа |
| 700-800 | Ошибки семантического анализа |

Компилятор может обрабатывать до 1000 различных ошибок.

**1.25 Контрольный пример**

Контрольный пример демонстрирует главные особенности языка SGV-2024: его фундаментальные типы, основные структуры, функции, использование функций статической библиотеки. Исходный код контрольного примера представлен в приложении А.

# **2 Структура транслятора**

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

В языке SGV-2024 исходный код преобразуется в ассемблер. Этот процесс выполняется транслятором, который состоит из нескольких взаимосвязанных модулей, каждый из которых выполняет определённые функции, описанные в разделе 2.1. Для генерации ассемблерного кода транслятор использует результаты работы лексического анализатора – таблицу лексем и таблицу идентификаторов. Входные параметры транслятора, определяющие выходные файлы, приведены в таблице 2.2. Структура транслятора языка SGV-2024 представлена на рисунке 2.1.

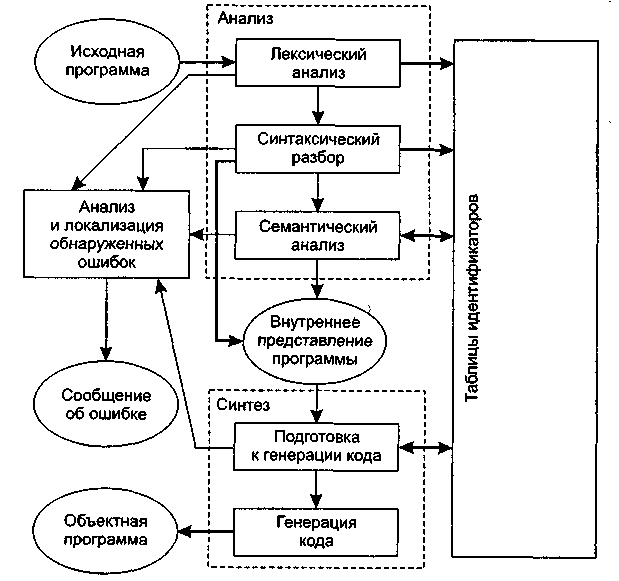


Рисунок 2.1 – Структура транслятора

Компилятор начинается свою работу с этапа лексического анализа, выполняемого специальной программой – лексическим анализатором (или сканером). Эта программа принимает на вход последовательность символов исходного языка и преобразует её в массив токенов (лексических единиц). Лексический анализатор заменяет исходные текстовые элементы их внутренними представлениями – лексемами, формируя промежуточное представление программы. Каждой лексеме присваивается тип и создаётся запись в таблице идентификаторов, которая содержит дополнительную информацию. Таблицы лексем (ТЛ), операторов (ТО) и идентификаторов (ТИ) передаются на следующий этап компиляции – синтаксический анализ.

Задачи лексического анализатора:

– удаление лишних пробелов;

– идентификация лексем;

– создание таблиц лексем, идентификаторов и операторов;

– вывод сообщений об ошибках при невозможности распознать лексемы или при обнаружении ошибок в тексте.

Синтаксический анализатор, следующая часть компилятора, проверяет соответствие исходного кода правилам грамматики. Его входными данными служат таблицы лексем и идентификаторов, а результатом работы становится дерево разбора.

Семантический анализатор, ещё один компонент транслятора, отвечает за выявление ошибок, которые невозможно обнаружить через регулярные или контекстно-свободные грамматики. Для анализа он использует таблицы лексем и идентификаторов.

Генератор кода завершает процесс трансляции, преобразуя обработанные данные в ассемблерный код. Его входными данными также являются таблицы лексем и идентификаторов, а результатом становится файл с ассемблерным кодом.

## **2.2 Перечень параметров транслятора**

Для работы с транслятором используются входные параметры, которые приведены в таблице 2.2.

Таблица 2.2 – Входные параметры языка SGV-2024

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке SGV-2024, имеющий расширение .txt | Не предусмотрено, обязательно к заполнению |
| -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 | Значение по умолчанию:  <имя in-файла>.out |

# **3 Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Первая стадия работы компилятора называется лексическим анализом, а программа, осуществляющая этот процесс, — лексическим анализатором (или сканером). На вход лексическому анализатору подаётся исходный код программы. Этот анализатор выделяет в тексте простейшие языковые конструкции и выполняет предварительную обработку, преобразуя последовательность символов в последовательность токенов.

Лексические единицы могут включать идентификаторы, числа, операционные символы, ключевые слова и другие элементы. Лексический анализатор заменяет эти элементы их внутренними представлениями, называемыми лексемами, для дальнейшей обработки. Каждой лексеме присваивается определённый тип, а также записывается информация в таблице идентификаторов, которая содержит дополнительные данные о каждом элементе.

Основные функции лексического анализатора:

– Удаление «пустых» символов и комментариев. Убирая пробелы, символы табуляции и перехода на новую строку, а также комментарии, лексический анализатор облегчает работу синтаксического анализатора, который не будет сталкиваться с этими элементами. Альтернативный способ — включение таких символов в грамматику, но он требует сложной реализации.

– Распознавание идентификаторов и ключевых слов.

– Распознавание констант.

– Распознавание разделителей и операционных знаков.

Исходный код программы представлен в приложении А, структура лексического анализатора представлена на рисунке 3.1.

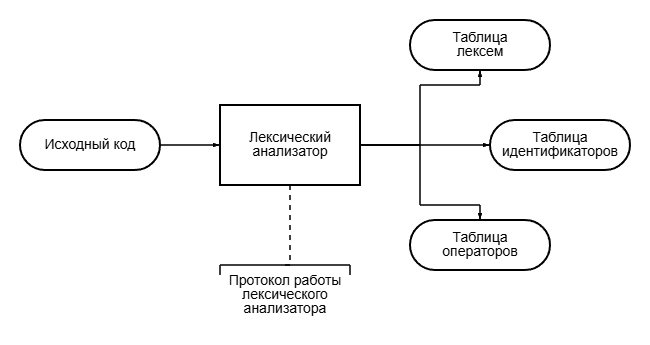


Рисунок 3.1 Структура лексического анализатора

## **3.2 Входные данные лексического анализатора**

На вход лексического анализатора подаётся исходный код входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка и производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив токенов.

В лексическом анализаторе есть таблица контроля входных символов, которая представлена на рисунке 3.2.

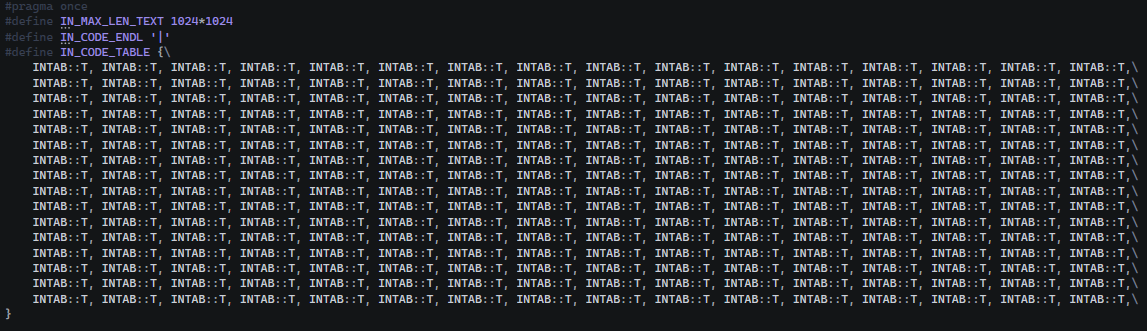


Рисунок 3.2 – Таблица контроля входных символов

Соответствие символов и их значений представлены в таблице 3.3.

Таблица 3.3 – Соответствие символов и их значений

|  |  |
| --- | --- |
| Символы | Значение в таблице входных символов |
| T | Разрешенный |
| F | Запрещенный |
| I | Игнорируется |

## **3.3 Параметры лексического анализатора**

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся как в файл журнала, так и в командную строку.

## **3.4 Алгоритм лексического анализа**

– Проверяет входной поток символов программы на допустимость, удаляет лишние пробелы и добавляет разделители для вычисления номеров строк каждой лексемы.

– Для выделенной части входного потока выполняет функцию распознавания лексемы.

– При успешном распознавании информация о лексеме заносится в таблицы лексем и идентификаторов, после чего алгоритм возвращается к первому этапу.

Формирует протокол работы.

– В случае неуспешного распознавания выводит сообщение об ошибке.

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «**bool**» представлен на рисунке 3.4, где S0 – начальное, а S4 – конечное состояние автомата.

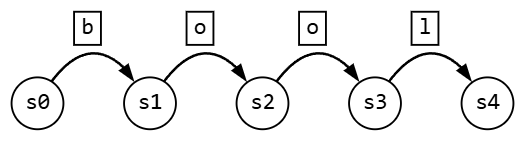


Рисунок 3.4 - Пример графа переходов для цепочки bool

## **3.5 Перечень ключевых слов языка**

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие ключевых слов и лексем приведено в таблице 3.5.

Таблица 3.5 Соответствие лексем и цепочек

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Ключевые слова | declare | d |
| integer, string, bool, char | t |
| main | m |
| function | f |
| return | r |
| if | s |
| else | e |
| Иное | Идентификатор | i |
| Литерал | l |
| Функции стандартной библиотеки | \_exp | i |
| \_pow |
| \_printStr |
| \_printBool |
| \_printInt |
| \_printChar |

Продолжение таблицы 3.5

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| Операторы | Арифметические (+, -, \*, /) | v |
| Логические (== != > < >= <=) | c |
| Присваивание (=) | = |

Каждое выражение соответствует определённый детерминированный конечный автомат, с помощью которого осуществляется разбор данного выражения. При успешном разборе выражение заносится в таблицу лексем. Если выражение является идентификатором или литералом, соответствующая информация дополнительно добавляется в таблицу идентификаторов. На рисунках 3.6 и 3.7 представлены структура конечного автомата и пример его графа переходов соответственно.



Рисунок 3.6 – Структура конечного автомата

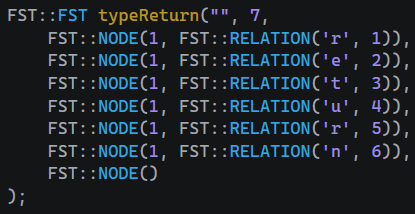


Рисунок 3.7 – Цепочка конечного автомата для слова return

В приложении Б представлены графы конечных автоматов для других слов языка.

## **3.6 Основные структуры данных**

Основными структурами данных для лексического анализатора являются таблица лексем, таблица идентификаторов и таблица операторов. Код на языке C++ со структурой таблицы идентификаторов представлен на рисунке 3.8. Код на языке C++ со структурой таблицы лексем представлен на рисунке 3.9. Код на языке C++ со структурой таблицы операторов представлен на рисунке 3.10.

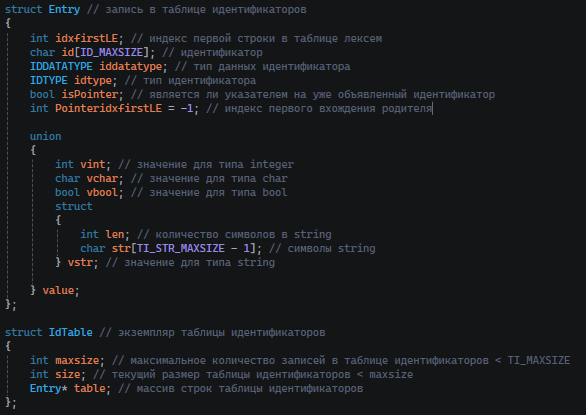


Рисунок 3.8 – Структура таблицы идентификаторов

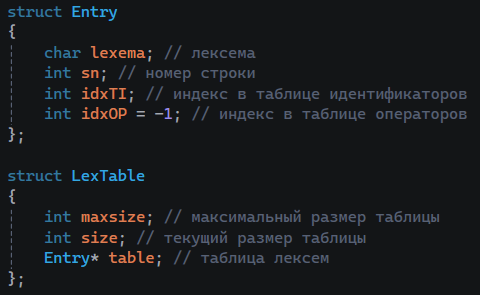


Рисунок 3.9 – Структура таблицы лексем

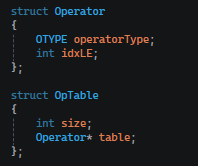


Рисунок 3.10 – Структура таблицы операторов

## **3.7 Перечень сообщений лексического анализатора**

Перечень сообщений лексического анализатора представлен на рисунке 3.11.

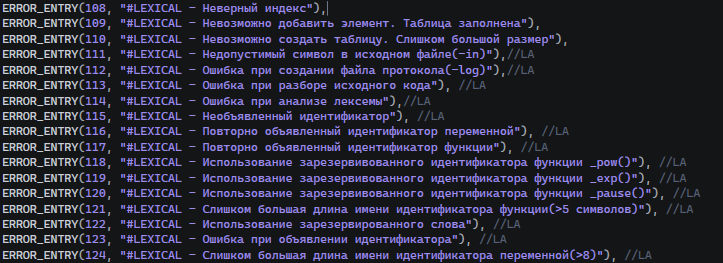
****

Рисунок 3.11 – Сообщения лексического анализатора

## **3.8 Принцип обработки ошибок**

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. При возникновении сообщения, лексический анализатор останавливает работу и далее сообщение выводится в файл протокола.

## **3.9 Параметры лексического анализатора**

Входными параметрами для лексического анализатора является исходный текст программы, написанный на языке SGV-2024, а также файл протокола в который записываются выходные данные (таблица лексем, таблица идентификаторов и таблица операций).

## **3.10 Алгоритм лексического анализа**

Лексический анализ основан на применении конечных автоматов, которые обрабатывают регулярные выражения. Регулярные выражения представляют собой формальный способ описания регулярных языков. Они состоят из констант и операторов, задающих наборы строк и операции над ними, и могут быть представлены в виде графов.

Процесс лексического анализа включает следующие этапы:

− Из входного потока символов программы на исходном языке удаляются лишние пробелы, а также добавляется разделитель, позволяющий определить номер строки для каждой лексемы.

− Создается массив, содержащий слова языка.

− Для каждого слова выполняется процедура распознавания лексемы.

− Если лексема успешно распознана, информация о ней заносится в таблицу лексем и таблицу идентификаторов, после чего процесс возвращается к первому этапу.

− В случае, если лексема является операцией, информация о ней также заносится в таблицу операций.

− В случае ошибки распознавания выводится сообщение об ошибке.

− Формируется отчет о выполненной работе.

## **3.11 Контрольный пример**

Результат работы лексического анализатора в виде таблиц лексем, операций и идентификаторов, соответствующих контрольному примеру, представлен в приложении В.

# **4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализатор — это модуль компилятора, который проверяет, соответствует ли исходный код правилам грамматики. На входе он использует таблицу лексем, созданную лексическим анализатором, а результатом его работы становится дерево разбора.

Структура синтаксического анализатора представлена на рисунке 4.1.

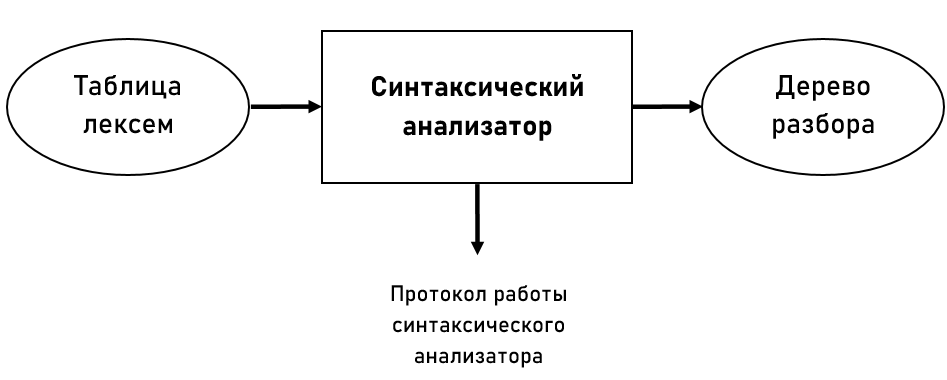
****

Рисунок 4.1 – Структура синтаксического анализатора

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

В синтаксическом анализаторе транслятора языка SGV-2024 используется контекстно-свободная грамматика , где

T – множество терминальных символов,

N – множество нетерминальных символов,

P – множество правил языка,

S – начальный символ грамматики, являющийся нетерминалом.

Грамматика представлена в нормальной форме Грейбах, что исключает наличие леворекурсивных правил. Структура правил из множества PPP строго подчиняется следующим условиям:

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

Такой подход гарантирует строгую структуру и упрощает обработку грамматики в анализаторе.

Правила языка SGV-2024 реализованные на языке C++ представлены в приложении Г.

Перечень правил, составляющих грамматику языка представлен в таблице 4.2.

Таблица 4.2 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | m{NrE;};  m{rE;};  tfi(F){NrE;};S  tfi(){NrE;};S  m{NrE;};S  tfi(F){NrE;};  tfi(){NrE;}; | Правильность структуры программы |
| N | dti;  dti=E;N  i=E;  tfi(F);  dti;N  rE;N  i=E;N  i=E;  iE;N  iE;  E;N  s(KcK){N};  s(KcK){N};N  s(KcK){N}e{N};N  s(KcK){N}e{N};  i();  i();N | Проверка операторов программы |
| E | i  l  iM  lM  (E)  i(W)  (E)M  i(W)M  i() | Проверка выражений программы |

Продолжение таблицы 4.2

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| W | i  l  i,W  l,W | Проверка подвыражений |
| F | ti  ti,F | Проверка параметров функции |
| M | vE  vEM | Проверка арифметических операций |
| K | i  l | Ограничение для if else |

## **4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку, описание которой представлено в таблице 4.3.

Таблица 4.3. – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонент | 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 представляет из себя множества терминальных и нетерминальных символов. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $). |
|  | Функция переходов автомата | Функция которая представляет из себя множество правил грамматики. |

Продолжение таблицы 4.3

|  |  |  |
| --- | --- | --- |
| Компонент | Определение | Описание |
|  | 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**

Основными структурами данных синтаксического анализатора являются структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка SGV-2024.

Данные структуры представлены в приложении Д.

## **4.5 Описание алгоритма синтаксического анализа**

Принцип конечного автомата синтаксического анализа:

– В магазин записывается стартовый символ, который будет служить отправной точкой для дальнейшей работы автомата.

– Затем на основе заранее подготовленных таблиц формируется входная лента, которая будет содержать символы для взаимодействия.

– После этого начинается работа автомата — он запускается и начинает процесс синтаксического анализа.

– На каждом шаге выбирается цепочка, соответствующая текущему нетерминальному символу, и эта цепочка записывается в магазин в обратном порядке. Таким образом, происходит дальнейшее развертывание грамматики.

– Если символы, находящиеся на верхушке стека и на ленте, совпадают, то они удаляются как из ленты, так и из стека. Если же символы не совпадают, автомат возвращается в предыдущее состояние, которое было сохранено ранее, и выбирает другую подходящую цепочку.

– В случае, если на верхушке стека обнаруживается нетерминал, то процесс возвращается к выбору новой цепочки.

– Если в процессе работы стек оказывается пустым, а лента также исчерпана, то это означает, что синтаксический анализ завершён успешно. В противном случае, если стек не пуст, либо лента ещё содержит символы, генерируется исключение, указывающее на ошибку в анализе.

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

Перечень сообщений синтаксического анализатора представлен на рисунке 4.4.

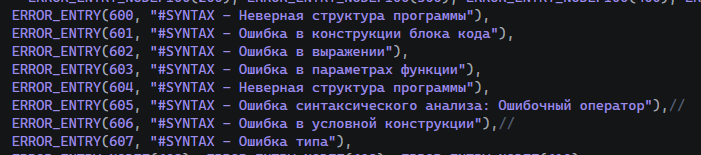


Рисунок 4.4 – Перечень сообщений синтаксического анализатора

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

Входными параметрами для синтаксического анализатора являются несколько ключевых элементов.

– Во-первых, это таблица лексем, которая была сформирована на этапе лексического анализа и содержит информацию о лексемах, встреченных в исходном тексте.

– Во-вторых, анализатор использует протокол работы, который описывает последовательность действий и условий для выполнения анализа.

– Третьим входным параметром являются правила контекстно-свободной грамматики, представленные в нормальной форме Грейбах, что упрощает процесс синтаксической обработки.

В процессе работы синтаксический анализатор генерирует два основных выходных параметра. Первый — это трассировка, которая представляет собой последовательное отображение прохода по таблице лексем, что позволяет проследить весь процесс анализа. Второй — дерево разбора, которое представляет собой структуру, отражающую синтаксическую структуру исходного текста согласно правилам грамматики. Оба результата записываются в файл протокола.

## **4.8 Принцип обработки ошибок**

Процесс обработки ошибок осуществляется следующим образом:

– Синтаксический анализатор анализирует все правила и возможные цепочки грамматических правил, чтобы найти подходящее соответствие с конструкцией, представленной в таблице лексем.

– Если не удаётся найти подходящую цепочку, генерируется ошибка.

– Все ошибки собираются в единую структуру ошибок.

– В случае обнаружения ошибки, после завершения процедуры трассировки, в протокол выводится диагностическое сообщение.

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода предоставлен в приложении Е в виде фрагмента трассировки и дерева разбора исходного кода.

# **5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализатор принимает на свой вход результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов и операторов. Некоторые проверки (такие как проверка на предварительное объявление переменной) осуществляются в процессе лексического анализа. Общая структура семантического анализатора представлена на рисунке 5.1.

****

Рисунок 5.1 – Структура семантического анализатора

## **5.2 Функции семантического анализатора**

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

За семантический анализ отвечает функция SemAnalize. Ее входными параметрами является таблицы лексем, идентификаторов и операторов.

## **5.3 Перечень сообщений семантического анализатора**

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

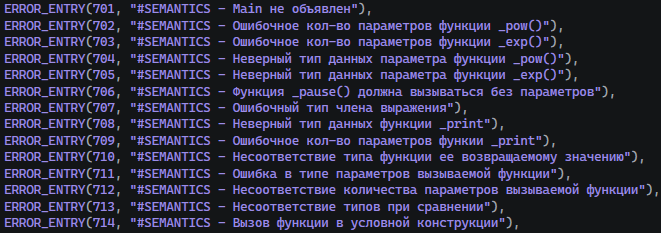


Рисунок 5.2 – Перечень сообщений семантического анализатора

## **5.4 Принцип обработки ошибок**

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибки происходит её протоколирование с номером ошибки и диагностическим сообщением; семантический анализ останавливается.

## **5.5 Контрольный пример**

Соответствие примеров некоторых семантических ошибок в исходном коде и диагностических сообщений об ошибках приведено в таблице 5.3.

Таблица 5.3 Соответствие примеров ошибок

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| z = \_exp(3, 7) + 5; | Ошибка 703: #SEMANTICS - Ошибочное кол-во параметров функции \_exp(), строка 56, позиция 207 |
| \_printInt(z, 7); | Ошибка 709: #SEMANTICS - Ошибочное кол-во параметров функии \_print, строка 57, позиция 214 |
| if(5 >= 'hello')  {  x = y - 1;  } | Ошибка 713: #SEMANTICS - Несоответствие типов при сравнении, строка 47, позиция 179 |
| y = 1 + 2 \* '5'; | Ошибка 707: #SEMANTICS - Ошибочный тип члена выражения, строка 46, позиция 169 |
| string function sayhi(){  \_printStr('hello');  return 'done';  };  main{  sayhi('test');  } | Ошибка 711: #SEMANTICS - Ошибка в типе параметров вызываемой функции, строка 44, позиция 160 |
| string function sa(integer k){  return k;  };  main{  sa();  } | Ошибка 712: #SEMANTICS - Несоответствие количества параметров вызываемой функции, строка 55, позиция 199 |

# **6. Вычисление выражений**

## **6.1 Выражения, допускаемые языком**

В языке SGV-2024 допускается вычисление выражений целочисленного типа, также поддерживается вызов функций внутри арифметических выражений. В выражениях могут использоваться арифметические операции. Операции и их приоритет представлен в таблице 6.1.

Таблица 6.1 – Операции и их приоритет

|  |  |  |
| --- | --- | --- |
| Тип операций | Операции | Приоритет |
| Арифметические операции | + – сложение  - – вычитание  \* – умножение  / – деление | 1  1  2  2 |

Примеры выражений языка SGV-2024 представлены на рисунке 6.2.

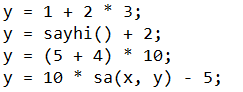


Рисунок 6.2 – Примеры выражений

## **6.2 Польская запись и примеры ее построения**

Выражения в языке SGV-2024 преобразовываются к обратной польской записи.

Обратная польская запись — это форма записи математических выражений, в которой операторы расположены после своих операндов. Выражение в обратной польской нотации читается слева направо: операция выполняется над двумя операндами, непосредственно стоящими перед знаком этой операции.

Алгоритм построения:

– исходная строка: выражение;

– результирующая строка: польская запись;

– стек: пустой;

– результирующая строка: польская запись;

– исходная строка просматривается слева направо;

– операнды переносятся в результирующую строку в порядке их следования;

– операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;

– операция выталкивает все операции с большим или равным приоритетом в результирующую строку;

– запятая не помещается в стек, если в стеке операции, то все выбираются в строку;

– отрывающая скобка помещается в стек;

– закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;

– закрывающая квадратная скобка выталкивает все до открывающей и генерирует @ – специальный символ, в которого записывается информация о вызываемой функции, а в поле приоритета для данной лексемы записывается число параметров вызываемой функции;

– по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

В таблице 6.3 представлен пример преобразования выражения в обратную польскую запись.

Таблица 6.3 – Преобразование выражения в польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| (i(i,i)vi)vi | - | - |
| (i,i)vi)vi | i | ( |
| vi)vi | iii@2 | - |
| i)vi | iii@2 | v |
| vi | iii@2iv | - |
| i | iii@2iv | v |
| - | iii@2iviv | - |

## **6.3 Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений в обратную польскую запись основа на функциях PolishNotation, CheckPriority и isSeparator.

PolishNotation принимает в себя как параметры таблицы лексем, идентификаторов и операций. Далее, с помощью функций CheckPriority и isSeparator выставляется приоритет операций и проверяются символы на сепараторы соответственно.

## **6.4 Контрольный пример**

В приложении В, в таблице лексем, приведено представление промежуточного кода, отображающее результаты преобразования выражений в польский формат.

# **7. Генерация кода**

## **7.1 Структура генератора кода**

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы в цепочку символов выходного языка. На вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом. Структура генератора кода представлена на рисунке 7.1.

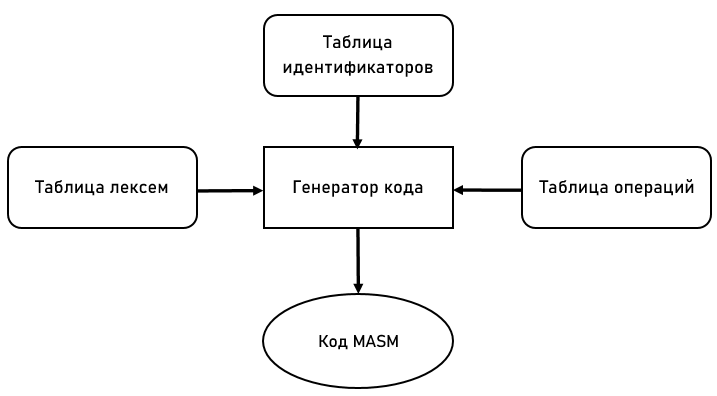
****

Рисунок 7.1 – Структура генератора кода

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

Элементы таблицы идентификаторов расположены сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке SGV-2024 и на языке ассемблера приведены в таблице 7.2.

Таблица 7.2 – Соответствия типов идентификаторов языка SGV-2024 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке SGV-2024 | Тип идентификатора на языке ассемблера | Пояснение |
| integer | dword | Хранит целочисленный тип данных. |
| bool | dword | Хранит булевый тип данных (в виде целого числа) |
| string | byte | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |
| char | dword | Хранит символьный тип данных. |

## **7.3 Статическая библиотека**

В языке RNA-2022 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Подключение библиотеки происходит с помощью includelib на этапе генерации кода. C помощью оператора EXTRN объявляются функции из библиотеки. Пример подключения библиотеки в исходном коде на языке ассемблера представлен на рисунке 7.3.

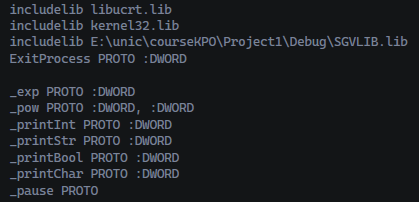


Рисунок 7.3 – Подключение статической библиотеки

## **7.4 Особенности генерации кода**

В таблице 7.4 представлены прототипы функций, осуществляющих генерацию года, и их описание.

Таблица 7.4 – Прототипы функций, осуществляющих генерацию кода

|  |  |
| --- | --- |
| Generator::Generator(LT::LexTable lt, IT::IdTable it, OT::OpTable \_ot, Parm::PARM parm); | Основная функция. Формирует поток выходного файла и вызывает другие генерирующие функции. |
| void Generator::Head() | Функция, генерирующая заголовок ассемблерного файла (подключение библиотек, указание прототипов функций и т.д.). |
| void Generator::Constants() | Функция, генерирующая сегмент констант. |
| void Generator::Data() | Функция, генерирующая сегмент данных. |
| void Generator::Code() | Функция, генерирующая сегмент кода. |

## **7.5 Входные параметры генератора кода**

На вход генератору кода поступают таблицы лексем, идентификаторов и операций. Результаты работы генератора кода выводятся в файл расширения .asm.

## **7.6 Контрольный пример**

Результат работы контрольного примера приведён на рисунке 7.5.

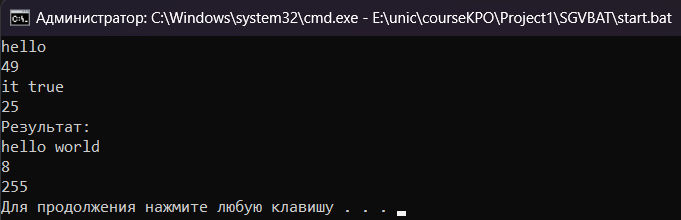


Рисунок 7.4 – Результат работы программы на языке SGV-2024

Результат выполнения всей программы и генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Ж.

# **8. Тестирование транслятора**

## **8.1 Общие положения**

Во время работы транслятора ошибки могут возникать на различных этапах: при анализе исходного текста программы, лексическом, синтаксическом или семантическом анализе. Транслятор фиксирует обнаруженные ошибки, записывая их в протокол. В записи указываются идентификатор ошибки, текст сообщения, а также номер строки и позиция в исходном коде. Обычно работа транслятора прекращается после появления ошибки, так как проблемы на одном этапе могут приводить к ошибкам на следующих (за исключением случая с синтаксическим анализатором).

## **8.2 Результаты тестирования**

В таблице 8.1 приведены результаты тестов для разных этапов трансляции.

Таблица 8.1 – Результаты тестов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| Лексический анализ | |
| declare integer a;  declare integer a; | Ошибка 116: #LEXICAL - Повторно объявленный идентификатор переменной, строка 4, позиция 20 |
| integer function ts(integer x, integer z){…};  string function ts(integer k){…}; | Ошибка 117: #LEXICAL - Повторно объявленный идентификатор функции, строка 13, позиция 76 |
| Синтаксический анализ | |
| declare int b != \_pow(k, 2); | Ошибка 602: #SYNTAX - Ошибка в выражении |
| string function sa(integer k)program {…};  declare string h;  string function sayhi() {…}; | Ошибка 600: #SYNTAX - Неверная структура программы |
| Семантический анализ | |
| if(5 >= 'hello')  {  x = y - 1;  } | Ошибка 713: #SEMANTICS - Несоответствие типов при сравнении, строка 47, позиция 179 |
| y = 1 + 2 \* '5'; | Ошибка 707: #SEMANTICS - Ошибочный тип члена выражения, строка 46, позиция 169 |

Результатом тестирования транслятора стало подтверждение работоспособности функций программы, вывод ошибок и генерация ассемблерного кода.

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования SGV-2024. В рамках проекта успешно реализованы следующие этапы:

– создана спецификация языка программирования;

– разработана архитектура транслятора;

– выполнена реализация лексического, синтаксического и семантического анализаторов;

– разработан генератор кода для ассемблера;

– проведено полное тестирование транслятора.

Финальная версия языка SGV-2024 обладает следующими возможностями:

– поддерживает 4 типа данных;

– реализует операторы вывода;

– позволяет вызывать функции стандартной библиотеки;

– включает 4 арифметических оператора для работы с выражениями;

– содержит 6 логических операторов;

– поддерживает работу с функциями и условными конструкциями.

Таким образом, в процессе выполнения курсовой работы были приобретены ценные знания и практические навыки в проектировании языков программирования и разработке программного обеспечения для трансляторов.

# **Список использованных литературных источников**

1. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003.
2. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005.
3. Принципы работы транслятора [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/435520/. – Дата доступа: 05.11.2024.
4. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003.
5. Наркевич, А. С. Конструирование программного обеспечения: метод. указания к выполнению курсового проекта для студентов специальности 1-40 01 01 «Программное обеспечение информационных технологий» / А. С. Наркевич – Минск : БГТУ, 2022.

**ПРИЛОЖЕНИЕ А**

|  |
| --- |
| integer function ts(integer x, integer z)  {  declare integer a;  declare string sts;  sts = 'hello world';  a = (2 + 2)\*2;  \_printStr('Результат:');  \_printStr(sts);  \_printInt(a);  a = \_pow(z, x);  return a;  };  string function sa(integer k)  {  declare string a;  declare integer b;  b = \_pow(k, 2);  \_printInt(b);  b = k;  if(b == k)  {  a = 'it true';  }  else  {  a = 'it false';  };  \_printStr(a);  return a;  };  string function sayhi()  {  \_printStr('hello');  return 'done';  };  main  {  declare integer x;  declare integer y;  declare integer z;  declare string ksa;  sayhi();  x = 2;  y = 1 + 2 \* 3;  if(x == y)  {  x = y - 1;  }  else  {  x = y;  };  ksa = sa(x);  z = \_exp(3) + 5;  \_printInt(z);  z = ts(x, y);  \_printInt(z);  \_pause();  return 0;  }; |

Листинг А.1 – Код контрольного примера

**ПРИЛОЖЕНИЕ Б**

|  |
| --- |
| FST::FST typeInteger("", 8,  FST::NODE(1, FST::RELATION('i', 1)),  FST::NODE(1, FST::RELATION('n', 2)),  FST::NODE(1, FST::RELATION('t', 3)),  FST::NODE(1, FST::RELATION('e', 4)),  FST::NODE(1, FST::RELATION('g', 5)),  FST::NODE(1, FST::RELATION('e', 6)),  FST::NODE(1, FST::RELATION('r', 7)),  FST::NODE()  );  FST::FST typeString("", 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()  );  FST::FST typeBool("", 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()  );  FST::FST typeChar("", 5,  FST::NODE(1, FST::RELATION('c', 1)),  FST::NODE(1, FST::RELATION('h', 2)),  FST::NODE(1, FST::RELATION('a', 3)),  FST::NODE(1, FST::RELATION('r', 4)),  FST::NODE()  );  FST::FST typeFunction("", 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()  );  FST::FST typeDeclare("", 8,  FST::NODE(1, FST::RELATION('d', 1)),  FST::NODE(1, FST::RELATION('e', 2)),  FST::NODE(1, FST::RELATION('c', 3)),  FST::NODE(1, FST::RELATION('l', 4)),  FST::NODE(1, FST::RELATION('a', 5)),  FST::NODE(1, FST::RELATION('r', 6)),  FST::NODE(1, FST::RELATION('e', 7)),  FST::NODE()  );  FST::FST typeReturn("", 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()  );  FST::FST typeMain("", 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()  );  FST::FST typePrintInt("", 10,  FST::NODE(1, FST::RELATION('\_', 1)),  FST::NODE(1, FST::RELATION('p', 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('t', 6)),  FST::NODE(1, FST::RELATION('I', 7)),  FST::NODE(1, FST::RELATION('n', 8)),  FST::NODE(1, FST::RELATION('t', 9)),  FST::NODE()  );  FST::FST typePrintStr("", 10,  FST::NODE(1, FST::RELATION('\_', 1)),  FST::NODE(1, FST::RELATION('p', 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('t', 6)),  FST::NODE(1, FST::RELATION('S', 7)),  FST::NODE(1, FST::RELATION('t', 8)),  FST::NODE(1, FST::RELATION('r', 9)),  FST::NODE()  );  FST::FST typePrintBool("", 11,  FST::NODE(1, FST::RELATION('\_', 1)),  FST::NODE(1, FST::RELATION('p', 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('t', 6)),  FST::NODE(1, FST::RELATION('B', 7)),  FST::NODE(1, FST::RELATION('o', 8)),  FST::NODE(1, FST::RELATION('o', 9)),  FST::NODE(1, FST::RELATION('l', 10)),  FST::NODE()  );  FST::FST typePrintChar("", 11,  FST::NODE(1, FST::RELATION('\_', 1)),  FST::NODE(1, FST::RELATION('p', 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('t', 6)),  FST::NODE(1, FST::RELATION('C', 7)),  FST::NODE(1, FST::RELATION('h', 8)),  FST::NODE(1, FST::RELATION('a', 9)),  FST::NODE(1, FST::RELATION('r', 10)),  FST::NODE()  );  FST::FST typeSpace("", 2,  FST::NODE(1, FST::RELATION(' ', 1)),  FST::NODE()  );  FST::FST typeLeftBrace("", 2,  FST::NODE(1, FST::RELATION('{', 1)),  FST::NODE()  );  FST::FST typeRightBrace("", 2,  FST::NODE(1, FST::RELATION('}', 1)),  FST::NODE()  );  FST::FST typeLeftThesis("", 2,  FST::NODE(1, FST::RELATION('(', 1)),  FST::NODE()  );  FST::FST typeRightThesis("", 2,  FST::NODE(1, FST::RELATION(')', 1)),  FST::NODE()  );  FST::FST typeSemicolon("", 2,  FST::NODE(1, FST::RELATION(';', 1)),  FST::NODE()  );  FST::FST typeComma("", 2,  FST::NODE(1, FST::RELATION(',', 1)),  FST::NODE()  );  FST::FST typePlus("", 2,  FST::NODE(1, FST::RELATION('+', 1)),  FST::NODE());  FST::FST typeMinus("", 2,  FST::NODE(1, FST::RELATION('-', 1)),  FST::NODE());  FST::FST typeDel("", 2,  FST::NODE(1, FST::RELATION('/', 1)),  FST::NODE());  FST::FST typeMulti("", 2,  FST::NODE(1, FST::RELATION('\*', 1)),  FST::NODE());  FST::FST typeEquals("", 2,  FST::NODE(1, FST::RELATION('=', 1)),  FST::NODE());  FST::FST typeIsEquals("", 3,  FST::NODE(1, FST::RELATION('=', 1)),  FST::NODE(1, FST::RELATION('=', 2)),  FST::NODE());  FST::FST typeIsNotEquals("", 3,  FST::NODE(1, FST::RELATION('!', 1)),  FST::NODE(1, FST::RELATION('=', 2)),  FST::NODE());  FST::FST typeGreaterOrEquals("", 3,  FST::NODE(1, FST::RELATION('>', 1)),  FST::NODE(1, FST::RELATION('=', 2)),  FST::NODE());  FST::FST typePow("", 5,  FST::NODE(1, FST::RELATION('\_', 1)),  FST::NODE(1, FST::RELATION('p', 2)),  FST::NODE(1, FST::RELATION('o', 3)),  FST::NODE(1, FST::RELATION('w', 4)),  FST::NODE());  FST::FST typeExp("", 5,  FST::NODE(1, FST::RELATION('\_', 1)),  FST::NODE(1, FST::RELATION('e', 2)),  FST::NODE(1, FST::RELATION('x', 3)),  FST::NODE(1, FST::RELATION('p', 4)),  FST::NODE());  FST::FST typePause("", 7,  FST::NODE(1, FST::RELATION('\_', 1)),  FST::NODE(1, FST::RELATION('p', 2)),  FST::NODE(1, FST::RELATION('a', 3)),  FST::NODE(1, FST::RELATION('u', 4)),  FST::NODE(1, FST::RELATION('s', 5)),  FST::NODE(1, FST::RELATION('e', 6)),  FST::NODE());  FST::FST typeLessOrEquals("", 3,  FST::NODE(1, FST::RELATION('<', 1)),  FST::NODE(1, FST::RELATION('=', 2)),  FST::NODE());  FST::FST typeIf("", 3,  FST::NODE(1, FST::RELATION('i', 1)),  FST::NODE(1, FST::RELATION('f', 2)),  FST::NODE());  FST::FST typeElse("", 5,  FST::NODE(1, FST::RELATION('e', 1)),  FST::NODE(1, FST::RELATION('l', 2)),  FST::NODE(1, FST::RELATION('s', 3)),  FST::NODE(1, FST::RELATION('e', 4)),  FST::NODE());  FST::FST typeTrue("", 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());  FST::FST typeFalse("", 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());  FST::FST typeLess("", 2,  FST::NODE(1, FST::RELATION('<', 1)),  FST::NODE());  FST::FST typeGreater("", 2,  FST::NODE(1, FST::RELATION('>', 1)),  FST::NODE());  FST::FST typeSlashN("", 2,  FST::NODE(1, FST::RELATION('\n', 1)),  FST::NODE());  FST::FST typeIdentificator("", 2,  FST::NODE(52,  FST::RELATION('a', 0), FST::RELATION('b', 0),  FST::RELATION('c', 0), FST::RELATION('d', 0),  FST::RELATION('e', 0), FST::RELATION('f', 0),  FST::RELATION('g', 0), FST::RELATION('h', 0),  FST::RELATION('i', 0), FST::RELATION('j', 0),  FST::RELATION('k', 0), FST::RELATION('l', 0),  FST::RELATION('m', 0), FST::RELATION('n', 0),  FST::RELATION('o', 0), FST::RELATION('p', 0),  FST::RELATION('q', 0), FST::RELATION('r', 0),  FST::RELATION('s', 0), FST::RELATION('t', 0),  FST::RELATION('u', 0), FST::RELATION('v', 0),  FST::RELATION('w', 0), FST::RELATION('x', 0),  FST::RELATION('y', 0), FST::RELATION('z', 0),  FST::RELATION('a', 1), FST::RELATION('b', 1),  FST::RELATION('c', 1), FST::RELATION('d', 1),  FST::RELATION('e', 1), FST::RELATION('f', 1),  FST::RELATION('g', 1), FST::RELATION('h', 1),  FST::RELATION('i', 1), FST::RELATION('j', 1),  FST::RELATION('k', 1), FST::RELATION('l', 1),  FST::RELATION('m', 1), FST::RELATION('n', 1),  FST::RELATION('o', 1), FST::RELATION('p', 1),  FST::RELATION('q', 1), FST::RELATION('r', 1),  FST::RELATION('s', 1), FST::RELATION('t', 1),  FST::RELATION('u', 1), FST::RELATION('v', 1),  FST::RELATION('w', 1), FST::RELATION('x', 1),  FST::RELATION('y', 1), FST::RELATION('z', 1)  ),  FST::NODE()  );  FST::FST typeCharLiteral("", 4,  FST::NODE(1,  FST::RELATION('\'', 1)),  FST::NODE(192,  FST::RELATION('!', 2), FST::RELATION('@', 2), FST::RELATION('#', 2), FST::RELATION('$', 2), FST::RELATION('%', 2), FST::RELATION('^', 2),  FST::RELATION('&', 2), FST::RELATION('\*', 2), FST::RELATION('(', 2), FST::RELATION(')', 2), FST::RELATION('-', 2), FST::RELATION('\_', 2),  FST::RELATION('+', 2), FST::RELATION('=', 2), FST::RELATION('~', 2), FST::RELATION('`', 2), FST::RELATION('1', 2), FST::RELATION('2', 2),  FST::RELATION('3', 2), FST::RELATION('4', 2), FST::RELATION('5', 2), FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2),  FST::RELATION('9', 2), FST::RELATION('0', 2), FST::RELATION(' ', 2), FST::RELATION(' ', 2), FST::RELATION('q', 2), FST::RELATION('w', 2),  FST::RELATION('e', 2), FST::RELATION('r', 2), FST::RELATION('t', 2), FST::RELATION('y', 2), FST::RELATION('u', 2), FST::RELATION('i', 2),  FST::RELATION('o', 2), FST::RELATION('p', 2), FST::RELATION('{', 2), FST::RELATION('[', 2), FST::RELATION('}', 2), FST::RELATION(']', 2),  FST::RELATION('\\', 2), FST::RELATION('|', 2), FST::RELATION('Q', 2), FST::RELATION('W', 2), FST::RELATION('R', 2), FST::RELATION('T', 2), FST::RELATION('Y', 2),  FST::RELATION('U', 2), FST::RELATION('I', 2), FST::RELATION('O', 2), FST::RELATION('P', 2), FST::RELATION('a', 2), FST::RELATION('s', 2),  FST::RELATION('d', 2), FST::RELATION('f', 2), FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('j', 2), FST::RELATION('k', 2),  FST::RELATION('l', 2), FST::RELATION(':', 2), FST::RELATION(';', 2), FST::RELATION('\'', 2), FST::RELATION('"', 2), FST::RELATION('A', 2),  FST::RELATION('S', 2), FST::RELATION('D', 2), FST::RELATION('F', 2), FST::RELATION('G', 2), FST::RELATION('H', 2), FST::RELATION('J', 2),  FST::RELATION('K', 2), FST::RELATION('L', 2), FST::RELATION('z', 2), FST::RELATION('x', 2), FST::RELATION('c', 2), FST::RELATION('v', 2),  FST::RELATION('b', 2), FST::RELATION('n', 2), FST::RELATION('m', 2), FST::RELATION(',', 2), FST::RELATION('.', 2), FST::RELATION('/', 2),  FST::RELATION('Z', 2), FST::RELATION('X', 2), FST::RELATION('C', 2), FST::RELATION('V', 2), FST::RELATION('B', 2), FST::RELATION('N', 2),  FST::RELATION('M', 2), FST::RELATION('<', 2), FST::RELATION('>', 2), FST::RELATION('?', 2), FST::RELATION('ё', 2), FST::RELATION('Ё', 2),  FST::RELATION('й', 2), FST::RELATION('ц', 2), FST::RELATION('у', 2), FST::RELATION('к', 2), FST::RELATION('е', 2), FST::RELATION('н', 2),  FST::RELATION('г', 2), FST::RELATION('ш', 2), FST::RELATION('щ', 2), FST::RELATION('з', 2), FST::RELATION('х', 2), FST::RELATION('ъ', 2),  FST::RELATION('Й', 2), FST::RELATION('Ц', 2), FST::RELATION('У', 2), FST::RELATION('К', 2), FST::RELATION('Е', 2), FST::RELATION('Н', 2),  FST::RELATION('Г', 2), FST::RELATION('Ш', 2), FST::RELATION('Щ', 2), FST::RELATION('З', 2), FST::RELATION('Х', 2), FST::RELATION('Ъ', 2),  FST::RELATION('ф', 2), FST::RELATION('ы', 2), FST::RELATION('в', 2), FST::RELATION('а', 2), FST::RELATION('п', 2), FST::RELATION('р', 2),  FST::RELATION('о', 2), FST::RELATION('л', 2), FST::RELATION('д', 2), FST::RELATION('ж', 2), FST::RELATION('э', 2), FST::RELATION('Ф', 2),  FST::RELATION('Ы', 2), FST::RELATION('В', 2), FST::RELATION('А', 2), FST::RELATION('П', 2), FST::RELATION('Р', 2), FST::RELATION('О', 2),  FST::RELATION('Л', 2), FST::RELATION('Д', 2), FST::RELATION('Ж', 2), FST::RELATION('Э', 2), FST::RELATION('я', 2), FST::RELATION('ч', 2),  FST::RELATION('с', 2), FST::RELATION('м', 2), FST::RELATION('и', 2), FST::RELATION('т', 2), FST::RELATION('ь', 2), FST::RELATION('б', 2),  FST::RELATION('ю', 2), FST::RELATION('Я', 2), FST::RELATION('Ч', 2), FST::RELATION('С', 2), FST::RELATION('М', 2), FST::RELATION('И', 2),  FST::RELATION('Т', 2), FST::RELATION('Ь', 2), FST::RELATION('Б', 2), FST::RELATION('Ю', 2)  ),  FST::NODE(1,  FST::RELATION('\'', 3)),  FST::NODE()  );  FST::FST typeStringLiteral("", 4,  FST::NODE(1,  FST::RELATION('\'', 1)),  FST::NODE(384,  FST::RELATION('!', 2), FST::RELATION('@', 2), FST::RELATION('#', 2), FST::RELATION('$', 2), FST::RELATION('%', 2), FST::RELATION('^', 2),  FST::RELATION('&', 2), FST::RELATION('\*', 2), FST::RELATION('(', 2), FST::RELATION(')', 2), FST::RELATION('-', 2), FST::RELATION('\_', 2),  FST::RELATION('+', 2), FST::RELATION('=', 2), FST::RELATION('~', 2), FST::RELATION('`', 2), FST::RELATION('1', 2), FST::RELATION('2', 2),  FST::RELATION('3', 2), FST::RELATION('4', 2), FST::RELATION('5', 2), FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2),  FST::RELATION('9', 2), FST::RELATION('0', 2), FST::RELATION(' ', 2), FST::RELATION(' ', 2), FST::RELATION('q', 2), FST::RELATION('w', 2),  FST::RELATION('e', 2), FST::RELATION('r', 2), FST::RELATION('t', 2), FST::RELATION('y', 2), FST::RELATION('u', 2), FST::RELATION('i', 2),  FST::RELATION('o', 2), FST::RELATION('p', 2), FST::RELATION('{', 2), FST::RELATION('[', 2), FST::RELATION('}', 2), FST::RELATION(']', 2),  FST::RELATION('\\', 2), FST::RELATION('|', 2), FST::RELATION('Q', 2), FST::RELATION('W', 2), FST::RELATION('R', 2), FST::RELATION('T', 2), FST::RELATION('Y', 2),  FST::RELATION('U', 2), FST::RELATION('I', 2), FST::RELATION('O', 2), FST::RELATION('P', 2), FST::RELATION('a', 2), FST::RELATION('s', 2),  FST::RELATION('d', 2), FST::RELATION('f', 2), FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('j', 2), FST::RELATION('k', 2),  FST::RELATION('l', 2), FST::RELATION(':', 2), FST::RELATION(';', 2), FST::RELATION('\'', 2), FST::RELATION('"', 2), FST::RELATION('A', 2),  FST::RELATION('S', 2), FST::RELATION('D', 2), FST::RELATION('F', 2), FST::RELATION('G', 2), FST::RELATION('H', 2), FST::RELATION('J', 2),  FST::RELATION('K', 2), FST::RELATION('L', 2), FST::RELATION('z', 2), FST::RELATION('x', 2), FST::RELATION('c', 2), FST::RELATION('v', 2),  FST::RELATION('b', 2), FST::RELATION('n', 2), FST::RELATION('m', 2), FST::RELATION(',', 2), FST::RELATION('.', 2), FST::RELATION('/', 2),  FST::RELATION('Z', 2), FST::RELATION('X', 2), FST::RELATION('C', 2), FST::RELATION('V', 2), FST::RELATION('B', 2), FST::RELATION('N', 2),  FST::RELATION('M', 2), FST::RELATION('<', 2), FST::RELATION('>', 2), FST::RELATION('?', 2), FST::RELATION('ё', 2), FST::RELATION('Ё', 2),  FST::RELATION('й', 2), FST::RELATION('ц', 2), FST::RELATION('у', 2), FST::RELATION('к', 2), FST::RELATION('е', 2), FST::RELATION('н', 2),  FST::RELATION('г', 2), FST::RELATION('ш', 2), FST::RELATION('щ', 2), FST::RELATION('з', 2), FST::RELATION('х', 2), FST::RELATION('ъ', 2),  FST::RELATION('Й', 2), FST::RELATION('Ц', 2), FST::RELATION('У', 2), FST::RELATION('К', 2), FST::RELATION('Е', 2), FST::RELATION('Н', 2),  FST::RELATION('Г', 2), FST::RELATION('Ш', 2), FST::RELATION('Щ', 2), FST::RELATION('З', 2), FST::RELATION('Х', 2), FST::RELATION('Ъ', 2),  FST::RELATION('ф', 2), FST::RELATION('ы', 2), FST::RELATION('в', 2), FST::RELATION('а', 2), FST::RELATION('п', 2), FST::RELATION('р', 2),  FST::RELATION('о', 2), FST::RELATION('л', 2), FST::RELATION('д', 2), FST::RELATION('ж', 2), FST::RELATION('э', 2), FST::RELATION('Ф', 2),  FST::RELATION('Ы', 2), FST::RELATION('В', 2), FST::RELATION('А', 2), FST::RELATION('П', 2), FST::RELATION('Р', 2), FST::RELATION('О', 2),  FST::RELATION('Л', 2), FST::RELATION('Д', 2), FST::RELATION('Ж', 2), FST::RELATION('Э', 2), FST::RELATION('я', 2), FST::RELATION('ч', 2),  FST::RELATION('с', 2), FST::RELATION('м', 2), FST::RELATION('и', 2), FST::RELATION('т', 2), FST::RELATION('ь', 2), FST::RELATION('б', 2),  FST::RELATION('ю', 2), FST::RELATION('Я', 2), FST::RELATION('Ч', 2), FST::RELATION('С', 2), FST::RELATION('М', 2), FST::RELATION('И', 2),  FST::RELATION('Т', 2), FST::RELATION('Ь', 2), FST::RELATION('Б', 2), FST::RELATION('Ю', 2),  FST::RELATION('!', 1), FST::RELATION('@', 1), FST::RELATION('#', 1), FST::RELATION('$', 1), FST::RELATION('%', 1), FST::RELATION('^', 1),  FST::RELATION('&', 1), FST::RELATION('\*', 1), FST::RELATION('(', 1), FST::RELATION(')', 1), FST::RELATION('-', 1), FST::RELATION('\_', 1),  FST::RELATION('+', 1), FST::RELATION('=', 1), FST::RELATION('~', 1), FST::RELATION('`', 1), FST::RELATION('1', 1), FST::RELATION('2', 1),  FST::RELATION('3', 1), FST::RELATION('4', 1), FST::RELATION('5', 1), FST::RELATION('6', 1), FST::RELATION('7', 1), FST::RELATION('8', 1),  FST::RELATION('9', 1), FST::RELATION('0', 1), FST::RELATION(' ', 1), FST::RELATION(' ', 1), FST::RELATION('q', 1), FST::RELATION('w', 1),  FST::RELATION('e', 1), FST::RELATION('r', 1), FST::RELATION('t', 1), FST::RELATION('y', 1), FST::RELATION('u', 1), FST::RELATION('i', 1),  FST::RELATION('o', 1), FST::RELATION('p', 1), FST::RELATION('{', 1), FST::RELATION('[', 1), FST::RELATION('}', 1), FST::RELATION(']', 1),  FST::RELATION('\\', 1), FST::RELATION('|', 1), FST::RELATION('Q', 1), FST::RELATION('W', 1), FST::RELATION('R', 1), FST::RELATION('T', 1), FST::RELATION('Y', 1),  FST::RELATION('U', 1), FST::RELATION('I', 1), FST::RELATION('O', 1), FST::RELATION('P', 1), FST::RELATION('a', 1), FST::RELATION('s', 1),  FST::RELATION('d', 1), FST::RELATION('f', 1), FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('j', 1), FST::RELATION('k', 1),  FST::RELATION('l', 1), FST::RELATION(':', 1), FST::RELATION(';', 1), FST::RELATION('\'', 1), FST::RELATION('"', 1), FST::RELATION('A', 1),  FST::RELATION('S', 1), FST::RELATION('D', 1), FST::RELATION('F', 1), FST::RELATION('G', 1), FST::RELATION('H', 1), FST::RELATION('J', 1),  FST::RELATION('K', 1), FST::RELATION('L', 1), FST::RELATION('z', 1), FST::RELATION('x', 1), FST::RELATION('c', 1), FST::RELATION('v', 1),  FST::RELATION('b', 1), FST::RELATION('n', 1), FST::RELATION('m', 1), FST::RELATION(',', 1), FST::RELATION('.', 1), FST::RELATION('/', 1),  FST::RELATION('Z', 1), FST::RELATION('X', 1), FST::RELATION('C', 1), FST::RELATION('V', 1), FST::RELATION('B', 1), FST::RELATION('N', 1),  FST::RELATION('M', 1), FST::RELATION('<', 1), FST::RELATION('>', 1), FST::RELATION('?', 1), FST::RELATION('ё', 1), FST::RELATION('Ё', 1),  FST::RELATION('й', 1), FST::RELATION('ц', 1), FST::RELATION('у', 1), FST::RELATION('к', 1), FST::RELATION('е', 1), FST::RELATION('н', 1),  FST::RELATION('г', 1), FST::RELATION('ш', 1), FST::RELATION('щ', 1), FST::RELATION('з', 1), FST::RELATION('х', 1), FST::RELATION('ъ', 1),  FST::RELATION('Й', 1), FST::RELATION('Ц', 1), FST::RELATION('У', 1), FST::RELATION('К', 1), FST::RELATION('Е', 1), FST::RELATION('Н', 1),  FST::RELATION('Г', 1), FST::RELATION('Ш', 1), FST::RELATION('Щ', 1), FST::RELATION('З', 1), FST::RELATION('Х', 1), FST::RELATION('Ъ', 1),  FST::RELATION('ф', 1), FST::RELATION('ы', 1), FST::RELATION('в', 1), FST::RELATION('а', 1), FST::RELATION('п', 1), FST::RELATION('р', 1),  FST::RELATION('о', 1), FST::RELATION('л', 1), FST::RELATION('д', 1), FST::RELATION('ж', 1), FST::RELATION('э', 1), FST::RELATION('Ф', 1),  FST::RELATION('Ы', 1), FST::RELATION('В', 1), FST::RELATION('А', 1), FST::RELATION('П', 1), FST::RELATION('Р', 1), FST::RELATION('О', 1),  FST::RELATION('Л', 1), FST::RELATION('Д', 1), FST::RELATION('Ж', 1), FST::RELATION('Э', 1), FST::RELATION('я', 1), FST::RELATION('ч', 1),  FST::RELATION('с', 1), FST::RELATION('м', 1), FST::RELATION('и', 1), FST::RELATION('т', 1), FST::RELATION('ь', 1), FST::RELATION('б', 1),  FST::RELATION('ю', 1), FST::RELATION('Я', 1), FST::RELATION('Ч', 1), FST::RELATION('С', 1), FST::RELATION('М', 1), FST::RELATION('И', 1),  FST::RELATION('Т', 1), FST::RELATION('Ь', 1), FST::RELATION('Б', 1), FST::RELATION('Ю', 1)  ),  FST::NODE(1,  FST::RELATION('\'', 3)),  FST::NODE()  );  FST::FST typeNumbLiteral("", 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()  ); |

Листинг Б.1 – Графы конечного автомата для цепочек языка

**ПРИЛОЖЕНИЕ В**

|  |
| --- |
| 1 tfi(ti,ti)  2 {  3 dti;  4 dti;  5 i=l;  6 i=llvlv;~~  7 i(l);  8 i(i);  9 i(i);  10 i=iii@2;~  11 ri;  12 };  13 tfi(ti)  14 {  15 dti;  16 dti;  17 i=iil@2;~  18 i(i);  19 i=i;  20 s(ici)  21 {  22 i=l;  23 }  24 e  25 {  26 i=l;  27 };  28 i(i);  29 ri;  30 };  31 t  32 fi()  33 {  34 i(l);  35 rl;  36 };  37 m  38 {  39 d  40 ti;  41 dti;  42 dti;  43 dti;  44 i();  45 i=l;  46 i=lllvv;  47 s(ici)  48 {  49 i=ilv;  50 }  51 e  52 {  53 i=i;  54 };  55 i=ii@1;  56 i=il@1lv;  57 i(i);  58 i=iii@2;~  59 i(i);  60 i();  61 rl;  62 }; |

Листинг В.1 – Таблица лексем

|  |
| --- |
| №: 1 idxfirstLE: 2 Идентификатор: ts Тип: Функция Тип данных: int  №: 2 idxfirstLE: 5 Идентификатор: tsx Тип: Параметр Тип данных: int  №: 3 idxfirstLE: 8 Идентификатор: tsz Тип: Параметр Тип данных: int  №: 4 idxfirstLE: 15 Идентификатор: tsa Тип: Переменная Тип данных: int Значение: 0  №: 5 idxfirstLE: 20 Идентификатор: tssts Тип: Переменная Тип данных: string  №: 6 idxfirstLE: 23 Указатель на: tssts  №: 7 idxfirstLE: 25 Идентификатор: L1 Тип: Литерал Тип данных: string Значение: hello world Длина строки: 11  №: 8 idxfirstLE: 28 Указатель на: tsa  №: 9 idxfirstLE: 31 Идентификатор: L2 Тип: Литерал Тип данных: int Значение: 2  №: 10 idxfirstLE: 33 Идентификатор: L3 Тип: Литерал Тип данных: int Значение: 2  №: 11 idxfirstLE: 36 Идентификатор: L4 Тип: Литерал Тип данных: int Значение: 2  №: 12 idxfirstLE: 39 Указатель на: \_printStr  №: 13 idxfirstLE: 41 Идентификатор: L5 Тип: Литерал Тип данных: string Значение: Результат: Длина строки: 10  №: 14 idxfirstLE: 45 Указатель на: \_printStr  №: 15 idxfirstLE: 47 Указатель на: tssts  №: 16 idxfirstLE: 51 Указатель на: \_printInt  №: 17 idxfirstLE: 53 Указатель на: tsa  №: 18 idxfirstLE: 57 Указатель на: tsa  №: 19 idxfirstLE: 59 Указатель на: \_pow  №: 20 idxfirstLE: 61 Указатель на: tsz  №: 21 idxfirstLE: 63 Указатель на: tsx  №: 22 idxfirstLE: 68 Указатель на: tsa  №: 23 idxfirstLE: 76 Идентификатор: sa Тип: Функция Тип данных: string  №: 24 idxfirstLE: 79 Идентификатор: sak Тип: Параметр Тип данных: int  №: 25 idxfirstLE: 86 Идентификатор: saa Тип: Переменная Тип данных: string  №: 26 idxfirstLE: 91 Идентификатор: sab Тип: Переменная Тип данных: int Значение: 0  №: 27 idxfirstLE: 94 Указатель на: sab  №: 28 idxfirstLE: 96 Указатель на: \_pow  №: 29 idxfirstLE: 98 Указатель на: sak  №: 30 idxfirstLE: 100 Идентификатор: L6 Тип: Литерал Тип данных: int Значение: 2  №: 31 idxfirstLE: 104 Указатель на: \_printInt  №: 32 idxfirstLE: 106 Указатель на: sab  №: 33 idxfirstLE: 110 Указатель на: sab  №: 34 idxfirstLE: 112 Указатель на: sak  №: 35 idxfirstLE: 117 Указатель на: sab  №: 36 idxfirstLE: 119 Указатель на: sak  №: 37 idxfirstLE: 124 Указатель на: saa  №: 38 idxfirstLE: 126 Идентификатор: L7 Тип: Литерал Тип данных: string Значение: it true Длина строки: 7  №: 39 idxfirstLE: 135 Указатель на: saa  №: 40 idxfirstLE: 137 Идентификатор: L8 Тип: Литерал Тип данных: string Значение: it false Длина строки: 8  №: 41 idxfirstLE: 143 Указатель на: \_printStr  №: 42 idxfirstLE: 145 Указатель на: saa  №: 43 idxfirstLE: 150 Указатель на: saa  №: 44 idxfirstLE: 159 Идентификатор: sayhi Тип: Функция Тип данных: string  №: 45 idxfirstLE: 165 Указатель на: \_printStr  №: 46 idxfirstLE: 167 Идентификатор: L9 Тип: Литерал Тип данных: string Значение: hello Длина строки: 5  №: 47 idxfirstLE: 172 Идентификатор: L10 Тип: Литерал Тип данных: string Значение: done Длина строки: 4  №: 48 idxfirstLE: 185 Идентификатор: mainx Тип: Переменная Тип данных: int Значение: 0  №: 49 idxfirstLE: 190 Идентификатор: mainy Тип: Переменная Тип данных: int Значение: 0  №: 50 idxfirstLE: 195 Идентификатор: mainz Тип: Переменная Тип данных: int Значение: 0  №: 51 idxfirstLE: 200 Идентификатор: mainksa Тип: Переменная Тип данных: string  №: 52 idxfirstLE: 203 Указатель на: sayhi  №: 53 idxfirstLE: 208 Указатель на: mainx  №: 54 idxfirstLE: 210 Идентификатор: L11 Тип: Литерал Тип данных: int Значение: 2  №: 55 idxfirstLE: 213 Указатель на: mainy  №: 56 idxfirstLE: 215 Идентификатор: L12 Тип: Литерал Тип данных: int Значение: 1  №: 57 idxfirstLE: 217 Идентификатор: L13 Тип: Литерал Тип данных: int Значение: 2  №: 58 idxfirstLE: 219 Идентификатор: L14 Тип: Литерал Тип данных: int Значение: 3  №: 59 idxfirstLE: 224 Указатель на: mainx  №: 60 idxfirstLE: 226 Указатель на: mainy  №: 61 idxfirstLE: 231 Указатель на: mainx  №: 62 idxfirstLE: 233 Указатель на: mainy  №: 63 idxfirstLE: 235 Идентификатор: L15 Тип: Литерал Тип данных: int Значение: 1  №: 64 idxfirstLE: 244 Указатель на: mainx  №: 65 idxfirstLE: 246 Указатель на: mainy  №: 66 idxfirstLE: 252 Указатель на: mainksa  №: 67 idxfirstLE: 254 Указатель на: sa  №: 68 idxfirstLE: 256 Указатель на: mainx  №: 69 idxfirstLE: 260 Указатель на: mainz  №: 70 idxfirstLE: 262 Указатель на: \_exp  №: 71 idxfirstLE: 264 Идентификатор: L16 Тип: Литерал Тип данных: int Значение: 3  №: 72 idxfirstLE: 267 Идентификатор: L17 Тип: Литерал Тип данных: int Значение: 5  №: 73 idxfirstLE: 270 Указатель на: \_printInt  №: 74 idxfirstLE: 272 Указатель на: mainz  №: 75 idxfirstLE: 276 Указатель на: mainz  №: 76 idxfirstLE: 278 Указатель на: ts  №: 77 idxfirstLE: 280 Указатель на: mainx  №: 78 idxfirstLE: 282 Указатель на: mainy  №: 79 idxfirstLE: 286 Указатель на: \_printInt  №: 80 idxfirstLE: 288 Указатель на: mainz  №: 81 idxfirstLE: 292 Указатель на: \_pause  №: 82 idxfirstLE: 298 Идентификатор: L18 Тип: Литерал Тип данных: int Значение: 0 |

Листинг В.2 – Таблица идентификаторов

|  |
| --- |
| Лексема: + Индекс в таблице лексем: 27  Лексема: \* Индекс в таблице лексем: 30  Лексема: == Индекс в таблице лексем: 99  Лексема: + Индекс в таблице лексем: 171  Лексема: \* Индекс в таблице лексем: 173  Лексема: == Индекс в таблице лексем: 179  Лексема: - Индекс в таблице лексем: 186  Лексема: + Индекс в таблице лексем: 211 |

Листинг В.3 – Таблица операций

**ПРИЛОЖЕНИЕ Г**

|  |
| --- |
| Rule(NS('S'), GRB\_ERROR\_SERIES + 0,  7,  Rule::Chain(8, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';')),  Rule::Chain(7, TS('m'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';')),  Rule::Chain(14, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),  Rule::Chain(14, TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),  Rule::Chain(9, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),  Rule::Chain(13, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';')),  Rule::Chain(13, TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'))  ),  Rule(NS('N'), GRB\_ERROR\_SERIES + 1, //Операторы программы  17,  Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),  Rule::Chain(7, TS('d'), TS('t'), TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(7, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';')),  Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),  Rule::Chain(4, TS('r'), NS('E'), TS(';'), NS('N')),  Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(4, TS('i'), NS('E'), TS(';'), NS('N')),  Rule::Chain(3, TS('i'), NS('E'), TS(';')),  Rule::Chain(3, NS('E'), TS(';'), NS('N')),  Rule::Chain(11, TS('s'), TS('('), NS('K'), TS('c'), NS('K'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';'), NS('N')),  Rule::Chain(10, TS('s'), TS('('), NS('K'), TS('c'), NS('K'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';')),  Rule::Chain(15, TS('s'), TS('('), NS('K'), TS('c'), NS('K'), TS(')'), TS('{'), NS('N'), TS('}'), TS('e'), TS('{'), NS('N'), TS('}'), TS(';'), NS('N')),  Rule::Chain(14, TS('s'), TS('('), NS('K'), TS('c'), NS('K'), TS(')'), TS('{'), NS('N'), TS('}'), TS('e'), TS('{'), NS('N'), TS('}'), TS(';')),  Rule::Chain(4, TS('i'), TS('('), TS(')'), TS(';')),  Rule::Chain(5, TS('i'), TS('('), TS(')'), TS(';'), NS('N'))  ),  Rule(NS('E'), GRB\_ERROR\_SERIES + 2, //Выражение  9,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(2, TS('i'), NS('M')),  Rule::Chain(2, TS('l'), NS('M')),  Rule::Chain(3, TS('('), NS('E'), TS(')')),  Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),  Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M')),  Rule::Chain(3, TS('i'), TS('('), TS(')'))  ),  Rule(NS('W'), GRB\_ERROR\_SERIES + 3, //Подвыражение  4,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('i'), TS(','), NS('W')),  Rule::Chain(3, TS('l'), TS(','), NS('W'))  ),  Rule(NS('F'), GRB\_ERROR\_SERIES + 4, //Параметры функции  2,  Rule::Chain(2, TS('t'), TS('i')),  Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))  ),  Rule(NS('M'), GRB\_ERROR\_SERIES + 5, //Выражение  2,  Rule::Chain(2, TS('v'), NS('E')),  Rule::Chain(3, TS('v'), NS('E'), NS('M'))  ),  Rule(NS('K'), GRB\_ERROR\_SERIES + 6, //Литерал или идентификатор  2,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l'))  ) |

Листинг Г.1 – Правила языка

**ПРИЛОЖЕНИЕ Д**



Рисунок Д.1 – FST.h

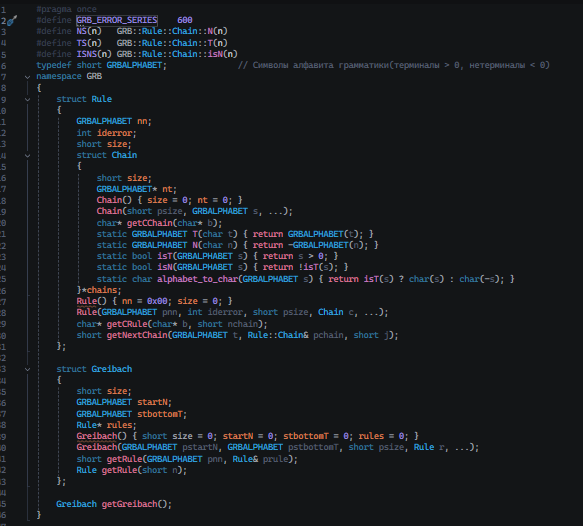


Рисунок Д.2 – GRB.h

**ПРИЛОЖЕНИЕ Е**

|  |
| --- |
| Шаг: Правило Входная лента Стек  0 : S->tfi(F){NrE;};S tfi(ti,ti){dti;dti;i=l;i= S$  0 : SAVESTATE: 1  0 : tfi(ti,ti){dti;dti;i=l;i= tfi(F){NrE;};S$  1 : fi(ti,ti){dti;dti;i=l;i=( fi(F){NrE;};S$  2 : i(ti,ti){dti;dti;i=l;i=(l i(F){NrE;};S$  3 : (ti,ti){dti;dti;i=l;i=(lv (F){NrE;};S$  4 : ti,ti){dti;dti;i=l;i=(lvl F){NrE;};S$  5 : F->ti ti,ti){dti;dti;i=l;i=(lvl F){NrE;};S$  5 : SAVESTATE: 2  5 : ti,ti){dti;dti;i=l;i=(lvl ti){NrE;};S$  6 : i,ti){dti;dti;i=l;i=(lvl) i){NrE;};S$  7 : ,ti){dti;dti;i=l;i=(lvl)v ){NrE;};S$  8 : TS\_NOK/NS\_NORULECHAIN  8 : RESSTATE  8 : ti,ti){dti;dti;i=l;i=(lvl F){NrE;};S$  9 : F->ti,F ti,ti){dti;dti;i=l;i=(lvl F){NrE;};S$  9 : SAVESTATE: 2  9 : ti,ti){dti;dti;i=l;i=(lvl ti,F){NrE;};S$  10 : i,ti){dti;dti;i=l;i=(lvl) i,F){NrE;};S$  11 : ,ti){dti;dti;i=l;i=(lvl)v ,F){NrE;};S$  12 : ti){dti;dti;i=l;i=(lvl)vl F){NrE;};S$  13 : F->ti ti){dti;dti;i=l;i=(lvl)vl F){NrE;};S$  13 : SAVESTATE: 3  13 : ti){dti;dti;i=l;i=(lvl)vl ti){NrE;};S$  14 : i){dti;dti;i=l;i=(lvl)vl; i){NrE;};S$  15 : ){dti;dti;i=l;i=(lvl)vl;i ){NrE;};S$  16 : {dti;dti;i=l;i=(lvl)vl;i( {NrE;};S$  17 : dti;dti;i=l;i=(lvl)vl;i(l NrE;};S$  …  1553: E->l l;}; E;};а$  1553: SAVESTATE: 100  1553: l;}; l;};а$  1554: ;}; ;};а$  1555: }; };а$  1556: ; ;а$  1557: а$  1558: LENTA\_END  1559: -------> NS\_LENTA\_END  --------------------------------------------------------------  Всего символов 242, Синтаксический анализ выполнен без ошибок  0 : S->tfi(F){NrE;};S  4 : F->ti,F  7 : F->ti  11 : N->dti;N  15 : N->dti;N  19 : N->i=E;N  21 : E->l  23 : N->i=E;N  25 : E->(E)M  26 : E->lM  27 : M->vE  28 : E->l  30 : M->vE  31 : E->l  33 : N->iE;N  34 : E->(E)  35 : E->l  38 : N->iE;N  39 : E->(E)  40 : E->i  43 : N->iE;N  44 : E->(E)  45 : E->i  48 : N->i=E;  50 : E->i(W)  52 : W->i,W  54 : W->i  58 : E->i  62 : S->tfi(F){NrE;};S  66 : F->ti  70 : N->dti;N  74 : N->dti;N  78 : N->i=E;N  80 : E->i(W)  82 : W->i,W  84 : W->l  87 : N->iE;N  88 : E->(E)  89 : E->i  92 : N->i=E;N  94 : E->i  96 : N->s(KcK){N}e{N};N  98 : K->i  100 : K->i  103 : N->i=E;  105 : E->l  110 : N->i=E;  112 : E->l  116 : N->iE;  117 : E->(E)  118 : E->i  122 : E->i  126 : S->tfi(){NrE;};Sа  132 : N->iE;  133 : E->(E)  134 : E->l  138 : E->l  142 : S->m{NrE;};  144 : N->dti;N  148 : N->dti;N  152 : N->dti;N  156 : N->dti;N  160 : N->i();N  164 : N->i=E;N  166 : E->l  168 : N->i=E;N  170 : E->lM  171 : M->vE  172 : E->lM  173 : M->vE  174 : E->l  176 : N->s(KcK){N}e{N};N  178 : K->i  180 : K->i  183 : N->i=E;  185 : E->iM  186 : M->vE  187 : E->l  192 : N->i=E;  194 : E->i  198 : N->i=E;N  200 : E->i(W)  202 : W->i  205 : N->i=E;N  207 : E->i(W)M  209 : W->l  211 : M->vE  212 : E->l  214 : N->iE;N  215 : E->(E)  216 : E->i  219 : N->i=E;N  221 : E->i(W)  223 : W->i,W  225 : W->i  228 : N->iE;N  229 : E->(E)  230 : E->i  233 : N->i();  238 : E->l |

Листинг Е.1 – Фрагмент трассировки и дерево разбора

**ПРИЛОЖЕНИЕ Ж**

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib E:\unic\courseKPO\Project1\Debug\SGVLIB.lib  ExitProcess PROTO :DWORD  \_exp PROTO :DWORD  \_pow PROTO :DWORD, :DWORD  \_printInt PROTO :DWORD  \_printStr PROTO :DWORD  \_printBool PROTO :DWORD  \_printChar PROTO :DWORD  \_pause PROTO  .stack 4096  .CONST  null\_division BYTE 'ERROR: DIVISION BY ZERO', 0  OVER\_FLOW BYTE 'ERROR: OVERFLOW', 0  true BYTE "true", 0  false BYTE "false", 0  empty\_string BYTE 0  L1 BYTE "hello world", 0  L2 DWORD 2  L3 DWORD 2  L4 DWORD 2  L5 BYTE "Результат:", 0  L6 DWORD 2  L7 BYTE "it true", 0  L8 BYTE "it false", 0  L9 BYTE "hello", 0  L10 BYTE "done", 0  L11 DWORD 2  L12 DWORD 1  L13 DWORD 2  L14 DWORD 3  L15 DWORD 1  L16 DWORD 3  L17 DWORD 5  L18 DWORD 0  .DATA  tsa DWORD 0  tssts DWORD empty\_string  saa DWORD empty\_string  sab DWORD 0  mainx DWORD 0  mainy DWORD 0  mainz DWORD 0  mainksa DWORD empty\_string  .CODE  ts PROC tsz:DWORD, tsx:DWORD  push offset L1  pop tssts  push L2  push L3  pop eax  pop ebx  add eax, ebx  push eax  push L4  pop eax  pop ebx  mul ebx  push eax  pop eax  cmp eax, 255  jg overflow  cmp eax, 0  jl overflow  mov tsa, eax  push offset L5  call \_printStr  push tssts  call \_printStr  push tsa  call \_printInt  push tsx  push tsz  call \_pow  push eax  pop eax  cmp eax, 255  jg overflow  cmp eax, 0  jl overflow  mov tsa, eax  push tsa  jmp local0  local0:  pop eax  ret  ts ENDP  sa PROC sak:DWORD  push L6  push sak  call \_pow  push eax  pop eax  cmp eax, 255  jg overflow  cmp eax, 0  jl overflow  mov sab, eax  push sab  call \_printInt  push sak  pop eax  cmp eax, 255  jg overflow  cmp eax, 0  jl overflow  mov sab, eax  mov eax, sab  cmp eax, sak  jz ifi1  jnz else1  ifi1:  push offset L7  pop saa  jmp ifEnd1  else1:  push offset L8  pop saa  ifEnd1:  push saa  call \_printStr  push OFFSET saa  jmp local1  local1:  pop eax  ret  sa ENDP  sayhi PROC  push offset L9  call \_printStr  push OFFSET L10  jmp local2  local2:  pop eax  ret  sayhi ENDP  main PROC  call sayhi  push L11  pop eax  cmp eax, 255  jg overflow  cmp eax, 0  jl overflow  mov mainx, eax  push L12  push L13  push L14  pop eax  pop ebx  mul ebx  push eax  pop eax  pop ebx  add eax, ebx  push eax  pop eax  cmp eax, 255  jg overflow  cmp eax, 0  jl overflow  mov mainy, eax  mov eax, mainx  cmp eax, mainy  jz ifi2  jnz else2  ifi2:  push mainy  push L15  pop ebx  pop eax  sub eax, ebx  push eax  pop eax  cmp eax, 255  jg overflow  cmp eax, 0  jl overflow  mov mainx, eax  jmp ifEnd2  else2:  push mainy  pop eax  cmp eax, 255  jg overflow  cmp eax, 0  jl overflow  mov mainx, eax  ifEnd2:  push mainx  call sa  push eax  pop mainksa  push L16  call \_exp  push eax  push L17  pop eax  pop ebx  add eax, ebx  push eax  pop eax  cmp eax, 255  jg overflow  cmp eax, 0  jl overflow  mov mainz, eax  push mainz  call \_printInt  push mainy  push mainx  call ts  push eax  pop eax  cmp eax, 255  jg overflow  cmp eax, 0  jl overflow  mov mainz, eax  push mainz  call \_printInt  push L18  jmp theend  jmp theend  SOMETHINGWRONG::  push offset null\_division  call \_printStr  jmp THEEND  THEEND:  push eax  call ExitProcess  overflow::  push offset OVER\_FLOW  call \_printStr  main ENDP  end main |

Листинг Ж.1 – Сгенерированный код ассемблера