Оглавление

[Введение 4](#_Toc533046684)

[Глава 1. Спецификация языка программирования 5](#_Toc533046685)

[1.1 Характеристика языка программирования 5](#_Toc533046686)

[1.2 Алфавит языка 5](#_Toc533046687)

[1.3 Применяемые сепараторы 5](#_Toc533046688)

[1.4 Применяемые кодировки 6](#_Toc533046689)

[1.5 Типы данных 6](#_Toc533046690)

[1.6 Преобразование типов данных 7](#_Toc533046691)

[1.7 Идентификаторы 7](#_Toc533046692)

[1.8 Литералы 7](#_Toc533046693)

[1.9 Объявление данных 7](#_Toc533046694)

[1.10 Инициализация данных 7](#_Toc533046695)

[1.11 Инструкции языка 8](#_Toc533046696)

[1.12 Операции языка 8](#_Toc533046697)

[1.13 Выражения и их вычисление 8](#_Toc533046698)

[1.14 Конструкции языка 8](#_Toc533046699)

[1.15 Области видимости идентификаторов. 9](#_Toc533046700)

[1.16 Семантические проверки 9](#_Toc533046701)

[1.17 Распределение оперативной памяти на этапе выполнения 9](#_Toc533046702)

[1.18 Стандартная библиотека и её состав 10](#_Toc533046703)

[1.19 Ввод и вывод данных 10](#_Toc533046704)

[1.20 Точка входа 10](#_Toc533046705)

[1.21 Препроцессор 10](#_Toc533046706)

[1.22 Соглашения о вызовах 10](#_Toc533046707)

[1.23 Объектный код 11](#_Toc533046708)

[1.24 Классификация сообщений транслятора 11](#_Toc533046709)

[1.25 Контрольный пример 11](#_Toc533046710)

[Глава 2. Структура транслятора 12](#_Toc533046711)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 12](#_Toc533046712)

[2.2 Перечень входных параметров транслятора 12](#_Toc533046713)

[2.3 Протоколы, формируемые транслятором 12](#_Toc533046714)

[Глава 3 Разработка лексического анализатора 14](#_Toc533046715)

[3.1 Структура лексического анализатора 14](#_Toc533046716)

[3.2 Контроль входных символов 14](#_Toc533046717)

[3.3 Удаление избыточных символов 14](#_Toc533046718)

[3.4 Перечень ключевых слов 15](#_Toc533046719)

[3.5 Основные структуры данных 16](#_Toc533046720)

[3.6 Структура и перечень сообщений лексического анализатора 16](#_Toc533046721)

[3.7 Принцип обработки ошибок 16](#_Toc533046722)

[3.8 Параметры лексического анализатора 16](#_Toc533046723)

[3.9 Алгоритм лексического анализа 16](#_Toc533046724)

[3.10 Контрольный пример 16](#_Toc533046725)

[Глава 4. Разработка синтаксического анализатора 17](#_Toc533046726)

[4.1 Структура синтаксического анализатора 17](#_Toc533046727)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 17](#_Toc533046728)

[4.3 Построение конечного магазинного автомата 18](#_Toc533046729)

[4.4 Основные структуры данных 19](#_Toc533046730)

[4.5 Описание алгоритма синтаксического разбора 19](#_Toc533046731)

[4.6 Структура и перечень сообщений синтаксического анализатора 19](#_Toc533046732)

[4.7 Параметры синтаксического анализатора и режимы его работы 20](#_Toc533046733)

[4.8 Принцип обработки ошибок 20](#_Toc533046734)

[4.9 Контрольный пример 20](#_Toc533046735)

[Глава 5. Разработка семантического анализатора 21](#_Toc533046736)

[5.1 Структура семантического анализатора 21](#_Toc533046737)

[5.2 Функции семантического анализатора 21](#_Toc533046738)

[5.3 Структура и перечень сообщений семантического анализатора 21](#_Toc533046739)

[5.4 Принцип обработки ошибок 22](#_Toc533046740)

[Глава 6. Преобразование выражений 23](#_Toc533046741)

[6.1 Выражения, допускаемые языком 23](#_Toc533046742)

[6.2 Польская запись и принцип ее построения 23](#_Toc533046743)

[6.3 Программная реализация обработки выражений 23](#_Toc533046744)

[6.4 Контрольный пример 23](#_Toc533046745)

[Глава 7. Генерация кода 24](#_Toc533046746)

[7.1 Структура генератора кода 24](#_Toc533046747)

[7.2 Представление типов данных в оперативной памяти 24](#_Toc533046748)

[7.3 Статическая библиотека 25](#_Toc533046749)

[7.4 Особенности алгоритма генерации кода 25](#_Toc533046750)

[7.5 Контрольный пример 25](#_Toc533046751)

[Глава 8. Тестирование транслятора 26](#_Toc533046752)

[8.1 Тестирование проверки на допустимость символов 26](#_Toc533046753)

[8.2 Тестирование лексического анализатора 26](#_Toc533046754)

[8.3 Тестирование синтаксического анализатора 26](#_Toc533046755)

[8.4 Тестирование семантического анализатора 26](#_Toc533046756)

[Приложение А 27](#_Toc533046757)

[Приложение Б 29](#_Toc533046758)

[Приложение В 33](#_Toc533046759)

[Приложение Г 34](#_Toc533046760)

[Приложение Е 35](#_Toc533046761)

[Приложение Ж 37](#_Toc533046762)

[Заключение 41](#_Toc533046763)

[Литература 42](#_Toc533046764)

# Введение

Задачей данного курсового проекта является разработка собственного языка программирования и компилятора для него. Язык называется GDV-2022. Написание компилятора будет осуществляться на языке C++.

Для выполнения курсового проекта были поставлены следующие задачи:

* Спецификация языка программирования
* Структура транслятора
* Разработка лексического анализатора
* Разработка синтаксического анализатора
* Разработка семантического анализатора
* Вычисление выражений
* Генерация кода на языке C++
* Контрольный пример
* Тестирование транслятора

# 1 Спецификация языка программирования

## 1.1 Характеристика языка программирования

Язык программирования GDV-2022 – это процедурный, универсальный, строго типизированный, не С-подобный, компилируемый язык. Не является объектно-ориентированным.

## 1.2 Алфавит языка

Алфавит языка GDV-2022 основывается на таблице ASCII, представленной в таблице 1.1. Используемые в алфавите символы: **[a … z], [A … Z], [0 … 9]**, **[А … Я], [а … я]** спецсимволы: **() , ; :`'** , а также символы пробела, табуляции и перевода строки.

Таблица 1.1. Таблица кодировок ASCII



## **1.3 Применяемые сепараторы**

Применяемые сепараторы в языке программирования GDV-2022 описаны в таблице 1.2.

Таблица 1.2. Сепараторы языка GDV-2022.

|  |  |
| --- | --- |
| Сепараторы | Назначение сепаратора |
| { … } | Для блока функций |
| ( … ) | Для фактических или формальных параметров функции, а также для приоритета операций |
| ‘пробел’, ‘табуляция’ | Для разделения цепочек (не допускается в названиях идентификаторов и ключевых слов) |
| , | Для разделения параметров функции, цикла For |
| &, |, ~ | Побитовые операции |
| ; | Для разделения программных инструкций |
| = | Оператор присваивания |
| ? | Условный оператор |
| \n | Символ перехода на новую строку |
| <, >, ==, != | Операторы сравнения |

## 1.4 Применяемые кодировки

Для написания исходного кода на языке программирования GDV-2022 используется кодировка Windows-1251.

Таблица 1.3. Кодировка Windows-1251

## 1.5 Типы данных

Язык GDV-2022 позволяет использовать 3 типа данных для переменных и функций: целочисленный (num), символьный(symb) и числа с плавающей точкой(float). А также есть особый тип данных для функции — action. Описание этих типов данных приведено в таблице 1.4.

Таблица 1.4. Типы данных языка GDV-2022

|  |  |
| --- | --- |
| Тип данных | Описание |
| num | Целочисленный четырехбайтный тип данных. Значения по умолчанию нет |
| symb | Символьный однобайтовый тип данных. Относительно является целочисленным типом данных, значения колеблются от -128 до 127 |
| float | Тип данных, отвечающий за числа с плавающей точкой. Занимает 4 байта в памяти |
| action | Особый тип данных функции, который указывает, что данная функция является процедурой, которая не возвращает никакого значения |

## 1.6 Преобразование типов данных

Преобразование типов данных происходит неявно. Так как все типы данных языка GDV-2022 можно отнести к численным, их взаимодействию особо ничего не мешает, но все-таки бывают такие случаи, где без приведения типов не обойтись. При вызове функций идентификаторы и литералы будут приводится к типу параметров функций. Также при использовании побитовых операций, тип данных float будет приведен к типу данных num. При использовании цикла For по типу данных параметра, определяющему шаг, будет определен тип данных, к которому будут приведены первые два параметра цикла.

Приведение в языке GDV-2022 позволяет не ограничивать возможности типа данных float, то есть использовать побитовые операции, позволяет определить, с чем работает цикл For, и позволяет беспрепятственно использовать функцию, передавая в параметры переменные, литералы любого типа данных.

## 1.7 Идентификаторы

При создании идентификатора можно использовать буквы латинского алфавита как верхнего, так и нижнего регистра, а также можно использовать цифры и символ нижнего подчеркивания. Максимальная длина идентификатора – 50 символов. При превышении длины, идентификаторы усекаются до длины, равной 50 символов. Идентификаторы не должны совпадать с ключевыми словами и с другими идентификаторами, созданных в одной и той же области видимости.

## 1.8 Литералы

В языке GDV-2022 предусмотрены 4 вида литералов: целочисленные и строковые, символьные и чисел с плавающей точкой. Краткое описание литералов приведено в таблице 1.5.

Таблица 1.5 Литералы

|  |  |
| --- | --- |
| Литерал | Пояснение |
| Целочисленный | Максимально допустимое значение 231-1. Минимально допустимым является -231-1. При выходе за пределы допустимости выводится соответствующая ошибка. В случае отрицательного значения используется знак минус. |
| Символьный | Используются символы из кодировки Windows-1251. При присвоении целочисленного значения или значения числа с плавающей точкой будет присвоен символ с соответствующей кодировкой. Если значение выше или ниже, ошибки не будет, так как система сама решит проблему с переполнением. |
| Числа с плавающей точкой | Максимальное значение — 3.402823466e+38F. Минимальное значение — 1.175494351e-38F. При выходе за пределы вызывается ошибка. В случае отрицательного значения используется знак минус. Литерал должен иметь в записи символ точки. |
| Строковый | Используются символы кодировки ASCII. Максимальный размер строки — 255. Используется только с потоком вывода console. |

## 1.9 Объявление данных

В языке программирования GDV-2022 необходимо объявить переменную до ее использования. Есть два типа объявления: с явной типизацией и с неявной. При явной мы должны сразу указывать тип данных переменной. При неявной типизации язык сам вычислит тип данных переменной, но для этого сразу нужно использовать оператор '=>', после которого должен идти либо идентификатор, либо литерал, иначе будет ошибка.

Для определения типа переменной нужно использовать ключевое слово is после идентификатора, а после него сам тип данных (num, symb, float).

Пример объявления переменной целочисленного типа: n is num;

Пример объявления переменной символьного типа: s is symb;

Пример объявления переменной типа числа с плавающей точкой: f is float;

Пример объявления переменной целочисленного типа с неявной типизацией:

n => 2;

Пример объявления переменной символьного типа с неявной типизацией:

s => ‘s’;

Пример объявления переменной типа числа с плавающей точкой с неявной типизацией: f => .2.

Объявление функции схоже с объявлением обычной переменной, также используется оператор is после идентификатора, после него уже следует ключевое слово foo, что обозначает, что это функция, далее идет открывающая круглая скобочка, после которой идет инициализация параметров. Инициализация параметров проходит также, как инициализация обычных переменных. Чтобы инициализировать несколько параметров, нужно сделать несколько инициализаций и разделить их запятыми. После инициализации будет следовать закрывающая кругла скобочка. Далее будет определен тип данных возвращаемого значения этой функции. Для определения используется, как правило, оператор is после скобочки. Типы данных функции совпадают с типами данных переменных, только тут добавляется дополнительный тип данных action. Тип данных указывается после оператора is. Далее будет идти тело функции, заключенное в фигурные скобки.

Пример объявления функции целочисленного типа:

Sum is foo(a is num, b is num) is num

{

return sum(a, b);

}

Также язык программирования GDV-2022 позволяет объявлять шаблонные функции. Шаблонная функция в языке GDV-2022 — функция, объявленная не в глобальной области видимости, а в другой, допустим в другой функции или в main. Их объявление такое же как и объявление обычной функции, только типа данных action нет и после закрывающей фигурной скобки будет идти символ точки с запятой.

## 1.10 Инициализация данных

Для инициализации переменных используются два оператора: = и =>. Их механика инициализации схожа, а именно слева от операторов должен быть идентификатор, а справа — литерал, либо значение, либо выражение. Только если оператор =, то справа может быть любое выражение, а, если оператор =>, справа могут быть выражения, начинающиеся с идентификатора или же с литерала.

## 1.11 Инструкции языка

Инструкции языка программирования GDV-2022 представлены в таблице 1.6

Таблица 1.6. Инструкции языка GDV-2022.

|  |  |
| --- | --- |
| Инструкция | Запись на языке GDV-2022 |
| Объявление переменной с явной типизацией | <идентификатор> is <тип данных>; |
| Объявление и инициализация переменной с неявной типизацией | <идентификатор> => <выражение>; |
| Вывод данных: | console([<список параметров>]); |
| Вызов подпрограммы | <идентификатор функции> ([<список параметров>]); |
| Перевод строки | console(); |
| Присваивание | <идентификатор> = <выражение>; |

## 1.12 Операции языка

В языке программирования GDV-2022 присутствуют операции, описанные в таблице 1.7. Операции, заключенные в (..) (круглые скобки) имеют наивысший приоритет, равный 4. Операция ~ (логическое не) имеет приоритет 3. Операция & (логического умножения) имеет приоритет, равный 2. Операция | (логическое сложение) имеет низший приоритет, равный 1.

Операции логического умножения и логического сложения являются бинарными, то есть требуют два операнда. Операция логического не является унарной, то есть требует только один операнд.

Таблица 1.7. Операции языка GDV-2022.

|  |  |
| --- | --- |
| Операции | Операторы |
| Побитовые | | (логическая сложение)  & (логическое умножение)  ~ (логическое не) |

## 1.13 Выражения и их вычисление

Предусмотрены следующие правила составления выражений:

* Рассматриваются слева направо.
* Для изменения приоритета операции используются круглые скобки ()
* Каждое выражение должно заканчиваться сепаратором

В выражения можно использовать комбинацию всех типов данных, кроме типа

данных процедуры action. В выражениях можно использовать все виды операций языка GDV-2022. Также есть специальные функции из статической библиотеки sum(), mult(), division(), minus() для выполнения арифметических операций.

Выражения можно использовать после операции присваивания и после оператора возвращения значения return.

## 1.14 Конструкции языка

Программные конструкции языка программирование GDV-2022 приведены в таблице 1.8.

Таблица 1.8. Программные конструкции языка

|  |  |
| --- | --- |
| Конструкции | Представление в языке |
| Точка входа в программу | main {…} |
| Пользовательская функция | <идентификатор> is foo(<идентификатор> is <тип данных>, …) is <тип данных>  {  …  return <выражение>;  } |
| Процедура | <идентификатор> is foo(<идентификатор> is <тип данных>, …) is action  {  …  } |
| Шаблонная функция | <идентификатор> is foo(<идентификатор> is <тип данных>, …) is <тип данных>  {  …  return <выражение>;  }; |
| Лямбда-функция в цикле For | <идентификатор> => {  …  } |

## 1.15 Области видимости идентификаторов.

Область видимости построена по принципу С++. Все идентификаторы должны быть вызваны из текущей области видимости или областей видимости, в которых они находятся. Дополнительных указателей не требуется. Для создания искусственной области видимости используются фигурные скобки {…}.

## 1.16 Семантические проверки

Перечень семантических проверок, предусмотренных языком программирования GDV-2022, приведен в таблице 1.9.

Таблица 1.9. Семантические проверки

|  |  |
| --- | --- |
| № | Проверка |
| 1 | Единственность точки входа |
| 2 | Превышение размера целочисленных, строковых и плавающих литералов |
| 3 | Правильность выражений |
| 4 | Правильность передаваемых в функцию параметров: количество, типы |
| 5 | Переопределение идентификаторов |
| 6 | Использование идентификаторов без их объявления |

## 1.17 Распределение оперативной памяти на этапе выполнения

Все переменные размещаются в стеке. Шаблонная функция в куче.

## 1.18 Стандартная библиотека и её состав

В языке GDV-2022 присутствует стандартная библиотека, которая подключает некоторые функции автоматически при трансляции исходного кода в язык С++, а некоторые функции можно подключить с помощью ключевого слова @import. Содержимое библиотеки и описание функций представлено в таблице 1.10.

Таблица 1.10. Стандартная библиотека языка GDV-2022

|  |  |
| --- | --- |
| Функция | Описание |
| sum is foo(a is num, b is num) is num; | Математическая функция. Заменяет арифметическое суммирование. Подключается автоматически. Тип параметров и возвращаемого значения целочисленный |
| sum is foo(a is float, b is float) is float; | Математическая функция. Заменяет арифметическое суммирование. Подключается автоматически. Тип параметров и возвращаемого значения плавающее значение |
| minus is foo(a is num, b is num) is num; | Математическая функция. Заменяет арифметическую разницу. Подключается автоматически. Тип параметров и возвращаемого значения целочисленный |
| minus is foo(a is float, b is float) is float; | Математическая функция. Заменяет арифметическую разницу. Подключается автоматически. Тип параметров и возвращаемого значения плавающее значение |
| mult is foo(a is num, b is num) is num; | Математическая функция. Заменяет арифметическое произведение. Подключается автоматически. Тип параметров и возвращаемого значения целочисленный |
| mult is foo(a is float, b is float) is float; | Математическая функция. Заменяет арифметическое произведение. Подключается автоматически. Тип параметров и возвращаемого значения плавающее значение |
| division is foo(a is num, b is num) is num; | Математическая функция. Заменяет арифметическое деление. Подключается автоматически. Тип параметров и возвращаемого значения целочисленный |
| division is foo(a is float, b is float) is float; | Математическая функция. Заменяет арифметическое деление. Подключается автоматически. Тип параметров и возвращаемого значения плавающее значение |
| pow is foo(number is num, power is num) is num; | Математическое функция. Выполняет роль возведения числа в степень. Подключается с помощью ключевого слова @import. Тип параметров и возвращаемого значения целочисленный |
| symb\_to\_num is foo(s is symb) is num; | Символьная функция. Переводит из символьного литерала в число. Если значение литерала число, то вернет это же число, только делая его целочисленным литералом. Подключается с помощью ключевого слова @import. Тип параметра символьный. Тип возвращаемого значения целочисленный |
| abs is foo(number is num) is num; | Математическая функция. Возвращает модуль от числа, переданного в параметр. Подключается при помощи ключевого слова @import. Типы данных параметра и возвращаемого значения совпадают и раны num |

## 1.19 Ввод и вывод данных

В языке GDV-2022 вывод данных осуществляется с помощью оператора console. В качестве аргумента могут выступать литералы, вызываемые функции и идентификаторы. В операторе console может быть неограниченное количество передаваемых аргументов. Ввод в данном языке программирования не предусмотрен.

## 1.20 Точка входа

Точкой входа в программе является ключевое слово “main”. Точка входа может отсутствовать. Но более чем одной точки входа быть не может.

## 1.21 Препроцессор

В языке GDV-2022 препроцессор не предусмотрен.

## 1.22 Соглашения о вызовах

Используется соглашение \_cdecl, то есть все параметры передаются в стек справа налево, память высвобождает сама функция.

## 1.23 Объектный код

Объектный код реализован на основе языка программирования С++.

## 1.24 Классификация сообщений транслятора

Классификация сообщений транслятора приведена в таблице 1.11.

Таблица 1.11. Сообщения транслятора

|  |  |
| --- | --- |
| Кодысообщений | Принадлежность |
| 0 – 200 | Системные ошибки, ошибки лексического анализа |
| 600-800 | Ошибки, семантического анализа, синтаксического анализа |
| 201-599, 801-999 | Зарезервированные коды ошибок |

## 1.25 Контрольный пример

Исходный код контрольного примера представлен ниже.

|  |
| --- |
| Fact is foo(a is num) is num  {  res => 1;    For(1, a, 1, el =>{  res = mult(res, el);  });  return res;  }  main  {  a => Fact(3);  (a == 6) ?  Truth  {  console(`a равно 6`);  }    return 0;  } |

# Глава 2. Структура транслятора

## 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Транслятор языка программирования GDV-2022 состоит из следующих частей:

Лексический анализатор – часть транслятора, на котором выполняется лексический анализ. На данном этапе распознаётся правильность составления лексем и идентификаторов.

Синтаксический анализатор – часть транслятора, на которой выполняется синтаксический анализ. Проверяется правильность расположения идентификаторов и ключевых слов в исходном коде. Для того, чтобы провести данную операцию используются таблица лексем и идентификаторов.

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть исходный код проверяется на наличие ошибок. Входными данными являются таблица лексем и идентификаторов.

Генератор кода – часть транслятора, выполняющая генерацию кода на языке С++ на основе полученных данных на предыдущих этапах трансляции. На вход генератора подаются таблица лексем и таблица идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

## 2.2 Перечень входных п**а**раметров транслятора

Для формирования файлов с результатами работы лексического, синтаксического и семантического анализаторов используются входные параметры транслятора, которые приведены в таблице 2.1.

Таблица 2.1. Входные параметры транслятора языка GDV-2022

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<имя in-файла> | Файл с исходным кодом на языке программирования GDV-2022, имеющий расширение .txt | Не предусмотрено |
| -log:<имя log-файла> | Файл, содержащий вывод протокола работы программы. | Значение по умолчанию:  <имя in-файла>.log |
| -out:<имя out-файла> | Выходной файл – результат работы транслятора. Содержит исходный код на языке ассемблера. | Значение по умолчанию:  <имя in-файла>.out |

## 2.3 Протоколы, формируемые транслятором

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

Таблица 2.2 Протоколы, формируемые транслятором языка GDV-2022

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" log.txt | Файл с протоколом работы транслятора языка программирования GDV-2022. Содержит таблицу лексем. |
| Выходной файл, заданный параметром "-out:" out.cpp | Результат работы программы – файл, содержащий исходный код на языке C++. |
| LT.txt | Сформированная таблица лексем |
| IT.txt | Сформированная таблица идентификаторов |
| SNT.txt | Содержит протокол работы синтаксического анализатора и дерево разбора, полученные на этапе синтаксического анализа. |

# Глава 3 Разработка лексического анализатора

## 3.1 Структура лексического анализатора

Лексический анализатор – часть компилятора, которая выполняет лексический анализ. На данном этапе распознаётся правильность составления лексем и идентификаторов языка. Для работы лексический анализатор использует исходный код на языке GDV-2022. В итоге будут сформированы таблица лексем и таблица идентификаторов. Также во время лексического анализа проверяется много синтаксических и семантических ошибок. А также тут строятся области видимости для идентификаторов. Сделано при помощи такой структуры данных, как стек. Во время выполнения лексического анализа некоторые элементы могут заменятся, примером является оператор =>. Допустим выражение a => 1 заменяется на a is num = 1.

Структура лексического анализатора представлена на рисунке 3.1.

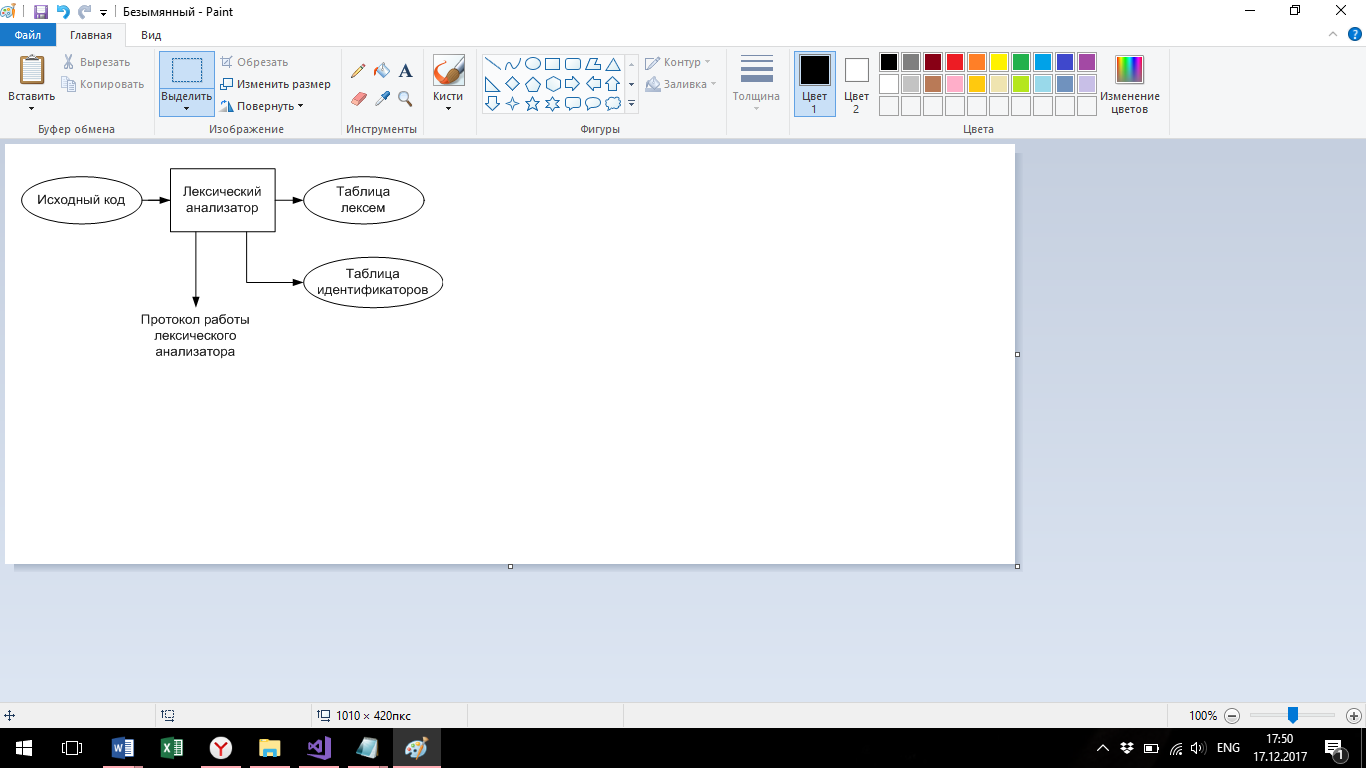


Рис. 3.1 Структура лексического анализатора

## 3.2 Контроль входных символов

При передаче исходного кода в лексический анализатор, все символы разделяются по определённым категориям, для дальнейшего использования. Категории входных символов представлены в таблице 3.1.

Таблица 3.1 Соответствие символов и их значений в таблице

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

## 3.3 Удаление избыточных символов

В языке программирования GDV-2022 предусмотрено удаление избыточных символов. Удаление избыточных символов происходит на этапе формирования слов, которые поступят на вход лексического анализатора. Пробелы и символы табуляции не участвуют в формировании слов, если только они не внутри строкового или символьного литерала.

## 3.4 Перечень ключевых слов

Соответствие ключевых слов, сепараторов, символов операций с лексемами приведено в таблице 3.2.

Таблица 3.2 Соответствие ключевых слов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Конструкция | Лексема | Примечание |
| num  float  symb  ref num  ref float  ref symb | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 50 символов. |
| Литерал | l | Литерал любого доступного типа. |
| foo | f | Объявление функции. |
| action | a | Ключевое слово для процедур – функций, не возвращающих значения. Указывается после второго оператора is. |
| return | r | Выход из функции типа action. Возвращение значения из функций других типов. |
| main | m | Точка входа в программу. |
| is | s | Оператор, позволяющий присвоить идентификатору любой тип данных. Используется также для проверки переменной на тип. |
| console | c | Поток вывода. |
| break | b | Оператор для выхода из цикла For |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| { | { | Начало блока/тела функции. |
| } | } | Закрытие блока/тела функции. |
| ( | ( | Передача параметров в функцию, приоритет операций. |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | = | Знак присваивания. |
| &, | | v | Бинарные операции. |
| ~ | ~ | Унарная операция. |
| For | p | Вызов цикла For. |
| skip | z | Оператор, позволяющий пропустить итерацию в цикле For. |
| >, <, ==, != | V | Операторы сравнения. |
| ? | ? | Условный оператор. |
| Truth | T | Блок, куда перейдет действие, если выражение в условном операторе истинно. |
| Lie | L | Блок, куда перейдет действие, если выражение в условном операторе ложно. |

Реализации графов переходов находятся в приложении А.

Пример реализованного конечного автомата ключевого слова main языка GDV-2022 представлен на рисунке 3.2.

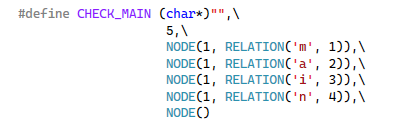


Рис. 3.2 Реализация конечного автомата для ключевого слова main

## 3.5 Основные структуры данных

Основные структуры данных языка GDV-2022 являются таблица лексем и таблица идентификаторов.

За таблицу лексем отвечает структура данных, называемая LexTable. У нее есть поле maxsize, которое отвечает за максимальное количество элементов в таблице. Следующее поле size, оно уже отвечает за количество элементов, находящихся внутри таблицы. И главное хранилище элементов — это поле table, которое является массивом элементов типа LT::Entry.

За элемент массива, который находится внутри таблицы лексем, отвечает тип данных Entry. Это структура данных, находящаяся внутри пространства имен LT. Тут есть такие поля, как lexema, которое отвечает за значение лексемы; как view, которое отвечает за то, какая бинарная операция используется вместо лексемы 'v'; как поле sn, которое отвечает за то, какая строка в исходном коде соответствует данной лексеме; как поле idxTI, которое используется только идентификаторами или же лексемами для отображения индекса данной лексемы в таблице идентификаторов.

За таблицу идентификаторов отвечает структура данных IdTable. Ее поля абсолютно такие же, как и уs таблицы лексем. Только вот элементом массива table является структура Entry, находящаяся внутри пространства имен IT. У нее есть поля: idxfirstLE, countParams, id, iddatatype, idtype, hasValue, isRef, isFromStatic, params, needToInt, value. Поле idxfirstLE отображает индекс данного идентификатора, лексемы в таблице лексем. Поле countParams используется только для идентификаторов, отвечающих за функции, отображает количество параметров данной функции. Поле id — это строка, в которой содержится имя идентификатора. Поле iddatatype — это перечисление, отвечающее за тип данных идентификатора, может принимать значения IT::ACTION, IT::NUM, IT::SYMB, IT::FLOAT, IT::STR. Поле idtype также является перечислением, только оно уже отвечает за то, какого типа данный идентификатор — функция, литерал, параметр, переменная. Поле hasValue — это булевая переменная, имеет значение true, если данный идентификатор инициализирован, false, если нет. Поле isRef — также булевая переменная, только она уже отвечает за то, чтобы проверять данный идентификатор, который является параметром, передается по ссылке. Поле isFromStatic также является булевой переменной, оно уже отвечает за то, чтобы понять, что данный идентификатор из стандартной библиотеки. Поле params — это вектор, в котором хранятся типы данных параметров функции. Используется для проверки передаваемых параметров в вызываемую функцию. Поле value — является объединением, в котором есть поля для хранения значений целочисленного, вещественного, символьного и строкового литералов.

Основные структуры данных приведены в приложении А.

## 3.6 Структура и перечень сообщений лексического анализатора

Структура сообщений содержит информацию о номере сообщения, номер строки, где было вызвано сообщение в исходном коде, информацию об ошибке. Перечень сообщений представлены в таблице 3.3.

Таблица 3.3. Перечень ошибок лексического анализатора

|  |  |
| --- | --- |
| Код сообщения | Содержание сообщения |
| 113 | Ошибка при создании таблицы лексем. Превышена емкость таблицы лексем |
| 114 | Ошибка при добавлении лексемы в таблицу. Таблица лексем заполнена |
| 115 | Ошибка при получении лексемы из таблицы. Недопустимый номер лексемы |

## 3.7 Принцип обработки ошибок

В случае возникновения ошибок происходит их протоколирование и описание в командной строке с номером ошибки и сообщением.

## 3.8 Параметры лексического анализатора

Параметром лексического анализатора является очередь из структур, полями которых являются лексемы в исходном коде, полученные на этапе проверки кода на допустимость символов.

## 3.9 Алгоритм лексического анализа

Алгоритм работы лексического анализа заключается в распознавании и разборе цепочек исходного кода на основе конечных автоматов, а также заполнение таблиц идентификаторов и лексем. Работу конечного автомата можно показать с помощью графа переходов. Пример графа для цепочки «main» приведен на рисунке 3.3.

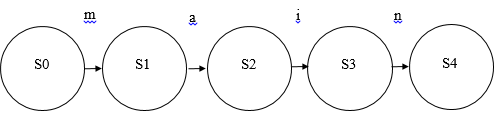


Рис 3.3 – Пример графа для цепочки main

## 3.10 Контрольный пример

Контрольный пример в виде таблиц лексем и идентификаторов представлен в приложении Б.

# Глава 4. Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора

Синтаксический анализ – фаза компилятора, которая выполняется после лексического анализа. В этой фазе будут распознаваться синтаксические конструкции. На вход синтаксического анализатора будет подаваться таблица лексем и таблица идентификаторов, а результатом работы будет дерево разбора.

## Контекстно-свободная грамматика, описывающая синтаксис языка

Синтаксис языка GDV-2022 описывается грамматикой типа 2 по иерархии Хомского:

G = <T, N, P, S>

T – множество терминальных символов (алфавит языка GDV-2022),

N – множество нетерминальных символов,

P – множество правил языка,

S – начальный символ грамматики, представленный нетерминальным символом «A».

Множество терминальных символов соответствует элементам, содержащимся в таблице лексем. Правила нетерминальных символов описаны в таблице 4.1

Таблица 4.1. Правила нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Нетерминалы | Назначение | Правила |
| A | Стартовый символ | iTA |
| iT |
| iW |
| iWA |
| iU=BXA |
| iU=BX |
| iUXA |
| iUX |
| i=BXA |
| i=BX |
| m{J}A |
| m{J} |
| m{G} |
| m{G}A |
| T | Инициализация и объявление функции | sfCU{G} |
| sfCU{J} |
| U | Определение типа данных | st |
| V | Тело функции типа Action | iUXV |
| iUX |
| i(FXV |
| i(FX |
| i(YXV |
| i(YX |
| iU=BXV |
| iU=BX |
| i=BXV |
| i=BX |
| {V}V |
| {V} |
| {}V |
| {} |
| cLXV |
| cLX |
| pOV |
| pO |
| (KVKY?MV |
| (KVKY?M |
| iTXV |
| iTX |
| rX |
| rXV |
| W | Инициализация функции типа action | sfCsa{V} |
| sfCsa{} |
| X | Точка с запятой | ; |
| Y | Закрывающая круглая скобка | ) |
| B | Выражения | l |
| i |
| i(FY |
| I(Y |
| ivB |
| lvB |
| I(FYvB |
| I(YvB |
| (BY |
| (BYvB |
| ~B |
| C | Конструкция параметров инициализируемой функции | (Y |
| (DY |
| D | Объявление параметров | iU |
| iU,D |
| E | Вызов функции | i(Y |
| i(FY |
| F | Аргументы вызываемой функции | i |
| l |
| i(FY |
| i(Y |
| i,F |
| l,F |
| i(FY,F |
| i(Y,F |
| G | Тело функции с возвращаемым значением | iUXG |
| iUXJ |
| i(FXG |
| i(FXJ |
| i(YXG |
| i(YXJ |
| iU=BXG |
| iU=BXJ |
| i=BXG |
| i=BXJ |
| {N}G |
| {N}J |
| {}G |
| {}J |
| cLXG |
| cLXJ |
| pOG |
| pOJ |
| (KVKY?MG |
| (KVKY?MJ |
| iTXG |
| iTXJ |
| H | Инициализация переменной | ist |
| I | Тело условного оператора | i(FYXJ |
| i(FYXI |
| i(FYX |
| i(YXJ |
| i(YXI |
| i(YX |
| iUXI |
| iUXJ |
| iUX |
| iU=BXI |
| iU=BXJ |
| iU=BX |
| i=BXI |
| i=BXJ |
| i=BX |
| {I}I |
| {I}J |
| {I} |
| {} |
| {}I |
| {}J |
| iTXI |
| iTXJ |
| iTX |
| cLXI |
| cLXJ |
| cLX |
| (KBKY?MI |
| (KBKY?MJ  (KBKY?M |
| rBX |
| rX |
| bX |
| bXI |
| zX |
| zXI |
| J | Возвращение значения функций | rBX |
| rX |
| K | Отвечает за все виды значений(функция, литерал, идентификатор) | i |
| l |
| i(Y |
| I(FY |
| M | Отвечает за блоки условного оператора | T{I} |
| T{I}L{I} |
| L{I} |
| T{} |
| L{} |
| T{}L{I} |
| T{I}L{} |
| T{}L{} |
| L | Тело оператора потока вывода | (FY |
| (Y |
| N | Отвечает за искусственную область видимости | i(FYXJ |
| i(FYXN |
| i(FYX |
| i(YXJ |
| i(YXN |
| i(YX |
| iUXN |
| iUXJ |
| iUX |
| iU=BXN |
| iU=BXJ |
| iU=BX |
| i=BXN |
| i=BXJ |
| i=BX |
| {I}N |
| {I}J |
| {I} |
| {} |
| {}N |
| {}J |
| iTXN |
| iTXJ |
| iTX |
| cLXN |
| cLXJ |
| cLX |
| (KBKY?MN |
| (KBKY?MJ  (KBKY?M |
| rBX |
| rX |
| bX |
| bXN |
| zX |
| zXN |
| O | Параметры цикла For | (K,K,K,i{P}YX |
| P | Тело цикла For | i(FYXP |
| i(FYX |
| i(YP |
| i(Y |
| iUXP |
| iUX |
| iU=BXP |
| iU=BX |
| i=BXP |
| i=BX |
| {P}P |
| {P} |
| {} |
| {}P |
| iTXP |
| iTX |
| cLXP |
| cLX |
| pOP |
| pO |
| (KVKY?MP |
| (KVKY?M |
| bX |
| bXP |
| zX |
| zXP |

## 4.3 Построение конечного магазинного автомата

Распознавателем грамматики является конечный автомат с магазинной памятью, который представляет собой семерку.Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

Таблица 4.2. Описание компонент магазинного автомата

|  |  |
| --- | --- |
| Компонента | Определение |
|  | Множество состояний автомата |
|  | Алфавит входных символов |
|  | Алфавит специальных магазинных символов |
|  | Функция переходов автомата |
|  | Начальное состояние автомата |
|  | Начальное состояние магазина автомата |
|  | Множество конечных состояний |

## 4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора представлены в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила и цепочки правил. Данные структуры представлены в приложении В.

## 4.5 Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора можно описать следующим образом:

* В магазин записывается стартовый символ.
* На основе полученной таблицы лексем формируется входная лента.
* Запускается автомат и выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке.
* Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется с ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала.
* Если в магазине встретился нетерминал, переходим к пункту 3.
* Если символ достиг символа дна стека, и лента в этот момент имеет символ дна стека, то синтаксический анализ выполнен успешно. Иначе генерируется ошибка.

## 4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен в таблице 4.3.

Таблица 4.3. Перечень ошибок синтаксического анализатора

|  |  |
| --- | --- |
| Код сообщения | Содержание сообщения |
| 650 | Ошибка в глобальной области! |
| 651 | Ошибка в выражении |
| 652 | Ошибка при построении контрукции параметров |
| 653 | Ошибки при указании типов данных для параметров |
| 654 | Ошибка при вызове функции |
| 655 | Ошибка в параметрах вызываемой функции |
| 656 | Ошибка в теле функции |
| 657 | Ошибка в инициализации переменных |
| 658 | Ошибка в использовании условного оператора |
| 659 | Ошибка при возвращении значения функции |
| 660 | Ошибка в при использовании все возможных типов значений |
| 661 | Ошибка в теле условного оператора |
| 662 | Ошибка при вызове оператора console(). Неверный синтаксис. console(Значение); |
| 663 | Ошибка в искусственной области видимости! |
| 664 | Ошибка в параметрах цикла |
| 665 | Ошибка в теле цикла |
| 666 | Ошибка при иницилизации функции |
| 667 | Ошибка в определении типа переменной/функции |
| 668 | Ошибка в теле функции типа Action"), |
| 669 | Ошибка при инициализации функции Action |
| 670 | Ошибка. Не хватает точки с запятой |
| 671 | Ошибка. Ожидалась закрывающая круглая скобка ‘)’ |

## 4.7 Параметры синтаксического анализатора и режимы его работы

Входными параметрами для синтаксического анализатора в языке программирования GDV-2022 являются таблица лексем и таблица идентификаторов.

## 4.8 Принцип обработки ошибок

Синтаксический анализатор перебирает все возможные правила и цепочки правила грамматики в целях поиска подходящего соответствия. Если ни одна из цепочек правила не подошла для рассматриваемой конструкции, то генерируется ошибка в соответствии с таблицей 4.3. Ошибка заносится в протокол.

## 4.9 Контрольный пример

Пример разбора исходного кода на языке программирования GDV-2022 синтаксическим анализатором представлен в приложении Г.

# Глава 5. Разработка семантического анализатора

## 5.1 Структура семантического анализатора

Семантический анализатор принимает на свой вход таблицы лексем, идентификаторов и результат работы синтаксического анализатора, то есть дерево разбора, и последовательно ищет необходимые ошибки. Также семантические проверки предусмотрены на этапе лексического анализа, а также на этапе генерации кода на язык С++.

## 5.2 Функции семантического анализатора

Семантический анализатор выполняет проверку на основе правил языка, описанных в п. 1.16. Он и есть та самая подпрограмма, которая занимается автоматическим приведением типов.

## 5.3 Структура и перечень сообщений семантического анализатора

Перечень сообщений семантического анализатора представлен в таблице 5.1.

Таблица 5.1. Перечень ошибок семантического анализатора

|  |  |
| --- | --- |
| Код сообщения | Содержание сообщения |
| 119 | Ошибка. В стандартной библиотеке GDV-2022 такой функции нет |
| 120 | Ошибка. Знак '-' может быть использован, только перед литералом! |
| 121 | Ошибка в синтаксисе! |
| 600 | Нет закрывающей кавычки |
| 601 | Найдено неопознанное слово |
| 602 | Использованна неинициализированная переменная |
| 603 | Некорректное объявление функции |
| 604 | При инициализации функции не хватает закрывающей скобки ')' |
| 605 | При инициализации функции невозможно узнать тип данных возвращаемого значения |
| 606 | Ошибка. Повторное объявление переменной! |
| 607 | Ошибка при присвоении значения. Левый аргумент операции присваивания должен быть идентификатором! |
| 608 | Невозможно определить тип переменной, инициализированной неявно! После оператора => должны быть либо идентификатор, либо литерал |
| 609 | Использована необъявленная переменная |
| 610 | Ошибка при вызове функции! Количество параметров не совпадает |
| 611 | При использовании лямбда-выражения el => {} не найдено ключевое слово For |
| 612 | Ошибка. В лямбда выражении в цикле For передаваемый параметр должен быть идентификатор |
| 613 | Ошибка. Неправильно употребленна лямбда функции. Она предназначена для цикла For(start, end, step, el => {}) |
| 614 | Ошибка. Параметр перед входящим в цикл For лямбда выражением должен быть идентификатором или литералом |
| 615 | Ошибка. Неправильная структура цикла For |
| 616 | Ошибка. Оператор break может быть использован только в теле цикла! |
| 617 | Ошибка. После имени функции ожидалась открывающая скобка '(' |
| 618 | Ошибка. Как ссылки можно инициализировать только параметры |
| 619 | Ошибка. Нельзя использовать литералы/функции в качестве передаваемого параметра, берущегося по ссылке |
| 620 | Синтаксическая ошибка. После имени функции ожидалась открывающая скобка '(' |
| 621 | Использован неверные управляющий символ(доступны только \\n, \\t |
| 622 | Ошибка. Не найдена пара для косой скобки '`' |
| 623 | Ошибка. Строковый литерал может использоваться только в качестве аргумента потока вывода |
| 624 | Ошибка. Превышено значение литерала типа num |
| 625 | Ошибка. Слишком маленькое значение литерала типа num |
| 626 | Ошибка в значение литерала типа float |
| 627 | Ошибка. Превышен размер строкового литерала |
| 628 | Ошибка. Функцию типа action нельзя использовать в выражениях или передавать в качестве параметра |
| 629 | Ошибка. Точка входа в программу main может быть описана лишь один раз |
| 630 | Ошибка. Функция типа action не может ничего возвращать |
| 631 | Ошибка. Функции, тип данных которых не action, должны возвращать значение |

## 5.4 Принцип обработки ошибок

Семантический анализатор, в случае возникновения ошибки, заносит её в протокол. Следующий этап трансляции не будет запущен при возникновении ошибки. Семантический анализ начинает проверки уже на стадии лексического анализа, если на этой стадии обнаружены семантические ошибки – программа завершит свою работу, оповестив пользователя, где и что произошло.

**5.5 Контрольный пример**

Обработка ошибок семантического анализатора представлена в п. 8.4.

# Глава 6. Преобразование выражений

## 6.1 Выражения, допускаемые языком

В языке программирования GDV-2022 выражения могут содержать вычисления целочисленных типов данных, а также допускаются вызов функций (возвращающих тип) внутри выражений. Приоритет операций представлен на таблице 6.1.

Таблица 6.1 Приоритет операция языка GDV-2022

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( ) | 4 |
| ~ | 3 |
| & | 2 |
| | | 1 |

## 6.2 Польская запись и принцип ее построения

Польская запись — довольно-таки полезный принцип разбора выражений. С помощью ее можно определять, как и когда будут вызываться определенные операции с их аргументами. Но так как язык GDV-2022 транслируется на С++, использование польской записи является излишним.

# Глава 7. Генерация кода

## 7.1 Структура генератора кода

Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. В соответствии с таблицей лексем строится выходной файл на языке С++, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода GDV-2022 представлена на рисунке 7.1.

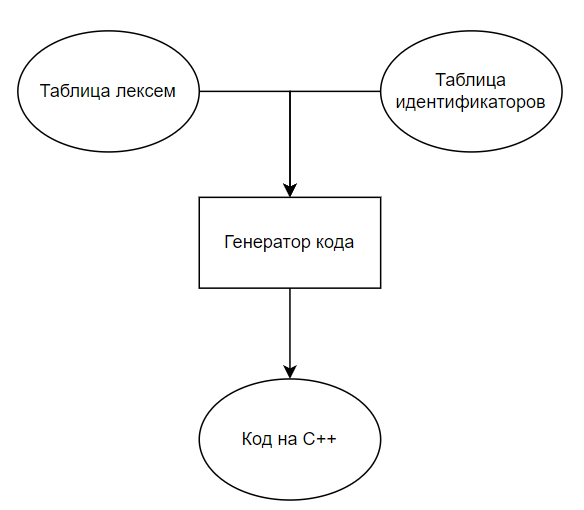


Рисунок 7.1 Структура генератора кода

## 7.2 Представление типов данных в оперативной памяти

Переменные, функции и все другие части кода будут написаны на языке С++ на своих местах, где они написаны на языке GDV-2022. Соответствие типов данных языка GDV-2020 с типами данных С++ представлено в таблице 7.1.

Таблица 7.1 Соответствия типов идентификаторов

|  |  |  |
| --- | --- | --- |
| Тип идентификатора | Тип идентификатора на языке C++ | Пояснение |
| num | int | Хранит четырехбайтовое целое число |
| symb | char | Хранит символ из кодировки Windows-1251 |
| float | float | Хранит четырехбайтовое вещественное число |

## 7.3 Статическая библиотека

В языке GDV-2022 предусмотрена статическая библиотека, которая содержит функции, написанные на языке C++, приведенные в таблице 7.2. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера.

Таблица 7.2 Статическая библиотека

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Параметры | Возвращаемое выражение | Пояснение |
| sum | int/float a, int/float b | a + b | Арифметическая сумма |
| mult | int/float a, int/float b | a \* b | Арифметическое произведение |
| division | int/float a, int/float b | a / b | Арифметическое деление |
| minus | int/float a, int/float | a – b | Арифметическая разность |
| For | int/float/char start, int/float/char end, int/float/char step, function<bool(int/float/char)> foo | Нет | Функция заменяющая цикл For. |

## 7.4 Особенности алгоритма генерации кода

Алгоритм генерации объектного кода выглядит следующим образом:

* С помощью цикла for идем по таблице лексем.
* С помощью условного оператора switch разыскиваем лексемы, соответствие которым есть на языке С++.
* Если лексема равна i, нужно понять — это объявление новой переменной, или часть лямбда-выражения в цикле For, или просто использование идентификатора. Также нужно выяснить, нужно ли приводить данный идентификатор к какому-либо типу данных.
* Если лексема равна l, нужно понять, нужно ли приводить ее к другому типу данных.
* Если лексемы равны ;, или, }, или {, то это означает, что при генерации следующего кода, его надо перенести на следующую строку и добавить табуляторы.

## 7.5 Контрольный пример

Контрольный пример С++ кода приведен в приложении Ж.

Результат генерации С++ кода приведен на рисунке 7.2.



Рис.7.2 Результат генерации С++ кода

# Глава 8. Тестирование транслятора

## 8.1 Тестирование проверки на допустимость символов

В языке GDV-2022 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 Тестирование проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  a => ‘/’;  return 0;  } | Ошибка: 111 : Недопустимый символ в исходном файле (-in)  Строка: 3 Символ: 7 |

## 8.2 Тестирование лексического анализатора

Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  a is char = '1;  return 0;  } | Ошибка: 600 : Нет закрывающей кавычки  Строка: 3 Символ: 0 |

## 8.3 Тестирование синтаксического анализатора

Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  a => 23  return 0;  } | 670: строка 4, Ошибка. Не хватает точки с запятой  670: строка 4, Ошибка. Не хватает точки с запятой  670: строка 4, Ошибка. Не хватает точки с запятой  Ошибка: 121 : Ошибка в синтаксисе! |

## 8.4 Тестирование семантического анализатора

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  b is num = a & 23;  return 0;  } | Ошибка: 609 : Использована необъявленная переменная  Строка: 3 Символ: 0 |

# Приложение А

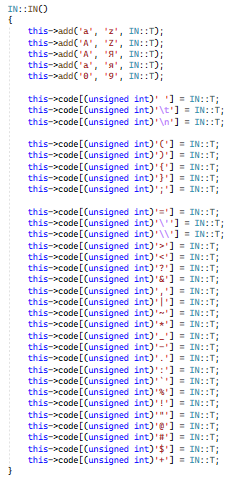


Рис. 1 Контроль входных символов

  
Рис. 2 Лексемы языка GDV-2022

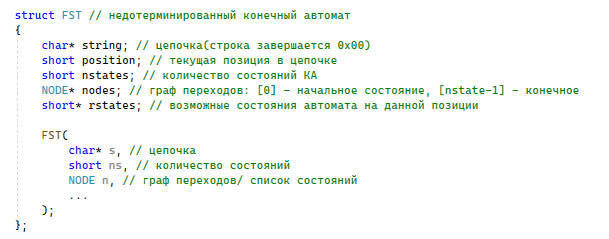
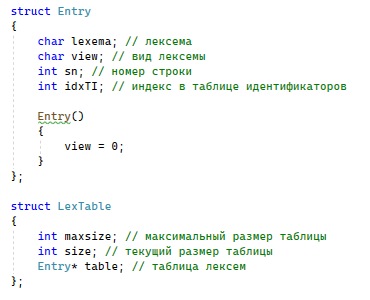
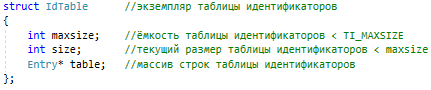


Рис.3 Реализация конечных автоматов





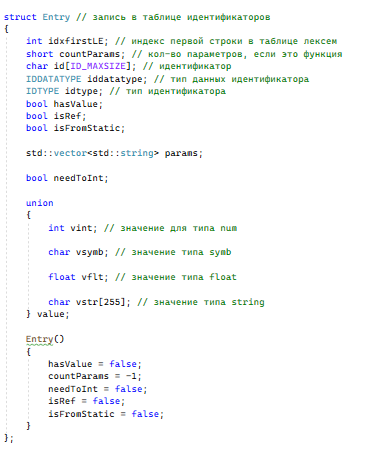


Рис. 4 Основные структуры данных лексического анализатора

# Приложение Б



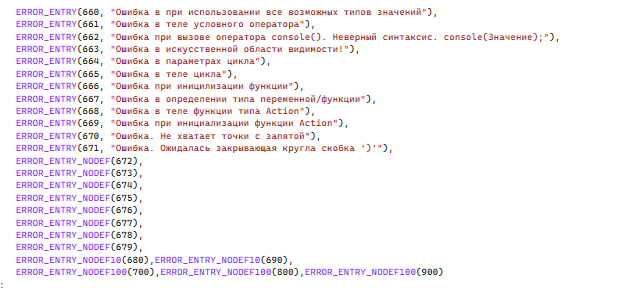
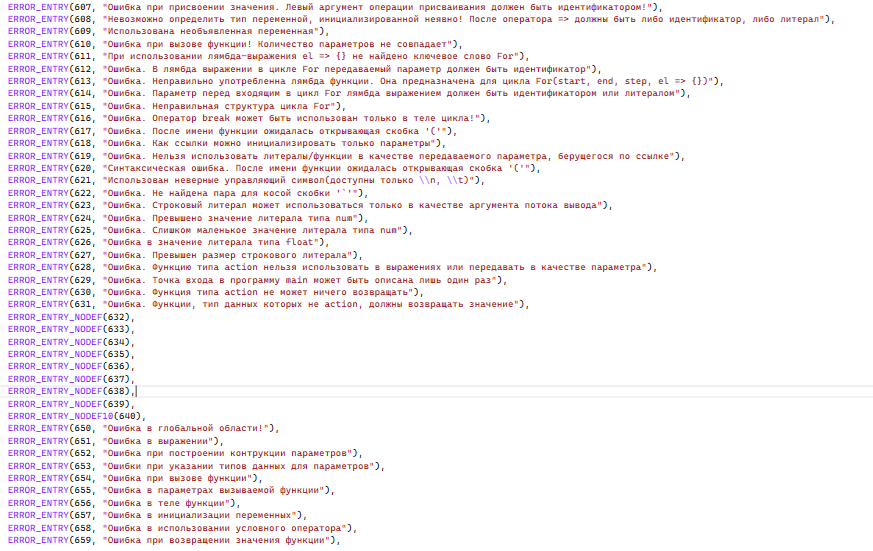
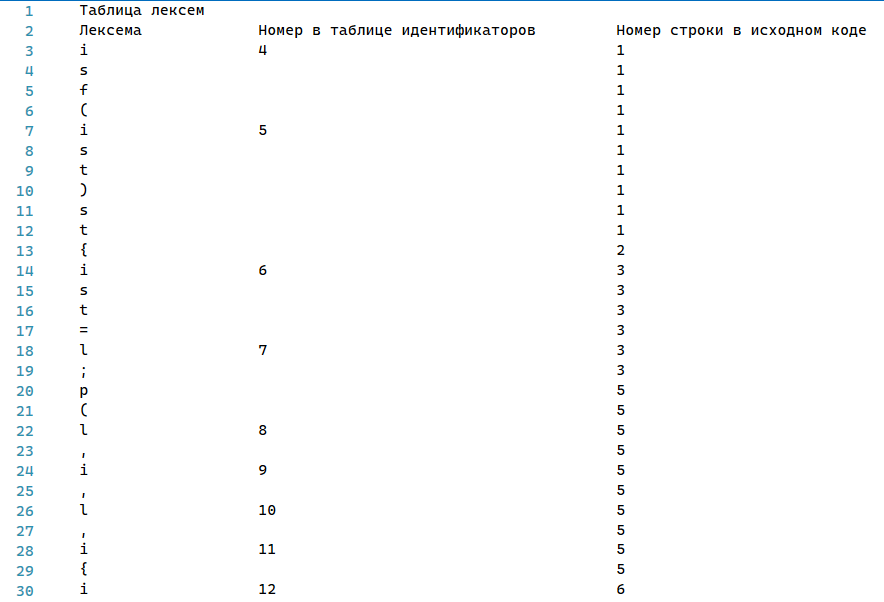
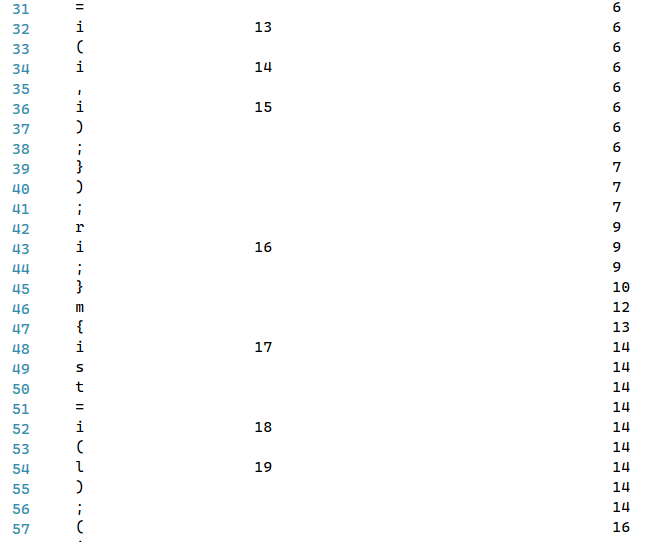


Рис.4 Таблица ошибок языка GDV-2022





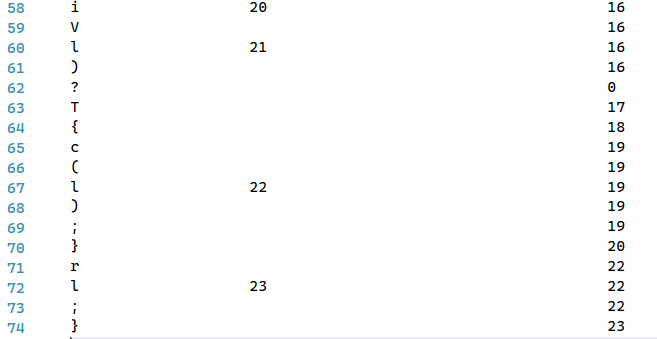
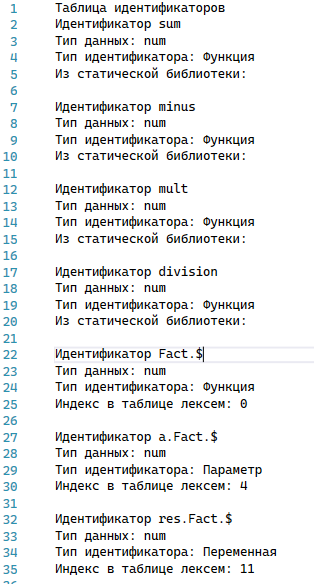
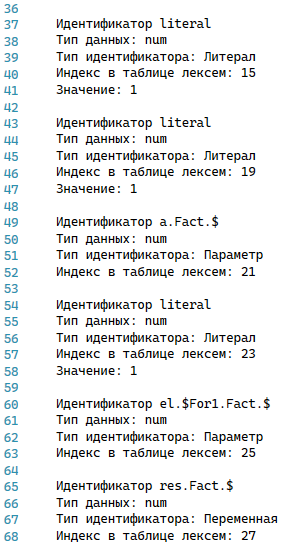
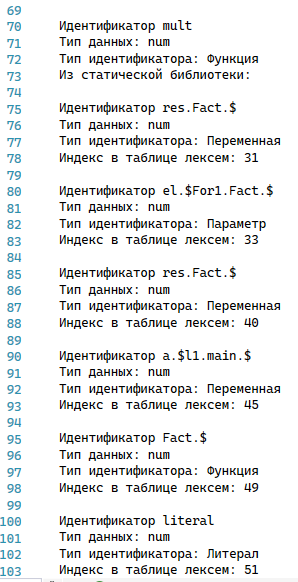


Рис.5 Таблица лексем языка GDV-2022







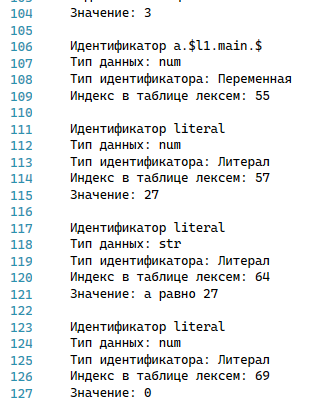


Рис.7 Таблица идентификаторов языка GDV-2022

# Приложение В

|  |
| --- |
| GRB::Greibach greibach(  NS(GLOBAL),  TS('$'),  22, // edit  Rule(NS(GLOBAL), GRB\_ERROR\_SERIES + 0, // глобальное пространство  14,  Rule::Chain(3, TS('i'), NS(INIT\_FUNC), NS(GLOBAL)),  Rule::Chain(2, TS('i'), NS(INIT\_FUNC)),  Rule::Chain(3, TS('i'), NS(INIT\_ACTION), NS(GLOBAL)),  Rule::Chain(2, TS('i'), NS(INIT\_ACTION)),  Rule::Chain(6, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON), NS(GLOBAL)),  Rule::Chain(5, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON)),  Rule::Chain(4, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON), NS(GLOBAL)),  Rule::Chain(3, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON)),  Rule::Chain(5, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON), NS(GLOBAL)),  Rule::Chain(4, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON)),  Rule::Chain(5, TS('m'), TS('{'), NS(BODY\_FUNC), TS('}'), NS(GLOBAL)),  Rule::Chain(4, TS('m'), TS('{'), NS(BODY\_FUNC), TS('}')),    Rule::Chain(4, TS('m'), TS('{'), NS(RETURN), TS('}')),  Rule::Chain(5, TS('m'), TS('{'), NS(RETURN), TS('}'), NS(GLOBAL))  ),  Rule(NS(INIT\_FUNC), GRB\_ERROR\_SERIES + 16, // ошибка при инициализации функции  2,  Rule::Chain(7, TS('s'), TS('f'), NS(IPARAMS), NS(DEF\_TYPE), TS('{'), NS(BODY\_FUNC), TS('}')),  Rule::Chain(7, TS('s'), TS('f'), NS(IPARAMS), NS(DEF\_TYPE), TS('{'), NS(RETURN), TS('}'))  ),  Rule(NS(DEF\_TYPE), GRB\_ERROR\_SERIES + 17, // ошибка при определении типа  1,  Rule::Chain(2, TS('s'), TS('t'))  ),    /////////////////////////    Rule(NS(BODY\_ACTION), GRB\_ERROR\_SERIES + 18, // ошибка в функции типа action  24,    Rule::Chain(4, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON), NS(BODY\_ACTION)),  Rule::Chain(3, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON)),    Rule::Chain(6, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), NS(SEMICOLON), NS(BODY\_ACTION)),  Rule::Chain(5, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), NS(SEMICOLON)),  Rule::Chain(5, TS('i'), TS('('), NS(RIGHTSCOPE), NS(SEMICOLON), NS(BODY\_ACTION)),  Rule::Chain(4, TS('i'), TS('('), NS(RIGHTSCOPE), NS(SEMICOLON)),  Rule::Chain(6, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON), NS(BODY\_ACTION)),  Rule::Chain(5, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON)),  Rule::Chain(5, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON), NS(BODY\_ACTION)),  Rule::Chain(4, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON)),  Rule::Chain(4, TS('{'), NS(BODY\_ACTION), TS('}'), NS(BODY\_ACTION)),  Rule::Chain(3, TS('{'), NS(BODY\_ACTION), TS('}')),  Rule::Chain(3, TS('{'), TS('}'), NS(BODY\_FUNC)),  Rule::Chain(2, TS('{'), TS('}')),  Rule::Chain(4, TS('c'), NS(CONSOLE), NS(SEMICOLON), NS(BODY\_ACTION)),  Rule::Chain(3, TS('c'), NS(CONSOLE), NS(SEMICOLON)),  Rule::Chain(3, TS(FOR), NS(FOR\_PARAM), NS(BODY\_ACTION)),  Rule::Chain(2, TS(FOR), NS(FOR\_PARAM)),  Rule::Chain(8, TS('('), NS(TYPES\_VALUES), TS('V'), NS(TYPES\_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY), NS(BODY\_ACTION)),  Rule::Chain(7, TS('('), NS(TYPES\_VALUES), TS('V'), NS(TYPES\_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY)),  Rule::Chain(4, TS('i'), NS(INIT\_FUNC), NS(SEMICOLON), NS(BODY\_ACTION)),  Rule::Chain(3, TS('i'), NS(INIT\_FUNC), NS(SEMICOLON)),    Rule::Chain(2, TS('r'), NS(SEMICOLON)),  Rule::Chain(3, TS('r'), NS(SEMICOLON), NS(BODY\_ACTION))  ),  Rule(NS(INIT\_ACTION), GRB\_ERROR\_SERIES + 19,  2,  Rule::Chain(8, TS('s'), TS('f'), NS(IPARAMS), TS('s'), TS('a'), TS('{'), NS(BODY\_ACTION), TS('}')),  Rule::Chain(7, TS('s'), TS('f'), NS(IPARAMS), TS('s'), TS('a'), TS('{'), TS('}'))  ),  Rule(NS(SEMICOLON), GRB\_ERROR\_SERIES + 20,  1,  Rule::Chain(1, TS(';'))  ),    Rule(NS(RIGHTSCOPE), GRB\_ERROR\_SERIES + 21,  1,  Rule::Chain(1, TS(')'))  ),  /////////////////////////  Rule(NS(EXPR), GRB\_ERROR\_SERIES + 1, // выражение  11,  Rule::Chain(1, TS('l')),  Rule::Chain(1, TS('i')),  Rule::Chain(4, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE)),  Rule::Chain(3, TS('i'), TS('('), NS(RIGHTSCOPE)),  Rule::Chain(3, TS('i'), TS('v'), NS(EXPR)),  Rule::Chain(3, TS('l'), TS('v'), NS(EXPR)),  Rule::Chain(6, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), TS('v'), NS(EXPR)),  Rule::Chain(5, TS('i'), TS('('), NS(RIGHTSCOPE), TS('v'), NS(EXPR)),  Rule::Chain(3, TS('('), NS(EXPR), NS(RIGHTSCOPE)),  Rule::Chain(5, TS('('), NS(EXPR), NS(RIGHTSCOPE), TS('v'), NS(EXPR)),  Rule::Chain(2, TS('~'), NS(EXPR))  ),  Rule(NS(IPARAMS), GRB\_ERROR\_SERIES + 2, // параметры  2,  Rule::Chain(2, TS('('), NS(RIGHTSCOPE)),  Rule::Chain(3, TS('('), NS(IPARAMS\_ARGS), NS(RIGHTSCOPE))  ),  Rule(NS(IPARAMS\_ARGS), GRB\_ERROR\_SERIES + 3, // инициализация параметров  2,  Rule::Chain(2, TS('i'), NS(DEF\_TYPE)),  Rule::Chain(4, TS('i'), NS(DEF\_TYPE), TS(','), NS(IPARAMS\_ARGS))  ),  Rule(NS(CALLFUNC), GRB\_ERROR\_SERIES + 4, // вызов функции  2,  Rule::Chain(3, TS('i'), TS('('), NS(RIGHTSCOPE)),  Rule::Chain(4, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE))  ),  Rule(NS(ARGS\_CFUNC), GRB\_ERROR\_SERIES + 5, // параметры вызываемой функции  8,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(4, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE)),  Rule::Chain(3, TS('i'), TS('('), NS(RIGHTSCOPE)),  Rule::Chain(3, TS('i'), TS(','), NS(ARGS\_CFUNC)),  Rule::Chain(3, TS('l'), TS(','), NS(ARGS\_CFUNC)),  Rule::Chain(6, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), TS(','), NS(ARGS\_CFUNC)),  Rule::Chain(5, TS('i'), TS('('), NS(RIGHTSCOPE), TS(','), NS(ARGS\_CFUNC))  ),  Rule(NS(BODY\_FUNC), GRB\_ERROR\_SERIES + 6, // тело функции  22,    Rule::Chain(6, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), NS(SEMICOLON), NS(BODY\_FUNC)),  Rule::Chain(6, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(5, TS('i'), TS('('), NS(RIGHTSCOPE), NS(SEMICOLON), NS(BODY\_FUNC)),  Rule::Chain(5, TS('i'), TS('('), NS(RIGHTSCOPE), NS(SEMICOLON), NS(RETURN)),    Rule::Chain(4, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON), NS(BODY\_FUNC)),  Rule::Chain(4, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(6, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON), NS(BODY\_FUNC)),  Rule::Chain(6, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(5, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON), NS(BODY\_FUNC)),  Rule::Chain(5, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(4, TS('{'), NS(VISAREA), TS('}'), NS(BODY\_FUNC)),  Rule::Chain(4, TS('{'), NS(VISAREA), TS('}'), NS(RETURN)),    Rule::Chain(3, TS('{'), TS('}'), NS(BODY\_FUNC)),  Rule::Chain(3, TS('{'), TS('}'), NS(RETURN)),  Rule::Chain(4, TS('c'), NS(CONSOLE), NS(SEMICOLON), NS(BODY\_FUNC)),  Rule::Chain(4, TS('c'), NS(CONSOLE), NS(SEMICOLON), NS(RETURN)),    Rule::Chain(3, TS(FOR), NS(FOR\_PARAM), NS(BODY\_FUNC)),  Rule::Chain(3, TS(FOR), NS(FOR\_PARAM), NS(RETURN)),    Rule::Chain(8, TS('('), NS(TYPES\_VALUES), TS('V'), NS(TYPES\_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY), NS(BODY\_FUNC)),  Rule::Chain(8, TS('('), NS(TYPES\_VALUES), TS('V'), NS(TYPES\_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY), NS(RETURN)),  Rule::Chain(4, TS('i'), NS(INIT\_FUNC), NS(SEMICOLON), NS(BODY\_FUNC)),  Rule::Chain(4, TS('i'), NS(INIT\_FUNC), NS(SEMICOLON), NS(RETURN))  ),    Rule(NS(INIT), GRB\_ERROR\_SERIES + 7, // инициализация переменных  1,  Rule::Chain(3, TS('i'), TS('s'), TS('t'))  ),    Rule(NS(IF), GRB\_ERROR\_SERIES + 8, // условие  36,  Rule::Chain(6, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(6, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), NS(SEMICOLON), NS(IF)),  Rule::Chain(5, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), NS(SEMICOLON)),  Rule::Chain(5, TS('i'), TS('('), NS(RIGHTSCOPE), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(5, TS('i'), TS('('), NS(RIGHTSCOPE), NS(SEMICOLON), NS(IF)),  Rule::Chain(4, TS('i'), TS('('), NS(RIGHTSCOPE), NS(SEMICOLON)),    Rule::Chain(4, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON), NS(IF)),  Rule::Chain(4, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(3, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON)),    Rule::Chain(6, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON), NS(IF)),  Rule::Chain(6, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(5, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON)),    Rule::Chain(5, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON), NS(IF)),  Rule::Chain(5, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(4, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON)),    Rule::Chain(4, TS('{'), NS(IF), TS('}'), NS(IF)),  Rule::Chain(4, TS('{'), NS(IF), TS('}'), NS(RETURN)),  Rule::Chain(3, TS('{'), NS(IF), TS('}')),  Rule::Chain(2, TS('{'), TS('}')),  Rule::Chain(3, TS('{'), TS('}'), NS(IF)),  Rule::Chain(3, TS('{'), TS('}'), NS(RETURN)),    Rule::Chain(4, TS('i'), NS(INIT\_FUNC), NS(SEMICOLON), NS(IF)),  Rule::Chain(4, TS('i'), NS(INIT\_FUNC), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(3, TS('i'), NS(INIT\_FUNC), NS(SEMICOLON)),    Rule::Chain(4, TS('c'), NS(CONSOLE), NS(SEMICOLON), NS(IF)),  Rule::Chain(4, TS('c'), NS(CONSOLE), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(3, TS('c'), NS(CONSOLE), NS(SEMICOLON)),    Rule::Chain(8, TS('('), NS(TYPES\_VALUES), TS('V'), NS(TYPES\_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY), NS(IF)),  Rule::Chain(8, TS('('), NS(TYPES\_VALUES), TS('V'), NS(TYPES\_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY), NS(RETURN)),  Rule::Chain(7, TS('('), NS(TYPES\_VALUES), TS('V'), NS(TYPES\_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY)),    Rule::Chain(3, TS('r'), NS(EXPR), NS(SEMICOLON)),  Rule::Chain(2, TS('r'), NS(SEMICOLON)),    Rule::Chain(2, TS('b'), NS(SEMICOLON)),  Rule::Chain(3, TS('b'), NS(SEMICOLON), NS(IF)),    Rule::Chain(2, TS(SKIP), NS(SEMICOLON)),  Rule::Chain(3, TS(SKIP), NS(SEMICOLON), NS(IF))  ),    Rule(NS(RETURN), GRB\_ERROR\_SERIES + 9, // return  2,  Rule::Chain(3, TS('r'), NS(EXPR), NS(SEMICOLON)),  Rule::Chain(2, TS('r'), NS(SEMICOLON))  ),    Rule(NS(TYPES\_VALUES), GRB\_ERROR\_SERIES + 10, // типы значений  4,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('i'), TS('('), NS(RIGHTSCOPE)),  Rule::Chain(4, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE))  ),    Rule(NS(IFBODY), GRB\_ERROR\_SERIES + 11, // тело if  8,  Rule::Chain(4, TS('T'), TS('{'), NS(IF), TS('}')),  Rule::Chain(8, TS('T'), TS('{'), NS(IF), TS('}'), TS('L'), TS('{'), NS(IF), TS('}')),  Rule::Chain(4, TS('L'), TS('{'), NS(IF), TS('}')),    Rule::Chain(3, TS('T'), TS('{'), TS('}')),  Rule::Chain(3, TS('L'), TS('{'), TS('}')),    Rule::Chain(7, TS('T'), TS('{'), TS('}'), TS('L'), TS('{'), NS(IF), TS('}')),  Rule::Chain(7, TS('T'), TS('{'), NS(IF), TS('}'), TS('L'), TS('{'), TS('}')),  Rule::Chain(6, TS('T'), TS('{'), TS('}'), TS('L'), TS('{'), TS('}'))  ),    Rule(NS(CONSOLE), GRB\_ERROR\_SERIES + 12, // вызов потока вывода  2,  Rule::Chain(3, TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE)),  Rule::Chain(2, TS('('), NS(RIGHTSCOPE))  ),    Rule(NS(VISAREA), GRB\_ERROR\_SERIES + 13, // тело функции  36,  Rule::Chain(6, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), NS(SEMICOLON), NS(VISAREA)),  Rule::Chain(6, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(5, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), NS(SEMICOLON)),  Rule::Chain(5, TS('i'), TS('('), NS(RIGHTSCOPE), NS(SEMICOLON), NS(VISAREA)),  Rule::Chain(5, TS('i'), TS('('), NS(RIGHTSCOPE), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(4, TS('i'), TS('('), NS(RIGHTSCOPE), NS(SEMICOLON)),  Rule::Chain(4, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON), NS(VISAREA)),  Rule::Chain(4, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(3, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON)),    Rule::Chain(6, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON), NS(VISAREA)),  Rule::Chain(6, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(5, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON)),    Rule::Chain(5, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON), NS(VISAREA)),  Rule::Chain(5, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(4, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON)),    Rule::Chain(4, TS('{'), NS(VISAREA), TS('}'), NS(VISAREA)),  Rule::Chain(4, TS('{'), NS(VISAREA), TS('}'), NS(RETURN)),  Rule::Chain(3, TS('{'), NS(VISAREA), TS('}')),    Rule::Chain(2, TS('{'), TS('}')),  Rule::Chain(3, TS('{'), TS('}'), NS(VISAREA)),  Rule::Chain(3, TS('{'), TS('}'), NS(RETURN)),    Rule::Chain(4, TS('i'), NS(INIT\_FUNC), NS(SEMICOLON), NS(VISAREA)),  Rule::Chain(4, TS('i'), NS(INIT\_FUNC), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(3, TS('i'), NS(INIT\_FUNC), NS(SEMICOLON)),    Rule::Chain(4, TS('c'), NS(CONSOLE), NS(SEMICOLON), NS(VISAREA)),  Rule::Chain(4, TS('c'), NS(CONSOLE), NS(SEMICOLON), NS(RETURN)),  Rule::Chain(3, TS('c'), NS(CONSOLE), NS(SEMICOLON)),    Rule::Chain(8, TS('('), NS(TYPES\_VALUES), TS('V'), NS(TYPES\_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY), NS(VISAREA)),  Rule::Chain(8, TS('('), NS(TYPES\_VALUES), TS('V'), NS(TYPES\_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY), NS(RETURN)),  Rule::Chain(7, TS('('), NS(TYPES\_VALUES), TS('V'), NS(TYPES\_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY)),  Rule::Chain(2, TS('b'), NS(SEMICOLON)),  Rule::Chain(3, TS('b'), NS(SEMICOLON), NS(VISAREA)),    Rule::Chain(3, TS('r'), NS(EXPR), NS(SEMICOLON)),  Rule::Chain(2, TS('r'), NS(SEMICOLON)),    Rule::Chain(2, TS(SKIP), NS(SEMICOLON)),  Rule::Chain(3, TS(SKIP), NS(SEMICOLON), NS(VISAREA))  ),    Rule(NS(FOR\_PARAM), GRB\_ERROR\_SERIES + 14, // Ошибка в параметрах цикла  1,  Rule::Chain(13, TS('('), NS(TYPES\_VALUES), TS(','), NS(TYPES\_VALUES), TS(','), NS(TYPES\_VALUES), TS(','), TS('i'), TS('{'), NS(FOR\_BODY), TS('}'), NS(RIGHTSCOPE), NS(SEMICOLON))  ),    Rule(NS(FOR\_BODY), GRB\_ERROR\_SERIES + 15, // Ошибка в теле цикла For  26,  Rule::Chain(6, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), NS(SEMICOLON), NS(FOR\_BODY)),  Rule::Chain(5, TS('i'), TS('('), NS(ARGS\_CFUNC), NS(RIGHTSCOPE), NS(SEMICOLON)),  Rule::Chain(5, TS('i'), TS('('), NS(RIGHTSCOPE), NS(SEMICOLON), NS(FOR\_BODY)),  Rule::Chain(4, TS('i'), TS('('), NS(RIGHTSCOPE), NS(SEMICOLON)),  Rule::Chain(4, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON), NS(FOR\_BODY)),  Rule::Chain(3, TS('i'), NS(DEF\_TYPE), NS(SEMICOLON)),    Rule::Chain(6, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON), NS(FOR\_BODY)),  Rule::Chain(5, TS('i'), NS(DEF\_TYPE), TS('='), NS(EXPR), NS(SEMICOLON)),    Rule::Chain(5, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON), NS(FOR\_BODY)),  Rule::Chain(4, TS('i'), TS('='), NS(EXPR), NS(SEMICOLON)),    Rule::Chain(4, TS('{'), NS(FOR\_BODY), TS('}'), NS(FOR\_BODY)),  Rule::Chain(3, TS('{'), NS(FOR\_BODY), TS('}')),  Rule::Chain(2, TS('{'), TS('}')),  Rule::Chain(3, TS('{'), TS('}'), NS(FOR\_BODY)),    Rule::Chain(4, TS('i'), NS(INIT\_FUNC), NS(SEMICOLON), NS(FOR\_BODY)),  Rule::Chain(3, TS('i'), NS(INIT\_FUNC), NS(SEMICOLON)),    Rule::Chain(4, TS('c'), NS(CONSOLE), NS(SEMICOLON), NS(FOR\_BODY)),  Rule::Chain(3, TS('c'), NS(CONSOLE), NS(SEMICOLON)),  Rule::Chain(3, TS(FOR), NS(FOR\_PARAM), NS(FOR\_BODY)),  Rule::Chain(2, TS(FOR), NS(FOR\_PARAM)),  Rule::Chain(8, TS('('), NS(TYPES\_VALUES), TS('V'), NS(TYPES\_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY), NS(FOR\_BODY)),  Rule::Chain(7, TS('('), NS(TYPES\_VALUES), TS('V'), NS(TYPES\_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY)),  Rule::Chain(2, TS('b'), NS(SEMICOLON)),  Rule::Chain(3, TS('b'), NS(SEMICOLON), NS(FOR\_BODY)),  Rule::Chain(2, TS(SKIP), NS(SEMICOLON),  Rule::Chain(3, TS(SKIP), NS(SEMICOLON), NS(FOR\_BODY))  )  )  ); |

Рис. 8 Структура Грамматики Грейбах

# Приложение Г

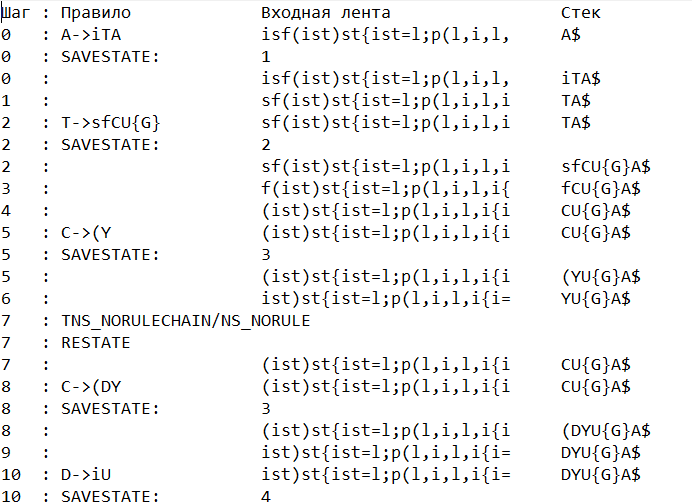


Рис. 125 Разбор исходного кода синтаксическим анализатором

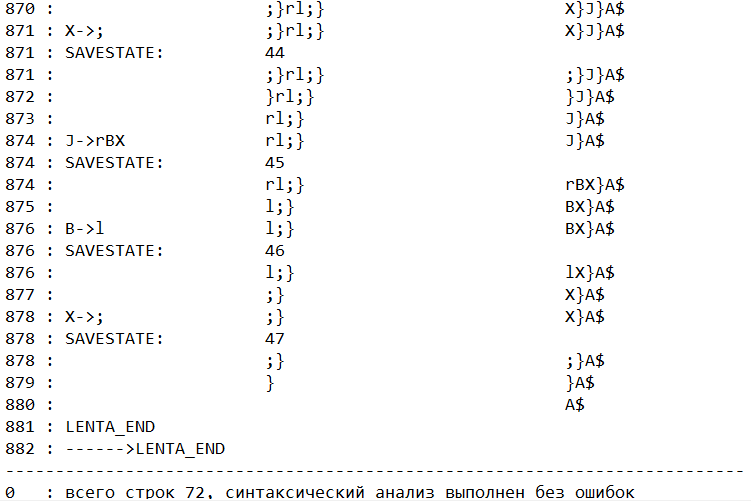


Рис. 125 (продолжение) Разбор исходного кода синтаксическим анализатором(конец)

# Приложение Ж

Таблица 1. Результат генерации кода

|  |
| --- |
| #include <iostream>  #include <windows.h>  #pragma comment(lib, "..\\GDV-2022\\Debug\\LIB.lib")  #include "..\\LIB\\framework.h"  int Fact\_$(int a\_Fact\_$)  {  int res\_Fact\_$ = 1;  For((int)1, (int)a\_Fact\_$, 1, [&](int el\_$For1\_Fact\_$)  {  res\_Fact\_$ = mult((int)res\_Fact\_$, (int)el\_$For1\_Fact\_$);      return false;  });  return res\_Fact\_$;    }  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  int a\_$l1\_main\_$ = Fact\_$((int)3);  if (a\_$l1\_main\_$ == 27)  {  std::cout << "a равно 27";    }  return 0;    } |

# Заключение

В ходе выполнения курсовой работы был разработан компилятор и генератор кода для языка программирования GDV-2022. Таким образом, были выполнены основные задачи данной курсовой работы:

* Сформулирована спецификация языка GDV-2022;
* 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, позволяющая расширять набор синтаксических конструкций языка только за счёт внесения изменений в разработанную грамматику;
* Разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
* Разработан транслятор кода на C++;
* Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка GDV-2022 включает:

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

Проделанная работа позволила получить необходимое представление о структурах и процессах, использующихся при построении компиляторов.

# Литература

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

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

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

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