МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора LOK-2023»

Выполнил студент Кивлинас Олег Леонидович

(Ф.И.О.)

Руководитель проекта ст.преп. Наркевич А.С.

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

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

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

Консультанты ст.преп. Наркевич А.С.

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

Курсовой проект защищен с оценкой

**Содержание**

[Введение 4](#_Toc122442276)

[1 Спецификация языка программирования 5](#_Toc122442277)

[1.1 Характеристика языка программирования 5](#_Toc122442278)

[1.2 Определение алфавита языка программирования 5](#_Toc122442279)

[1.3 Применяемые сепараторы 5](#_Toc122442280)

[1.4 Применяемые кодировки 6](#_Toc122442281)

[1.5 Типы данных 7](#_Toc122442282)

[1.6 Преобразование типов данных 7](#_Toc122442283)

[1.7 Идентификаторы 8](#_Toc122442284)

[1.8 Литералы 8](#_Toc122442285)

[1.9 Объявление данных 9](#_Toc122442286)

[1.10 Инициализация данных 10](#_Toc122442287)

[1.11 Инструкции языка 10](#_Toc122442288)

[1.12 Операции языка 10](#_Toc122442289)

[1.13 Выражения и их вычисление 11](#_Toc122442290)

[1.14 Конструкции языка 11](#_Toc122442291)

[1.15 Области видимости идентификаторов. 12](#_Toc122442292)

[1.16 Семантические проверки 12](#_Toc122442293)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc122442294)

[1.18 Стандартная библиотека и её состав 13](#_Toc122442295)

[1.19 Ввод и вывод данных 14](#_Toc122442296)

[1.20 Точка входа 14](#_Toc122442297)

[1.21 Препроцессор 14](#_Toc122442298)

[1.22 Соглашения о вызовах 14](#_Toc122442299)

[1.23 Объектный код 15](#_Toc122442300)

[1.24 Классификация сообщений транслятора 15](#_Toc122442301)

[1.25 Контрольный пример 15](#_Toc122442302)

[2 Структура транслятора 16](#_Toc122442303)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 16](#_Toc122442304)

[2.2 Перечень параметров транслятора 16](#_Toc122442305)

[2.3 Протоколы, формируемые транслятором 16](#_Toc122442306)

[3 Разработка лексического анализатора 18](#_Toc122442307)

[3.1 Структура лексического анализатора 18](#_Toc122442308)

[3.2 Входные и выходные данные лексического анализатора 18](#_Toc122442309)

[3.3 Параметры лексического анализатора 18](#_Toc122442310)

[3.4 Алгоритм лексического анализатора 19](#_Toc122442311)

[3.5 Контроль входных символов 19](#_Toc122442312)

[3.6 Удаление избыточных символов 19](#_Toc122442313)

[3.7 Перечень ключевых слов 19](#_Toc122442314)

[3.8 Основные структуры данных 21](#_Toc122442315)

[3.9 Структура и перечень сообщений лексического анализатора 22](#_Toc122442316)

[3.10 Принцип обработки ошибок 22](#_Toc122442317)

[3.11 Контрольный пример 22](#_Toc122442317)

[4 Разработка синтаксического анализатора 23](#_Toc122442318)

[4.1 Структура синтаксического анализатора 23](#_Toc122442319)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 23](#_Toc122442320)

[4.3 Построение конечного магазинного автомата 28](#_Toc122442321)

[4.4 Основные структуры данных 29](#_Toc122442322)

[4.5 Описание алгоритма синтаксического разбора 29](#_Toc122442323)

[4.6 Параметры синтаксического анализатора 30](#_Toc122442324)

[4.7 Структура и перечень сообщений синтаксического анализатора 30](#_Toc122442325)

[4.8 Принцип обработки ошибок 31](#_Toc122442326)

[4.9 Контрольный пример 31](#_Toc122442327)

[5 Разработка семантического анализатора 32](#_Toc122442328)

[5.1 Структура семантического анализатора 32](#_Toc122442329)

[5.2 Функции семантического анализатора 32](#_Toc122442330)

[5.3 Структура и перечень сообщений семантического анализатора 32](#_Toc122442331)

[5.4 Принцип обработки ошибок 33](#_Toc122442332)

[5.5 Контрольный пример 33](#_Toc122442332)

[6 Вычисление выражений 35](#_Toc122442333)

[6.1 Выражения, допускаемые языком 35](#_Toc122442334)

[6.2 Польская запись и принцип ее построения 35](#_Toc122442335)

[6.3 Программная реализация обработки выражений 35](#_Toc122442336)

[6.4 Контрольный пример 35](#_Toc122442337)

[7 Генерация кода 36](#_Toc122442338)

[7.1 Структура генератора кода 36](#_Toc122442339)

[7.2 Представление типов данных в оперативной памяти 36](#_Toc122442340)

[7.3 Статическая библиотека 36](#_Toc122442341)

[7.4 Особенности алгоритма генерации кода 37](#_Toc122442342)

[7.5 Параметры, управляющие генерацией кода 37](#_Toc122442342)

[7.6 Контрольный пример 37](#_Toc122442343)

[8 Тестирование транслятора 39](#_Toc122442344)

[8.1 Общие положения **Ошибка! Закладка не определена.**](#_Toc122442345)

[8.2 Результаты тестирования 39](#_Toc122442346)

[Заключение 41](#_Toc122442349)

[Литература 44](#_Toc122442350)

[Приложение А 45](#_Toc122442351)

[Приложение Б 46](#_Toc122442352)

[Приложение В 48](#_Toc122442353)

[Приложение Г 62](#_Toc122442354)

[Приложение Д 64](#_Toc122442354)

[Приложение Е 66](#_Toc122442355)

[Приложение Ж 72](#_Toc122442356)

# 

# Введение

Целью курсового проектирования является разработка транслятора для языка программирования LOK-2023, уникального языка программирования, который требует эффективной интерпретации и перевода в низкоуровневый язык ассемблера. Основным языком программирования, выбранным для реализации транслятора, является С++, известный своей эффективностью и высокой производительностью.

Транслятор – это комплекс отдельных программ, позволяющих преобразовывать исходный код на одном языке программирования в исходный код на другом языке программирования.

Транслятор состоит из следующих частей:

– лексический анализатор;

– семантический анализатор;

– синтаксический анализатор;

– генератор исходного кода на языке JavaScript.

Исходя из цели курсового проекта, были поставлены следующие задачи:

– разработка дизайна языка;

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка и преобразование выражений;

– генерация кода на язык JavaScript;

– тестирование транслятора.

В данном курсовом проекте будет определена спецификация языка программирования, представлена структура транслятора, показана разработка лексического анализатора, порождающего таблицы лексем и идентификаторов, рассказано о синтаксическом анализаторе, который выполняет синтаксический разбор текста с распечаткой протокола разбора и дерева разбора на основе таблицы лексем, описан семантический анализатор, показана его работа, решены вопросы преобразования выражений, допускаемых языком и приведена часть протокола для контрольного примера, отображающая результаты преобразования выражений в польский формат, представлена генерация кода, где из промежуточного представления порождается код на целевом языке, а так же описано тестирование кода.

# 1 Спецификация языка программирования

## 1.1 Характеристика языка программирования

Язык программирования LOK-2023 – это процедурный, универсальный, строго типизированный, компилируемый язык. Процедурный - это означает, что язык LOK-2023 организован вокруг процедур и функций, которые могут быть вызваны и выполнены в определенном порядке. Универсальный - это означает, что он может быть предназначен для разработки различных типов программ, от простых сценариев до сложных приложений. Строго типизированный - это означает, что каждая переменная и выражение в языке LOK-2023 имеют определенный тип данных, и компилятор строго проверяет соответствие типов. Компилируемый - это означает, что исходный код программы на языке LOK-2023 должен быть скомпилирован в машинный код перед выполнением программы.

## 1.2 Определение алфавита языка программирования

Алфавит языка программирования – это множество символов, которые можно использовать для написания программ на данном языке программирования.

Алфавит языка LOK-2023 основывается на таблице ASCII, которая представлена в таблице 1.1.

Таблица 1.1 – Таблица кодировок ASCII



В алфавите языка LOK-2023 используются следующие символы: [a … z], [A … Z], [0 … 9], [А … Я], [а … я], спецсимволы: () , ; : ` ' & | ~ < > = ! , а также символы перевода строки, табуляции и пробела.

## **1.3 Применяемые сепараторы**

Сепараторы – это символы, которые применяются для разбиения элементов в исходном коде и обозначения конца выражений или инструкций. Они используются для явного указания, где одна часть кода заканчивается, а другая начинается, обеспечивая правильный синтаксис и структуру программы.

Применяемые сепараторы в языке программирования LOK-2023 описаны в таблице 1.2.

Таблица 1.2 – Сепараторы языка LOK-2023

|  |  |
| --- | --- |
| Сепараторы | Назначение и область применения сепаратора |
| { … } | Для блока функций или условной конструкции/цикла |
| ( … ) | Для фактических или формальных параметров функции, а также для установки приоритета операций |
| ‘пробел’, ‘табуляция’ | Для разделения цепочек/лексем языка (не допускается использование в названиях идентификаторов и ключевых слов) |
| , | Для разделения параметров функции, а так же цикла For |
| & | ~ | Побитовые операции |
| < > == != | Операторы сравнения |
| \n | Символ перехода на новую строку |

Сепараторы играют важную роль в обеспечении правильного синтаксиса и структуры программы.

## 1.4 Применяемые кодировки

Для написания исходного кода на языке программирования LOK-2023 используется кодировка Windows-1251, представленная на рисунке 1.1.



Рисунок 1.1 – Кодировка Windows-1251

Windows-1251 представляет собой набор символов и кодировок, являющаяся стандартной 8-битной кодировкой для русских версий [Microsoft Windows](https://ru.wikipedia.org/wiki/Windows" \o "Windows).

## 1.5 Типы данных

Язык программирования LOK-2023 поддерживает и позволяет использовать три типа данных для переменных и функций: целочисленный (num), символьный (symb) и числа с плавающей точкой (float). Кроме того, в языке LOK‑2023 есть особый тип данных для функции — action. Описание этих типов данных приведено в таблице 1.4.

Таблица 1.3 – Типы данных языка LOK-2023

|  |  |
| --- | --- |
| Тип данных | Описание |
| num | Фундаментальный тип данных. Предусмотрен для объявления целочисленных данных. В памяти занимает 4 байта. Значения по умолчанию нет. Максимально допустимое значение: 231-1. Минимально допустимым является -231-1 |
| symb | Фундаментальный тип данных. Является целочисленным типом данных, который предусмотрен для объявления символьных данных. В памяти занимает 1 байт. Значения по умолчанию нет. Максимально допустимое значение: 127. Минимально допустимое значение: -128 |
| float | Фундаментальный тип данных. Предусмотрен для объявления чисел с плавающей точной. В памяти занимает 4 байта. Значения по умолчанию нет. Максимально допустимое значение: 3.402823466e+38F. Минимально допустимое значение: 1.175494351e-38F |
| action | Особый тип данных. Предусмотрен для указания того, что данная функция является процедурой |

Ключевое слово ref используется только для параметров.

## 1.6 Преобразование типов данных

Преобразование типов данных в языке программирования LOK-2023 происходит неявно. Преобразования предусмотрены между целочисленными (num) и типом данных, для чисел с плавающей точкой (float).

Преобразования применяются в следующих случаях:

* при вызове функций, аргументы, переданные в нее, будут приведены к типу параметров данной функции;
* 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.

## 1.7 Идентификаторы

Идентификатор – это последовательность символов, которая служит в качестве имени для идентификации сущностей в языке программирования.

В языке LOK-2023 идентификаторы используются для именования переменных, функций и процедур, а также их параметров.

При создании идентификатора необходимо учитывать, что максимальная длина идентификатора составляет 50 символов. Если идентификатор превышает эту длину, он будет усечен до 50 символов. Идентификаторы должны быть уникальными в пределах своей области видимости и не должны совпадать с ключевыми словами или другими идентификаторами, созданными в той же самой области видимости. Для создания идентификаторов необходимо использовать буквы латинского алфавита как верхнего, так и нижнего регистра, а также цифры и символ нижнего подчеркивания. Формальное описание идентификатора представлено в таблице 1.4.

Таблица 1.4 – Формальное описание идентификатора (БНФ)

|  |
| --- |
| <идентификатор> ::= <буква> { <буква> | <цифра> | '\_' }  <буква> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z  <цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Примеры правильных идентификаторов: temperature, totalCount, x, getFullName.

Примеры неправильных идентификаторов: symb, foo, @name, ans++.

## 1.8 Литералы

Литералы представляют собой постоянные значения, назначаемые неизменяемым переменным. Они представляют собой фиксированные значения, которые не могут быть изменены. Литералы хранятся в памяти, но не имеют ссылок, как у переменных.

В языке LOK-2023 предусмотрены 4 вида литералов: целочисленные, строковые, символьные и числа с плавающей точкой. Краткое описание литералов приведено в таблице 1.5.

Таблица 1.5 – Литералы языка LOK-2023

|  |  |
| --- | --- |
| Литерал | Описание |
| Целочисленный | Может быть представлен в двух форматах: в десятичном виде, где используются цифры от 0 до 9 в виде последовательности, а также в шестнадцатеричном виде, где используются цифры от 0 до 9 и буквы от A до F, которым предшествует символ ‘x’ |
| Символьный | Символьный литерал представляет собой отдельный символ, заключённый в одинарные кавычки (‘’) |
| Числа с плавающей точкой | Представлен в виде последовательности цифр от 0 до 9, обязательно присутствия точки в записи литерала |

Продолжение таблицы 1.5

|  |  |
| --- | --- |
| Литерал | Описание |
| Строковый | Символы, заключенные в двойные кавычки (“”), только rvalue. Максимальное число символов в литерале – 255. Использование двойных кавычек внутри строкового литерала не допускается. |

Строковый литерал используется только внутри оператора вывода console.

## 1.9 Объявление данных

При объявлении переменных в языке LOK-2023 необходимо учитывать следующие правила:

* необходимо объявлять переменную до её использования;
* существует два типа объявления: с явной и с неявной типизацией;
* при явной необходимо сразу указывать тип данных с помощью ключевого слова is, после которого и идёт определение типа данных;
* при неявной типизации язык сам вычислит тип данных переменной, но для этого нужно использовать оператор '=>', после которого должен идти либо идентификатор, либо литерал, иначе будет ошибка.

Пример объявления переменной с явной типизацией: n is num, s is symb, f is float.

Пример объявления переменной с неявной типизацией: n => 2, s => ‘s’, f => 5.2.

Объявление функции схоже с объявлением обычной переменной:

* используется оператор is после идентификатора функции;
* далее идёт ключевое слово foo;
* после объявления параметров функции опять используется оператор is и объявляется тип функции.

Пример объявления функции целочисленного типа представлен в листинге 1.1.

|  |
| --- |
| MyFunc is foo(a is num) is num  {  return sum(a, 1);  } |

Листинг 1.1 Объявление функции целочисленного типа

Также язык программирования LOK-2023 позволяет объявлять шаблонные функции. Шаблонная функция в языке LOK-2023 — функция, объявленная не в глобальной области видимости, а в другой функции или в main. Их объявление такое же, как и объявление обычной функции, только типа данных action отсутствует, и после закрывающей фигурной скобки будет идти символ точки с запятой.

## 1.10 Инициализация данных

После объявления переменной можно присвоить ей значение с помощью оператора присваивания (=). Этот оператор является бинарным и требует два операнда. Слева от оператора должна находиться переменная, а справа — выражение, которое вычисляется и присваивает значение переменной.

## 1.11 Инструкции языка

Инструкции языка программирования LOK-2023 представлены в таблице 1.6

Таблица 1.6 – Инструкции языка LOK-2023

|  |  |  |
| --- | --- | --- |
| Инструкция | Запись на языке SMI-2023 | Пример |
| Объявление переменной с явной типизацией | <идентификатор> is <тип данных>; | a is num; |
| Объявление переменной с неявной типизацией | <идентификатор> => <выражение>; | c => ‘K’; |
| Присваивание | <идентификатор> = <выражение>; | val = 27; |
| Вывод данных | console(<список параметров>); | console(var); |
| Перевод строки | console(); | console(); |
| Вызов подпрограммы | <идентификатор функции> (<список параметров>); | mult(a, b); |
| Разветвление | (<условное выражение>) ? {<тело блока истины>} : {<тело блока лжи>} | (a == 5) ? { console(1); } : { console(2);} |
| Цикличность | For(<идентификатор|литерал|функция>,<идентификатор|литерал|функция>, <идентификатор|литерал|функция>, <идентификатор> => {<тело цикла For>}); | For(1, a, 1, i => { console(i)}; |

Тело цикла For не должно быть пустым.

## 1.12 Операции языка

В языке LOK-2023 присутствуют операции, описанные в таблице 1.7.

Операции логического умножения и логического сложения в языке LOK-2023 являются бинарными, то есть требуют два операнда. Операция логического отрицания является унарной, то есть требует только один операнд. Кроме того, в языке LOK-2023 предусмотрены операции сравнения значений, которые также представлены в таблице 1.7.

Таблица 1.7 – Операции языка LOK-2023

|  |  |  |
| --- | --- | --- |
| Тип операции | Операторы | Приоритет |
| Побитовые | | (логическое сложение)  & (логическое умножение)  ~ (логическое не) | 1  2  3 |
| Сравнения | > (больше)  < (меньше)  == (равенство по значению)  != (значения не равны)  is (проверить тип переменной) | -1  -1  -1  -1  -1 |
| Операция запятая | , | 1 |

Операции, заключенные в (..) (круглые скобки) имеют наивысший приоритет, равный 4.

## 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 выражениях языка LOK-2023 можно использовать комбинации всех типов данных, за исключением процедуры action и строковых литералов. В выражениях также допускаются все виды операций языка LOK-2023. Кроме того, язык предоставляет специальные функции из статической библиотеки, такие как sum(), minus(), mult() и division(), которые позволяют выполнять арифметические операции.

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

## 1.14 Конструкции языка

Программные конструкции языка программирование LOK-2023 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>;  } |

Продолжение таблицы 1.8

|  |  |
| --- | --- |
| Конструкции | Представление в языке |
| Процедура | <идентификатор> is foo(<идентификатор> is <тип данных>, …) is action  {  …  } |
| Шаблонная функция | <идентификатор> is foo(<идентификатор> is <тип данных>, …) is <тип данных>  {  …  return <выражение>;  }; |

В конце функции, всегда должен идти return. А конце процедуры – наоборот, не должен.

## 1.15 Область видимости идентификаторов

Область видимости в языке LOK-2023 организована следующим образом: все идентификаторы должны быть доступны из текущей области видимости или из вложенных областей видимости. Для этого не требуется использование дополнительных указателей. Для создания искусственной области видимости в языке LOK-2023 используются фигурные скобки { ... }.

## 1.16 Семантические проверки

В языке программирования LOK-2023 выполняются следующие семантические проверки:

* единственность точки входа;
* переопределение идентификаторов;
* использование идентификаторов без их объявления;
* проверка соответствия типа функции и возвращаемого параметра;
* правильность передаваемых в функцию параметров: количество, типы;
* правильность выражений;
* превышение размера плавающих, строковых и целочисленных литералов;

## 1.17 Распределение оперативной памяти на этапе выполнения

Все переменные размещаются в стеке.

## 1.18 Стандартная библиотека и её состав

Язык LOK-2023 предлагает стандартную библиотеку, которая автоматически включает некоторые функции при трансляции исходного кода в язык JavaScript. Некоторые функции, такие как round(), pow() и abs(), можно подключить с помощью ключевого слова @import. Содержимое библиотеки и описание функций представлено в таблице 1.9.

Таблица 1.9 – Стандартная библиотека языка LOK-2023

|  |  |
| --- | --- |
| 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 |

Продолжение таблицы 1.9

|  |  |
| --- | --- |
| 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 |
| 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 равны num |
| round is foo(number is num) is float; | Математическая функция. Возвращает округленное число, переданное в параметр. Подключается с помощью ключевого слова @import. Типы данных параметра и возвращаемого значения совпадают и равны float |

Стандартная библиотека в языке LOK-2023 предлагает значительное преимущество, заключающееся в возможности повторного использования кода.

## 1.19 Ввод и вывод данных

В языке LOK-2023 предусмотрен вывод данных, который осуществляется с помощью оператора console(). В качестве аргумента которого, 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

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке LOK-2023 отсутствует.

## 1.22 Соглашения о вызовах

В языке LOK-2023 используется соглашение вызова \_cdecl, что означает, что все параметры передаются в стек справа налево. При таком соглашении память, выделенная для параметров, освобождается автоматически функцией. Это означает, что разработчику не нужно самостоятельно освобождать память, выделенную для передаваемых параметров, так как это обязанность самой функции. Такое соглашение упрощает процесс вызова функций и обеспечивает более удобное и надежное управление памятью в языке LOK-2023.

## 1.23 Объектный код

Объектный код реализован на основе языке программирования JavaScript.

## 1.24 Классификация сообщений транслятора

Классификация сообщений транслятора приведена в таблице 1.11.

Таблица 1.11 – Сообщения транслятора

|  |  |
| --- | --- |
| Код ошибки | Описание ошибок |
| 0-110 | Системные ошибки, ошибки параметров |
| 111-199 | Ошибки лексического анализа |
| 600-639 | Ошибки семантического анализа |
| 640-699 | Ошибки синтаксического анализа |
| 200-599, 700-999 | Зарезервированные коды ошибок |

## 1.25 Контрольный пример

Исходный код контрольного примера представлен в приложении А.

# 2 Структура транслятора

## 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Транслятор языка программирования LOK-2023 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 JavaScript 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 LOK-2023

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<имя in-файла> | Файл с исходным кодом на языке программирования LOK-2023, 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. | Значение по умолчанию:  <имя in-файла>.out |

Входные параметры транслятора – это специфические данные или настройки, которые передаются транслятору для обработки и преобразования исходного кода.

## 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 LOK-2023

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" log.txt | Файл с протоколом работы транслятора языка программирования LOK-2023. Содержит таблицу лексем |
| Выходной файл, заданный параметром "-out:" out.cpp | Результат работы программы – файл, содержащий исходный код на языке JavaScript |
| 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 |

Лог состоит из 4 файлов: сам log-файл, таблицы лексем, идентификаторов, разбора.

# 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 LOK-2023. 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 выражение a is num = 1.

Структура лексического анализатора представлена на рисунке 3.1.

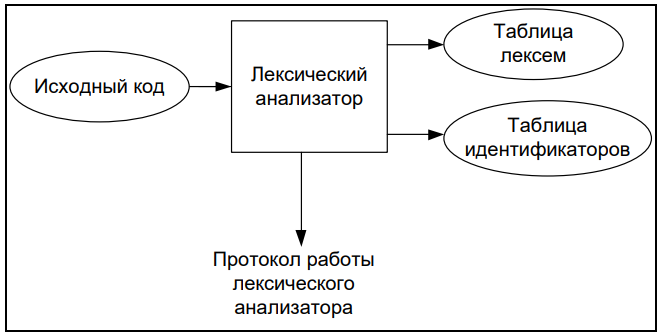


Рис. 3.1 Структура лексического анализатора

Данный рисунок содержит обработанный лексическим анализатором текст исходной программы

## 3.2 Входные и выходные данные лексического анализатора

Лексический анализатор принимает на вход два параметра: исходный текст программы, написанный на языке LOK-2023, и файл протокола, куда записываются результаты анализа в виде таблицы лексем и таблицы идентификаторов.

## 3.3 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.

## 3.4 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 3.2.

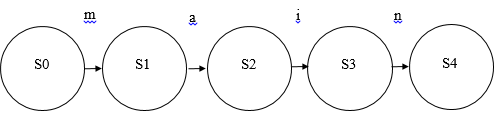


Рис 3.2 – Пример графа для цепочки main

## 3.5 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 |

В данной таблице перечислены и классифицированы все символы, которые могут быть использованы в качестве входных данных.

## 3.6 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы.

В языке программирования LOK-2023 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.

## 3.7 Перечень ключевых слов

Лексический анализатор выполняет преобразование исходного текста программы, заменяя лексические единицы на соответствующие лексемы, с целью создания промежуточного представления исходной программы. В таблице 3.2 приведено соответствие между ключевыми словами и лексемами.

Таблица 3.2 – Соответствие ключевых слов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Конструкция | Лексема | Примечание |
| num (ref num)  float (ref float)  symb (ref symb) | t | Названия типов данных языка. Ключевое слово ref обозначает то, что параметр будет передан по ссылке |
| Идентификатор | i | Максимальная 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 |

Пример реализованного конечного автомата ключевого слова main языка LOK-2023 представлен на рисунке 3.3.

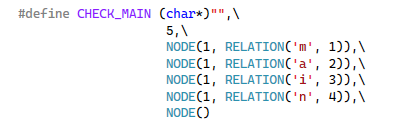


Рис. 3.3 – Реализация конечного автомата для ключевого слова main

Реализации графов переходов находятся в приложении B.

## 3.8 Основные структуры данных

Основные структуры данных языка LOK-2023 включают таблицу лексем (LexTable) и таблицу идентификаторов (IdTable).

Таблица лексем (LexTable) представляет собой структуру данных, которая содержит информацию о лексических единицах программы. Она имеет следующие поля:

* maxsize: максимальное количество элементов в таблице;
* size: текущее количество элементов в таблице;
* table: массив элементов типа LT::Entry, являющихся основным хранилищем лексем.

Каждый элемент массива в таблице лексем (Entry) представляет отдельную лексему и находится в пространстве имен LT. Он содержит следующие поля:

* lexema: значение лексемы;
* view: бинарная операция, используемая вместо лексемы 'v';
* sn: номер строки в исходном коде, соответствующей данной лексеме;
* idxTI: индекс данной лексемы в таблице идентификаторов. Используется только для идентификаторов или лексем, представляющих идентификаторы.

Таблица идентификаторов (IdTable) также имеет аналогичную структуру данных как у таблицы лексем. Ее поля аналогичны полям таблицы лексем, за исключением элементов массива table, которые являются структурами Entry, находящимися в пространстве имен IT. Эти элементы содержат следующие поля:

* idxfirstLE: индекс данного идентификатора в таблице лексем;
* countParams: количество параметров данной функции (используется только для функций);
* id: имя идентификатора;
* iddatatype: тип данных идентификатора, представленный перечислением (например, IT::ACTION, IT::NUM, IT::SYMB, IT::FLOAT, IT::STR);
* idtype: тип данного идентификатора (функция, литерал, параметр, переменная), также представленный перечислением;
* hasValue: булево значение, указывающее, инициализирован ли данный идентификатор;
* isRef: булево значение, указывающее, передается ли данный идентификатор в качестве параметра по ссылке;
* isFromStatic: булево значение, указывающее, является ли данный идентификатор из стандартной библиотеки;
* params: вектор, содержащий типы данных параметров функции. Используется для проверки передаваемых параметров в вызываемую функцию;
* value: объединение полей, используемых для хранения значений целочисленных, вещественных, символьных и строковых литералов.

## 3.9 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 |

Макрос ERROR\_ENTRY служит для удобства и эффективности работы с лексическим анализатором.

## 3.10 Принцип обработки ошибок

Лексический анализатор использует таблицу сообщений для обработки ошибок. В случае возникновения ошибок происходит их протоколирование и описание в командной строке с номером ошибки и сообщением.

## 3.11 Контрольный пример

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

# 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 LOK-2023 описывается грамматикой типа 2 по иерархии Хомского:

G = <T, N, P, S>

T – множество терминальных символов (алфавит языка LOK-2023),

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} |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминалы | Назначение | Правила |
| 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 |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
|  |  | (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 |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
|  |  | 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{} |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| 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 |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
|  |  | 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 |

Множество терминалов дает возможность наиболее точно вывести ошибку.

## 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 | 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. |

Продолжение таблицы 4.2

|  |  |  |
| --- | --- | --- |
|  | 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. |
|  | 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. |

Автомат с магазинной памятью, также известный как автомат с "стеком", представляет собой устройство, которое может принимать последовательность символов, выполнять определенные действия на основе входных данных и изменять свое состояние в соответствии с заранее заданными правилами.

## 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 Г.

## 4.5 Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора можно описать по следующей схеме:

1. В магазин записывается стартовый символ;
2. На основе полученной таблицы лексем формируется входная лента;
3. Запускается автомат и выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;
4. 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;
5. Если в магазине встретился нетерминал, переходим к пункту 3;
6. 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 Параметры синтаксического анализатора

Входными параметрами для синтаксического анализатора в языке программирования LOK-2023 являются таблица лексем и таблица идентификаторов.

## 4.7 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 конструкции 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 ‘)’ |

Количество правил соответствует количеству ошибок.

## 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 LOK-2023 синтаксическим анализатором представлен в приложении Д.

# 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 JavaScript.

## 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 LOK-2023 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 | Использована 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 |

Продолжение таблица 5.1

|  |  |
| --- | --- |
| 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 употреблена 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 |

Ошибка категории SEMANTIC является ошибкой, которая возникает, когда программа компилируется без ошибок, но при выполнении проявляются некорректные или непредсказуемые результаты.

## 5.4 Принцип обработки ошибок

При возникновении ошибки, семантический анализатор регистрирует ее в протоколе. Если возникает ошибка, следующий этап трансляции не будет запущен. Семантический анализатор начинает проверки уже на стадии лексического анализа, и, если обнаружены семантические ошибки на этой стадии, программа завершает свою работу, уведомляя пользователя о месте и характере ошибки.

**5.5 Контрольный пример**

Обработка ошибок семантического анализатора представлена в п. 8.2.

# 6 Преобразование выражений

## 6.1 Выражения, допускаемые языком

В языке программирования LOK-2023 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 LOK-2023

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

## 6.2 Польская запись и принцип ее построения

Польская запись представляет собой полезный принцип разбора выражений, который позволяет определить порядок и вызов операций с их аргументами. Однако, поскольку язык LOK-2023 транслируется на JavaScript, полное использование польской записи невозможно. Вместо этого она применяется на этапе семантического анализа для проверки, чтобы процедуры не использовались в выражениях, чтобы строковые литералы не могли быть использованы в выражениях, и чтобы, при использовании побитовых операций, тип данных float приводился к типу данных num, при этом уведомляя пользователя об этих ограничениях.

## 6.3 Программная реализация обработки выражений

Программная реализация обработки выражений представлена в приложении Е.

## 6.4 Контрольный пример

Пример преобразования выражения к обратной польской записи приведен в таблице 6.2.

Таблица 6.2. Пример выражения в польской записи

|  |  |
| --- | --- |
| Исходная строка | Результирующая строка |
| i = i&l|i; | i=il&i|; |

Выражение в обратной польской нотации читается слева направо: операция выполняется над двумя операндами, непосредственно стоящими перед знаком этой операции.

# 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 JavaScript, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться.

## 7.2 Представление типов данных в оперативной памяти

Переменные, функции и все другие части кода будут написаны на языке JavaScript на своих местах, где они написаны на языке LOK-2023. Соответствие типов данных языка LOK-2023 с типами данных JavaScript представлено в таблице 7.1.

Таблица 7.1 Соответствия типов идентификаторов

|  |  |  |
| --- | --- | --- |
| Тип идентификатора | Тип идентификатора на языке JavaScript | Пояснение |
| num | int | Хранит четырехбайтовое целое число |
| symb | char | Хранит символ из кодировки Windows-1251 |
| float | float | Хранит четырехбайтовое вещественное число |
| ref num | int& | Целочисленный параметр, передающий по ссылке |
| ref symb | char& | Символьный параметр, передающийся по ссылке |
| ref float | float& | Вещественный параметр, передающийся по ссылке |

## 7.3 Статическая библиотека

В языке LOK-2023 предусмотрена статическая библиотека, которая содержит функции, написанные на языке JavaScript, приведенные в таблице 7.2. Объявление функций статической библиотеки генерируется автоматически.

Таблица 7.2 Статическая библиотека

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Параметры | Возвращаемое выражение | Пояснение |
| sum | int/float a, int/float b | a + b | Арифметическая сумма |

Продолжение таблицы 7.2

|  |  |  |  |
| --- | --- | --- | --- |
| 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. |

Данные функции входят в стандартную библиотеку LOK-2023.

## 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 JavaScript;
* 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 Параметры, управляющие генерацией кода

Генератор кода является важным инструментом в процессе разработки программного обеспечения. Его главная задача заключается в преобразовании информации из таблиц лексем и идентификаторов в машинный код, который может быть исполнен компьютером. При этом результаты работы генератора кода выводятся в специальный файл с расширением .cpp.

Таблицы лексем и идентификаторов содержат информацию о различных синтаксических элементах программы и их соответствующих значениях. Лексемы представляют собой отдельные элементы языка программирования, такие как ключевые слова, операторы, константы и другие. Идентификаторы, в свою очередь, являются именами переменных, функций или других элементов программы, которые используются для их определения и обращения к ним в коде.

## 7.6 Контрольный пример

Контрольный пример JavaScript кода приведен в приложении Ж. Результат генерации кода приведен на рисунке 7.2.

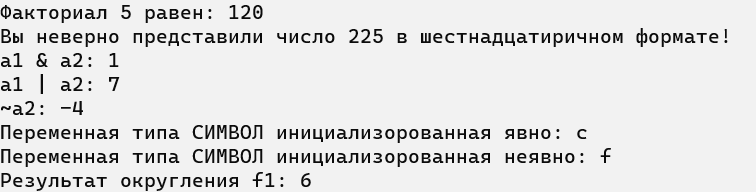


Рис.7.2 – Результат работы программы на языке LOK-2023

# 8 Тестирование транслятора

## 8.1 Общие положения

В процессе выполнения трансляции программы могут возникать ошибки на различных этапах. Эти этапы включают анализ исходного текста программы, лексический анализ, синтаксический анализ и семантический анализ. Когда возникают ошибки, транслятор отслеживает их и записывает в файл протокола. В этом файле указывается идентификатор ошибки, сообщение об ошибке, а также строка и позиция в исходном тексте программы, где ошибка была обнаружена. Обычно, после возникновения ошибки работа транслятора прекращается, поскольку ошибка на одном из этапов трансляции может привести к возникновению ошибок на последующих этапах, за исключением синтаксического анализатора.

## 8.2 Результаты тестирования

Результаты тестирования лексического анализатора показаны в таблице 8.1.

Таблица 8.1 Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| Проверка на допустимость символов | |
| main  {  a => ‘/’;  return 0;  } | Ошибка: 111. Недопустимый символ в исходном файле (‑in)  Строка: 3 Символ: 7 |
| Лексический анализ | |
| main  {  a is char = '1;  return 0;  } | Ошибка: 600: Нет закрывающей кавычки  Строка: 3 |
| main  {  console(5 + `kjtj`);  } | Ошибка: 601: Найдено неопознанное слово  Строка: 3 |
| main  {  console('\g');  } | Ошибка: 621 : Использован неверные управляющий символ(доступны только \n, \t) |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  a => 23  return 0;  } | Ошибка 670: Не хватает точки с запятой  Ошибка: 121: Ошибка в синтаксисе!  Строка: 4 |
| Синтаксический анализ | |
| fact is foo(a is num) is num  {  a = 5;  } | Ошибка 656: строка 4, Ошибка в теле функции  Ошибка 659: строка 4, Ошибка при возвращении значения функции |
| main  {  (5 == 5) : }  } | 661: строка 3, Ошибка в теле условного оператор |
| main  {  console 5  } | Ошибка 662: строка 3, Ошибка при вызове оператора console(). Неверный синтаксис. console(Значение); |
| main  {  a is num = 5;  For(5, 5, 5 a => {  });  } | Ошибка: 664 : Ошибка в параметрах цикла! |
| ff is foo(a is num, b is num) is num  {  return 5 console 5;  } | Ошибка 670: строка 3, Ошибка. Не хватает точки с запятой |
| Семантический анализ | |
| fact is foo(a is num)  {  res => 1;  For(1, a, 1, i =>  {  res = mult(res, i);  });  return res;  } | Ошибка: 605 : При инициализации функции невозможно узнать тип данных возвращаемого значения |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  a is num;  a is num;  } | Ошибка: 606: Ошибка! Повторное использование переменной  Строка: 3 |
| ff is foo(a is num, b is num) is num  {  res is num = mult(a, b);  return res;  }  main  {  a => 1;  b => 2;  c => 3;  res is num = ff(a, b, c);  } | Ошибка: 610 : Ошибка при вызове функции! Количество параметров не совпадает |
| fact is foo(a is num) is num  {  res => 1;  For(1, a, 1, 5 =>  {  res = mult(res, i);  });  return res;  } | Ошибка: 612. В лямбда выражении в цикле For передаваемый параметр должен быть идентификатор  Строка: 7 |
| main  {  break;  } | Ошибка: 616 : Ошибка. Оператор break может быть использован только в теле цикла! |
| func is foo(a is num) is num  {  return a;  }  main  {  res is ref num = 5;  } | Ошибка: 618 : Ошибка. Как ссылки можно инициализировать только параметры |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| func is foo(a is num) is num{  return a;  }  main{  res is num = func 5);  } | Ошибка: 620 : Синтаксическая ошибка. После имени функции ожидалась открывающая скобка '(' |
| main{  a is num = 5;  For(1, a, 55555555555555, i => {  console(i);  });  } | Ошибка: 624 : Ошибка. Превышено значение литерала типа num |
| main{  a is num = 5;  For(1, a, -55555555555555, i => {  console(i);  });  } | Ошибка: 625 : Ошибка. Слишком маленькое значение литерала типа num |
| func is foo(a is num) is action{  console(a);  }  main{  b is num = func(5);  } | Ошибка: 628 : Ошибка. Функцию типа action нельзя использовать в выражениях или передавать в качестве параметра |
| main{}  main  {} | Ошибка: 629 : Ошибка. Точка входа в программу main может быть описана лишь один раз |
| func is foo(a is num) is action{  return a;  }  main{  b is num = func(5);  } | Ошибка: 630 : Ошибка. Функция типа action не может ничего возвращать |
| main{  5 => 5;  } | Ошибка: 632 : Ошибка. Слева от оператора => должен быть незадействованный в данной области видимости идентификатор |

# Заключение

В ходе выполнения курсовой работы был разработан транслятор для языка программирования LOK-2023, а также генератор кода, который переводит в ассемблер. Были выполнены основные задачи курсового проекта:

– разработан дизайн языка;

– разработана спецификация языка программирования;

– разработана структура транслятора;

– разработан лексический анализатор;

– разработан синтаксический анализатор;

– разработан семантический анализатор;

– разработан генератор кода на язык JavaScript;

– проведено тестирование транслятора.

Итоговая версия языка LOK-2023 включает:

* 3 типа данных;
* поддержку операторов вывода;
* возможность вызова функций стандартной библиотеки;
* наличие бинарных операторов для вычисления выражений;
* наличие операторов сравнения для вычисления выражений;
* поддержка функций, процедур, условного оператор, цикличности.

Таким образом, в ходе выполнения курсового проекта были получены новые знания и навыки в проектировании систем программирования и в разработке программного обеспечения для систем программирования.

# Литература

1. 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, 2023. – 31 с

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

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

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

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

**Приложение А**

|  |
| --- |
| fact is foo(a is num) is num  {  res => 1;  For(1, a, 1, i =>  {  res = mult(res, i);  });  return res;  }  your is foo(n is num) is action  {  (n == x3FD) ? {  console(`Вы верно представили число `, n, ` в шестнадцатиричном формате!\n`);  } : {  console(`Вы неверно представили число `, n, ` в шестнадцатиричном формате!\n`);  }  }  main  {  n is num = 5;  res is num = fact(n);  console(`Факториал `, n, ` равен: `, res, '\n');  n1 is num = 225;  your(n1);  a1 is num = 5;  a2 => 3;  bit1 is num = a1 & a2;  bit2 is num = a1 | a2;  bit3 is num = ~a2;  console(`a1 & a2: `, bit1, '\n');  console(`a1 | a2: `, bit2, '\n');  console(`~a2: `, bit3, '\n');  c is symb = 'c';  console(`Переменная типа СИМВОЛ инициализорованная явно: `,c, '\n');  s => 'f';  console(`Переменная типа СИМВОЛ инициализорованная неявно: `,s, '\n');  f1 is float = 5.6;  @import round;  console(`Результат округления f1: `, round(f1));  } |

Листинг 1 – Реализация контрольного примера

**Приложение Б**

|  |
| --- |
| struct FST  {  char\* string;  short position;  short nstates;  NODE\* nodes;  short\* rstates;  FST(  char\* s, // цепочка  short ns, // количество состояний  NODE n, // граф переходов/ список состояний  ...  );  }; |

Листинг 2 – Реализация конечного автомата

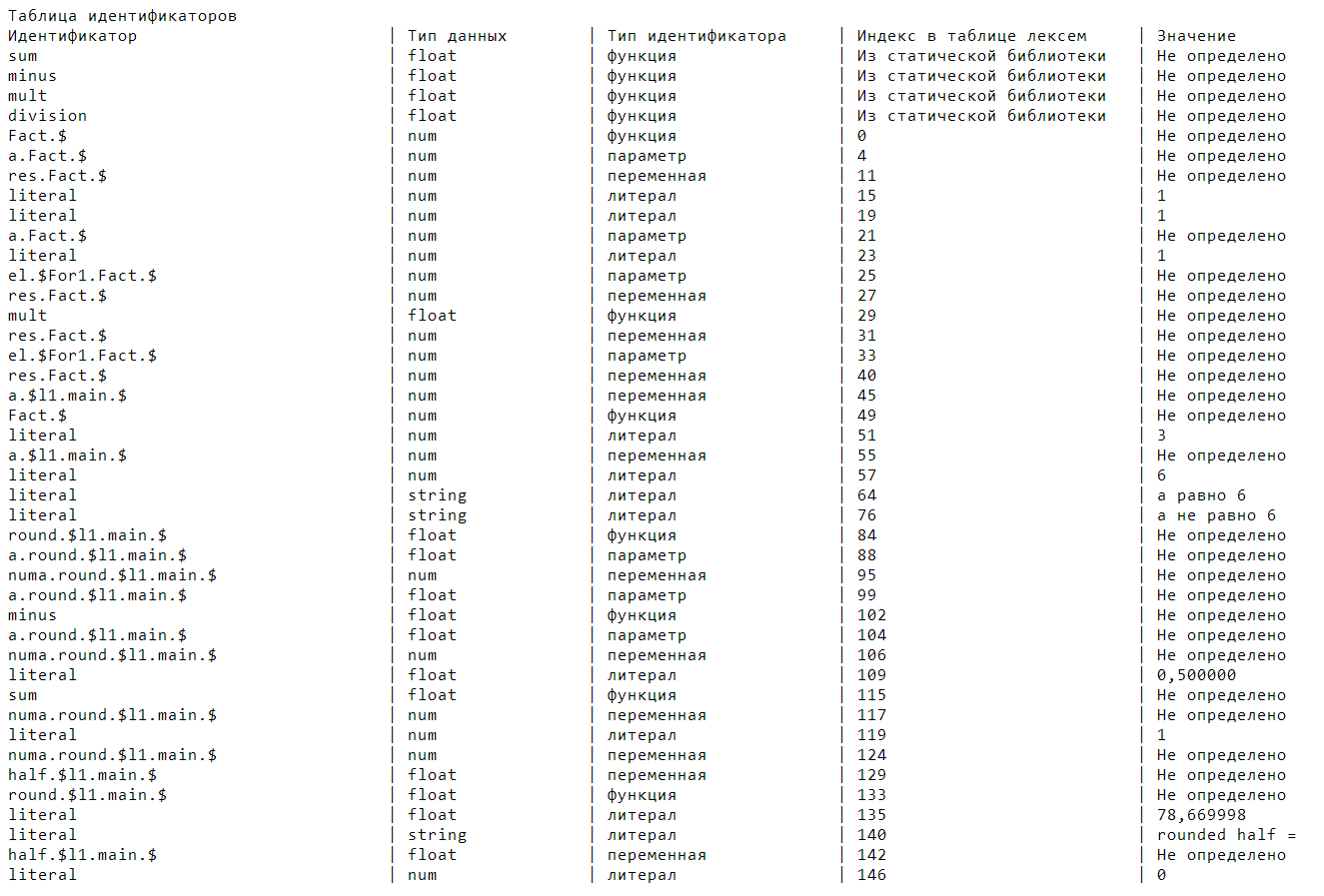


Рисунок 1 – Таблица идентификаторов

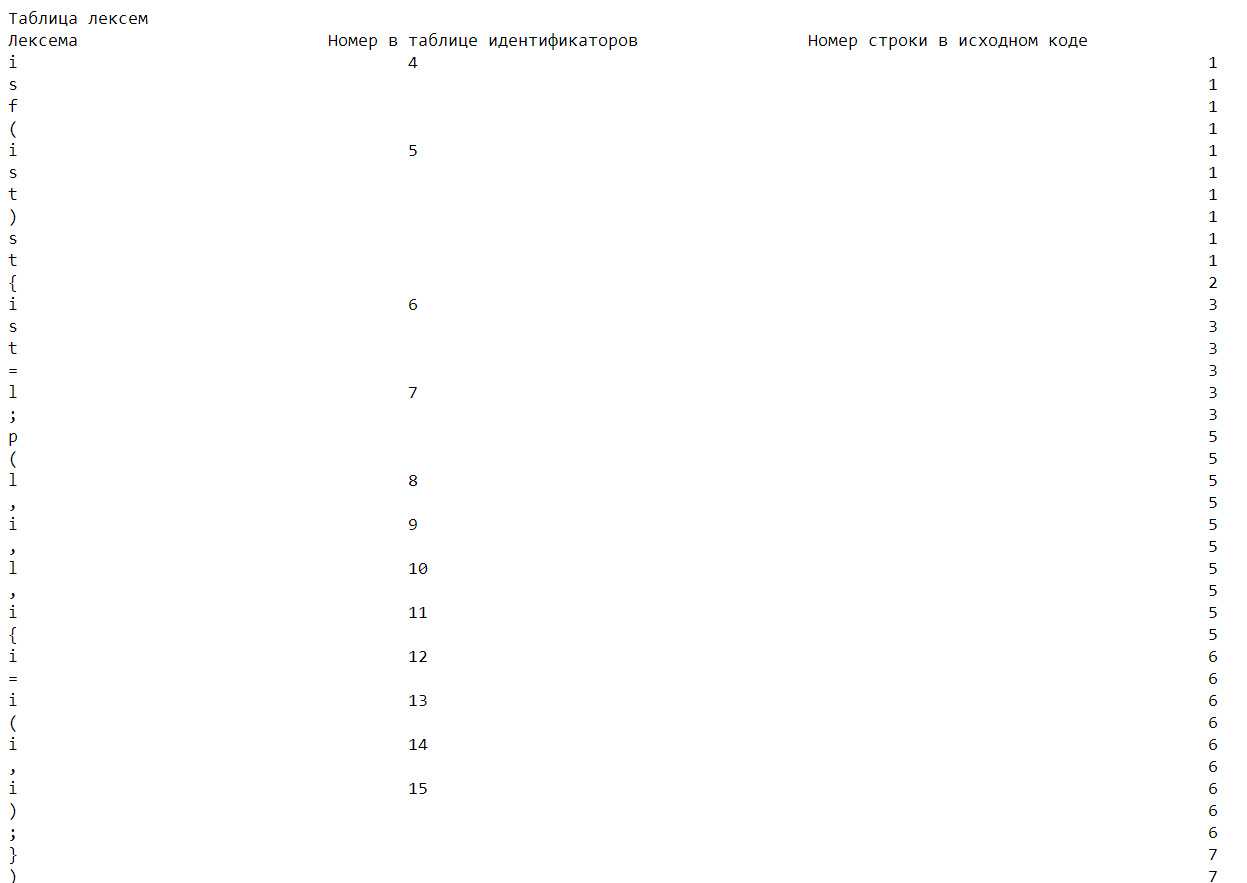


Рисунок 2 – Начало таблицы лексем

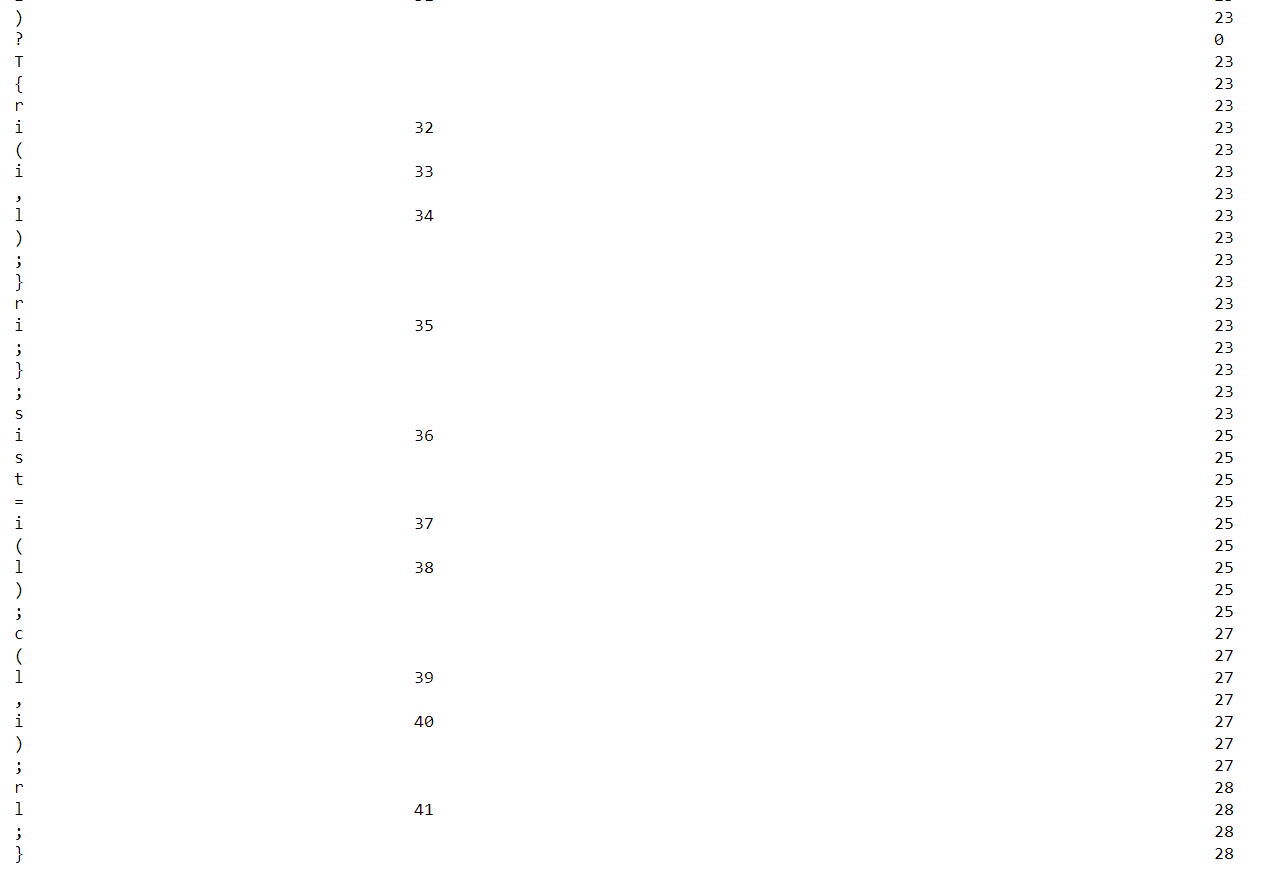


Рисунок 3 – Окончание таблицы лексем

**Приложение В**

|  |
| --- |
| GRB::Greibach greibach(  NS(GLOBAL),  TS('$'),  22, // edit  Rule(NS(GLOBAL), GRB\_ERROR\_SERIES + 0, // глобальное пространство  12,  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('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  24,  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::Chain(4, TS('r'), NS(EXPR), NS(SEMICOLON), NS(BODY\_FUNC)),  Rule::Chain(3, TS('r'), NS(EXPR), NS(SEMICOLON))  ),  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  38,  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(4, TS('r'), NS(EXPR), NS(SEMICOLON), NS(IF)),  Rule::Chain(3, TS('r'), NS(SEMICOLON), NS(IF)),  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,  38,  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(4, TS('r'), NS(EXPR), NS(SEMICOLON), NS(VISAREA)),  Rule::Chain(3, TS('r'), 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))  )  ); |

Листинг 3 – Правила языка LOK-2023

**Приложение Г**

|  |
| --- |
| struct Entry // запись в таблице идентификаторов  {  int idxfirstLE; // индекс первой строки в таблице лексем  short countParams; // кол-во параметров, если это функция  char id[ID\_MAXSIZE]; // идентификатор  IDDATATYPE iddatatype; // тип данных идентификатора  IDTYPE idtype; // тип идентификатора  bool hasValue;  bool isRef;  bool isFromStatic;  std::vector<std::string> params;  bool needToInt;  union  {  int vint; // значение для типа num  char vsymb; // значение типа symb  float vflt; // значение типа float  char vstr[255]; // значение типа string  } value;  Entry()  {  hasValue = false;  countParams = -1;  needToInt = false;  isRef = false;  isFromStatic = false;  }  }; |

Листинг 4 – Пример одной из основных структур данных

|  |
| --- |
| struct IdTable // экземпляр таблицы идентификаторов  {  int maxsize; // максимальное количество записей в таблице идентификаторов < TI\_MAXSIZE  int size; // текущий размер таблицы идентификаторов < maxsize  Entry\* table; // массив строк таблицы идентификаторов  }; |

Листинг 5 – Пример одной из основных структур данных

**Приложение Д**

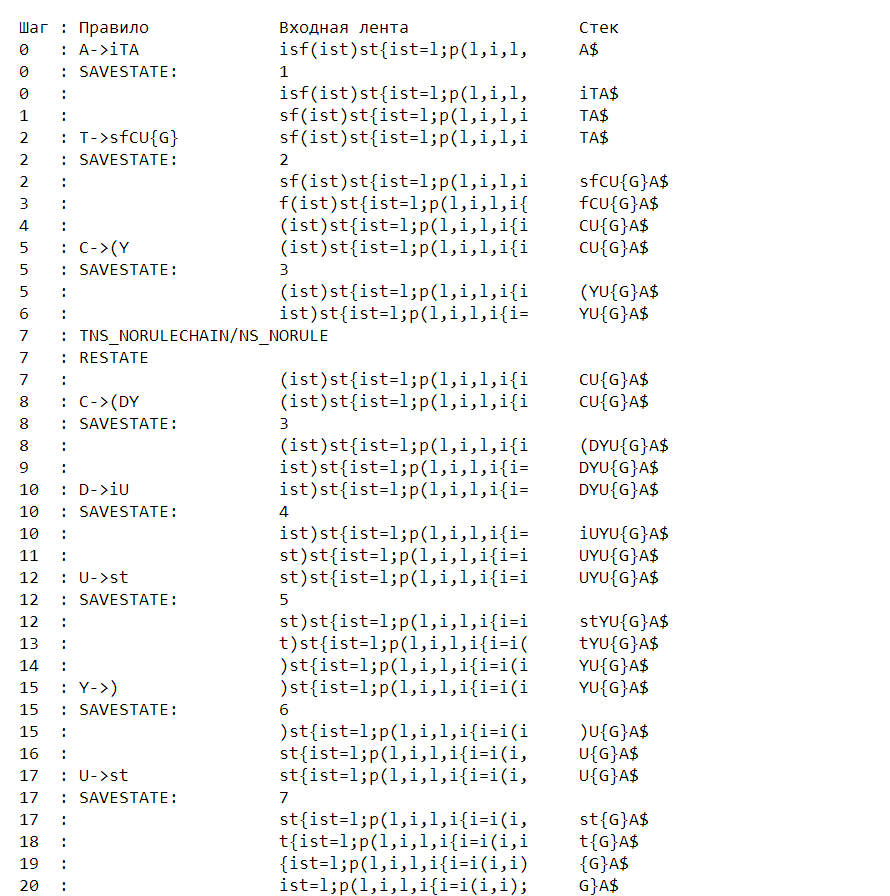


Рисунок 4 – Начало разбора синтаксическим анализатором исходного кода

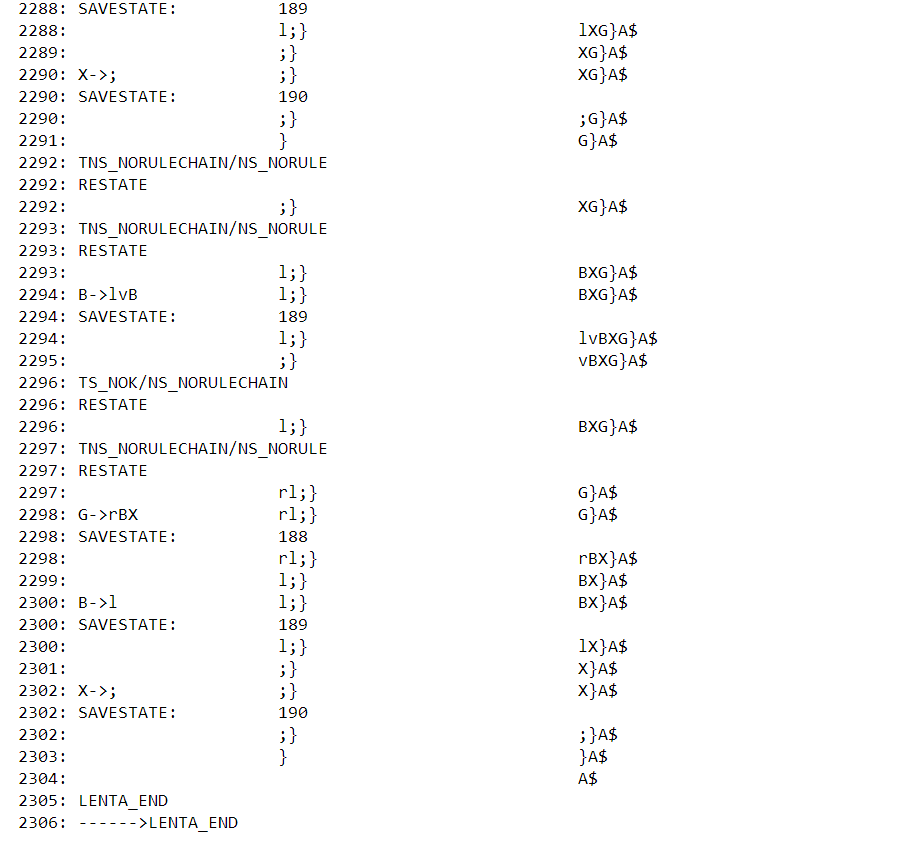


Рисунок 5 – Конец разбора синтаксическим анализатором исходного кода

**Приложение Е**

|  |
| --- |
| namespace PN  {  typedef char operation;  void go\_to\_pol(char\* expr, map<char, IT::Entry\*> dict, LT::LexTable& lextable, IT::IdTable& idtable);  bool Polish(LT::LexTable& lextable, IT::IdTable& idtable)  {  auto expr = new char[200];  auto cls = (char\*)"";  int size;  int index;  char\* polExprs;  char el;  map<char, IT::Entry\*> dict;  IT::Entry\* ide;  LT::Entry\* lte;  auto table = lextable.table;  bool isFunc;  int cntScopes;  for (int i = 0; i < lextable.size; i++)  {  if (table[i].lexema == '=' || table[i].lexema == 'r')  {  i++;  strcpy\_s(expr, cls);  dict.clear();  size = 0;  isFunc = false;  cntScopes = 0;  el = 'a';  while (table[i].lexema != ';')  {  if (table[i].lexema == '(' && table[i - 1].lexema == 'i')  {  isFunc = true;  cntScopes = 0;  }  if (!isFunc)  {  if (table[i].lexema != 'i' && table[i].lexema != 'l')  {  expr[size++] = (table[i].lexema == 'v' ? table[i].view : table[i].lexema);  }  else  {  expr[size++] = el++;  dict[el - 1] = &idtable.table[table[i].idxTI];  }  }  if (table[i].lexema == '(' && isFunc)  {  cntScopes++;  }  else if (table[i].lexema == ')' && isFunc)  {  cntScopes--;  if (cntScopes == 0)  {  isFunc = false;  }  }  i++;  }  expr[size] = 0;  polExprs = PolishNotation(expr);  go\_to\_pol(polExprs, dict, lextable, idtable);  }  }  delete[] expr;  return true;  }  map<operation, int> priorities =  {  {'(', 1},  {')', 1},  {'|', 2},  {'&', 3},  {'~', 4}  };  char\* PolishNotation(char\* expression)  {  /\*if (!checkExpression(expression))  {  return false;  }\*/  char\* polishExpression = new char[strlen(expression)];  strcpy\_s(polishExpression, strlen(expression), "");  goThrowStr(expression, polishExpression);  return polishExpression;  }  void go\_to\_pol(char\* expr, map<char, IT::Entry\*> dict, LT::LexTable& lextable, IT::IdTable& idtable)  {  ushort len = strlen(expr);  IT::Entry\* edi;  for (ushort i = 0; i < len; i++)  {  if (!isOperation(expr[i]))  {  edi = dict[expr[i]];  if (edi->iddatatype == IT::FLT && len > 1)  {  edi->needToInt = true;  cout << "| Предупреждение: переменная/функция/литерал ";  if (edi->idtype == IT::L)  {  cout << edi->value.vflt;  }  else  {  cout << edi->id;  }  cout << " приведена к типу num при выполение побитовых операций! Строка: " << lextable.table[edi->idxfirstLE].sn << " |\n";  }  else if (edi->iddatatype == IT::ACTION)  {  throw ERROR\_THROW\_IN(628, lextable.table[edi->idxfirstLE].sn, -1);  }  }  }  }  bool checkExpression(char\* expression)  {  if (  !checkBrackets(expression) ||  !checkOperations(expression)  )  {  return false;  }  return true;  }  bool checkBrackets(char\* expression)  {  stack<char> brackets;  for (int i = 0; i < strlen(expression); i++)  {  if (expression[i] == '(')  {  brackets.push(expression[i]);  }  else if (expression[i] == ')')  {  if (brackets.empty())  {  return false;  }  else  {  brackets.pop();  }  }  }  if (brackets.empty())  {  return true;  }  else  {  return false;  }  }  bool checkOperations(char\* expression)  {  for (int i = 0; i < strlen(expression); i++)  {  if (isOperation(expression[i]) && !isBrackets(expression[i]))  {  if (i == 0)  {  return false;  }  else if (i == strlen(expression) - 1 && !isBrackets(expression[i]))  {  return false;  }  else if (isOperation(expression[i - 1]) && !isBrackets(expression[i - 1]))  {  return false;  }  else if (isOperation(expression[i + 1]) && !isBrackets(expression[i + 1]))  {  return false;  }  }  }  return true;  }  void goThrowStr(char\* expr, char\* polExprs)  {  stack<operation> \_stack;  int polExprIndex = 0;  for (auto i = 0; i < strlen(expr); i++)  {  if (isOperand(expr[i]))  {  polExprs[polExprIndex++] = expr[i];  }  else if (isOperation(expr[i]))  {  if (\_stack.empty() ||  expr[i] == '(' ||  priorities[expr[i]] > priorities[\_stack.top()])  {  \_stack.push(expr[i]);  }  else  {  while (  !\_stack.empty() &&  priorities[expr[i]] <= priorities[\_stack.top()])  {  if (!isBrackets(\_stack.top()))  {  polExprs[polExprIndex++] = \_stack.top();  }  \_stack.pop();  }  \_stack.push(expr[i]);  }  }  }  polExprs[polExprIndex] = '\0';  if (!\_stack.empty())  {  writeTo(polExprs, \_stack);  }  }  void writeTo(char\* expr, stack<operation> \_stack)  {  int exprIndex = strlen(expr);  while (!\_stack.empty())  {  if (!isBrackets(\_stack.top()))  {  expr[exprIndex++] = \_stack.top();  }  \_stack.pop();  }  expr[exprIndex] = '\0';  }  bool isOperand(char symb)  {  return  (symb >= '0' && symb <= '9') ||  (symb >= 'a' && symb <= 'z') ||  (symb >= 'A' && symb <= 'Z');  }  bool isOperation(char symb)  {  return  symb == '&' ||  symb == '|' ||  symb == '~' ||  isBrackets(symb);  }  bool isBrackets(char symb)  {  return symb == '(' ||  symb == ')';  }  } |

Листинг 6 – Программная реализация польской нотации

**Приложение Ж**

|  |
| --- |
| #include <iostream>  #include <windows.h>  #pragma comment(lib, "D:\\CP\\CPP\\LIB.lib")  #include "D:\\CP\\LIB\\framework.h"  int fact\_$(int a\_fact\_$)  {  int res\_fact\_$ = 1;  For((int)1, (int)a\_fact\_$, 1, [&](int i\_$For1\_fact\_$)  {  res\_fact\_$ = mult((int)res\_fact\_$, (int)i\_$For1\_fact\_$);      return false;  });  return (int)(res\_fact\_$);    }  void your\_$(int n\_your\_$)  {  if (n\_your\_$ == 1021)  {  std::cout << "Вы верно представили число " << n\_your\_$ << " в шестнадцатиричном формате!\n";    }  else  {  std::cout << "Вы неверно представили число " << n\_your\_$ << " в шестнадцатиричном формате!\n";    }  }  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  int n\_$l4\_main\_$ = 5;  int res\_$l4\_main\_$ = fact\_$((int)n\_$l4\_main\_$);  std::cout << "Факториал " << n\_$l4\_main\_$ << " равен: " << res\_$l4\_main\_$ << '\n';  int n1\_$l4\_main\_$ = 225;  your\_$((int)n1\_$l4\_main\_$);  int a1\_$l4\_main\_$ = 5;  int a2\_$l4\_main\_$ = 3;  int bit1\_$l4\_main\_$ = a1\_$l4\_main\_$ & a2\_$l4\_main\_$;  int bit2\_$l4\_main\_$ = a1\_$l4\_main\_$ | a2\_$l4\_main\_$;  int bit3\_$l4\_main\_$ = ~a2\_$l4\_main\_$;  std::cout << "a1 & a2: " << bit1\_$l4\_main\_$ << '\n';  std::cout << "a1 | a2: " << bit2\_$l4\_main\_$ << '\n';  std::cout << "~a2: " << bit3\_$l4\_main\_$ << '\n';  char c\_$l4\_main\_$ = 'c';  std::cout << "Переменная типа СИМВОЛ инициализорованная явно: " << c\_$l4\_main\_$ << '\n';  char s\_$l4\_main\_$ = 'f';  std::cout << "Переменная типа СИМВОЛ инициализорованная неявно: " << s\_$l4\_main\_$ << '\n';  float f1\_$l4\_main\_$ = 5.6;  auto round\_$l4\_main\_$ = [&](float a\_round\_$l4\_main\_$)  {  int numa\_round\_$l4\_main\_$ = a\_round\_$l4\_main\_$;  if (minus((float)a\_round\_$l4\_main\_$, (float)numa\_round\_$l4\_main\_$) > 0.5)  {  return (float)(sum((int)numa\_round\_$l4\_main\_$, (int)1));    }  return (float)(numa\_round\_$l4\_main\_$);    };  std::cout << "Результат округления f1: " << round\_$l4\_main\_$((float)f1\_$l4\_main\_$);  return (int)(0);    } |

Листинг 7 – Результат генерации контрольного примера в язык JavaScript