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 Конструирование программного обеспечения

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора KAD-2023»

Выполнил студент Кантарович Андрей Дмитриевич

(Ф.И.О.)

Руководитель проекта ст. преп. Наркевич Аделина Сергеевна

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

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

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

Консультанты ст. преп. Наркевич Аделина Сергеевна

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

Нормоконтролер ст. преп. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[Введение 6](#_Toc153726190)

[1 Спецификация языка программирования 7](#_Toc153726191)

[**1.1 Характеристика языка программирования** 7](#_Toc153726192)

[**1.2 Определение алфавита языка программирования** 7](#_Toc153726193)

[**1.3 Применяемые сепараторы** 7](#_Toc153726194)

[**1.4 Применяемые кодировки** 7](#_Toc153726195)

[**1.5 Типы данных** 8](#_Toc153726196)

[**1.6 Преобразование типов данных** 8](#_Toc153726197)

[**1.7 Идентификаторы** 8](#_Toc153726198)

[**1.8 Литералы** 9](#_Toc153726199)

[**1.9 Объявление данных** 9](#_Toc153726200)

[**1.10 Инициализация данных** 9](#_Toc153726201)

[**1.11 Инструкции языка** 9](#_Toc153726202)

[**1.12 Операции языка** 10](#_Toc153726204)

[**1.13 Выражения и их вычисления** 10](#_Toc153726205)

[**1.14 Конструкции языка** 10](#_Toc153726206)

[**1.15 Область видимости идентификаторов** 11](#_Toc153726207)

[**1.16 Семантические проверки** 11](#_Toc153726208)

[**1.17 Распределение оперативной памяти на этапе выполнения** 12](#_Toc153726209)

[**1.18 Стандартная библиотека и ее состав** 12](#_Toc153726210)

[**1.19 Ввод и вывод данных** 13](#_Toc153726211)

[**1.20 Точка входа** 13](#_Toc153726212)

[**1.21 Препроцессор** 13](#_Toc153726213)

[**1.22 Соглашения о вызовах** 13](#_Toc153726214)

[**1.23 Объектный код** 13](#_Toc153726215)

[**1.24 Классификация сообщений** 13](#_Toc153726216)

[**1.25 Контрольный пример** 14](#_Toc153726217)

[2 Структура транслятора 15](#_Toc153726218)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 15](#_Toc153726219)

[**2.2 Перечень входных параметров транслятора** 16](#_Toc153726220)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 17](#_Toc153726221)

[3 Разработка лексического анализатора 18](#_Toc153726222)

[**3.1 Структура лексического анализатора** 18](#_Toc153726223)

[**3.2 Контроль входных символов** 19](#_Toc153726224)

[**3.3 Удаление избыточных символов** 19](#_Toc153726225)

[**3.4 Перечень ключевых слов** 20](#_Toc153726226)

[**3.5 Основные структуры данных** 22](#_Toc153726227)

[**3.6 Принцип обработки ошибок** 23](#_Toc153726228)

[**3.7 Структура и перечень сообщений лексического анализатора** 23](#_Toc153726229)

[**3.8 Параметры лексического анализатора** 23](#_Toc153726230)

[**3.9 Алгоритм лексического анализа** 23](#_Toc153726231)

[**3.10 Контрольный пример** 24](#_Toc153726232)

[4 Разработка синтаксического анализатора 25](#_Toc153726233)

[**4.1 Структура синтаксического анализатора** 25](#_Toc153726234)

[**4.2 Контекстно-свободная грамматика, описывающая синтаксис языка** 25](#_Toc153726235)

[**4.4 Основные структуры данных** 27](#_Toc153726236)

[**4.5 Описание алгоритма синтаксического разбора** 27](#_Toc153726237)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 27](#_Toc153726238)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 28](#_Toc153726239)

[**4.8 Принцип обработки ошибок** 28](#_Toc153726240)

[**4.9 Контрольный пример** 28](#_Toc153726241)

[5 Разработка семантического анализатора 29](#_Toc153726242)

[**5.1 Структура семантического анализатора** 29](#_Toc153726243)

[**5.2 Функции семантического анализатора** 29](#_Toc153726244)

[**5.3 Структура и перечень сообщений семантического анализатора** 29](#_Toc153726245)

[**5.4 Принцип обработки ошибок** 30](#_Toc153726246)

[**5.5 Контрольный пример** 30](#_Toc153726247)

[6 Вычисление выражений 31](#_Toc153726248)

[**6.1 Выражения, допускаемые языком** 31](#_Toc153726249)

[**6.2 Польская запись и принцип её построения** 31](#_Toc153726250)

[**6.3 Программная реализация обработки выражений** 32](#_Toc153726251)

[**6.4 Контрольный пример** 32](#_Toc153726252)

[7 Генерация кода 33](#_Toc153726253)

[**7.1 Структура генератора кода** 33](#_Toc153726254)

[**7.2 Представление типов данных в оперативной памяти** 33](#_Toc153726255)

[**7.3 Статическая библиотека** 34](#_Toc153726256)

[**7.4 Особенности алгоритма генерации кода** 34](#_Toc153726257)

[**7.5 Входные параметры генератора кода** 35](#_Toc153726258)

[**7.6 Контрольный пример** 35](#_Toc153726259)

[8 Тестирование транслятора 36](#_Toc153726260)

[**8.1 Тестирование проверки на допустимость символов** 36](#_Toc153726261)

[**8.2 Тестирование лексического анализатора** 36](#_Toc153726262)

[**8.3 Тестирование синтаксического анализатора** 37](#_Toc153726263)

[**8.4 Тестирование семантического анализатора** 39](#_Toc153726264)

[Список использованных источников 42](#_Toc153726265)

[Приложение А 43](#_Toc153726266)

[Приложение Б 45](#_Toc153726267)

[Приложение В 46](#_Toc153726268)

[Приложение Г 67](#_Toc153726269)

[Приложение Д 69](#_Toc153726270)

[Приложение Е 75](#_Toc153726271)

# **Введение**

В данном курсовом проекте мною поставлена задача создать удобный, понятный язык программирования KAD-2023, написанный на языке C++, и транслятор для данного языка, который будет транслировать KAD-2023 в язык ассемблера.  
 Транслятор – программа или средство, выполняющее преобразование программы, представленной на одном из языков программирования, в программу, написанную на другом языке.

В данном курсовом проекте поставлены следующие задачи:

1. Разработка спецификации KAD-2023;
2. Разработка структуры трансляции;
3. Разработка лексического анализатора;
4. Разработка синтаксического анализатора;
5. Разработка семантического анализатора;
6. Разбор арифметических выражений;
7. Тестирование транслятора.

# **1 Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

KAD-2023 является процедурным, строго типизированным языком программирования, который транслируется на язык ассемблера. KAD-2023 является языком общего назначения.

## **1.2 Определение алфавита языка программирования**

В алфавит языка программирования KAD-2023 входят символы латиницы ([a-z] [A-Z]), кириллицы ([а-я] [А-Я]), а так же символы операций (+ - \* / %, <, >) и сепараторов ({} () ; , пробел) . Таблица входных символов представлена в пункте 3.2.

## **1.3 Применяемые сепараторы**

Сепараторы в языке программирования KAD-2023 представлены в таблице 1.1.

Таблица 1.1 – Сепараторы языка KAD-2023

|  |  |
| --- | --- |
| Сепаратор | Правила и назначение |
| Пробел | Служит для разделения идентификаторов и ключевых слов. Не может использоваться в них. |
| Точка с запятой ; | Разделяет инструкции |
| Скобки ( ) | Параметры. Определение приоритетности операции в выражении. |
| Фигурные скобки { } | Определяет программный блок. |
| Запятая , | Разделитель параметров. |

Сепараторы – это программные конструкции, служащие для разделения блоков кода.

## **1.4 Применяемые кодировки**

Кодировка, используемая для написания программ на языке KAD-2023 – кодировка Windows-1251. Таблица символов Windows-251 представлена на рисунке 1.1.



Рисунок 1.1 – Таблица символов кодировки Windows-1251

Windows-1251 – набор символов и кодировка, являющаяся 8-битной кодировкой для русских версий Windows.

## **1.5 Типы данных**

Тип данных представляет собой набор или группировку значений данных, обычно задаваемых набором возможных значений, набором разрешенных операций над этими значениями и/или представлением этих значений в виде типов компьютеров.

Типы данные, реализованные в языке программирования KAD-2023:

* Целочисленный (2 байта)
* Строковый

## **1.6 Преобразование типов данных**

В языке программирования KAD-2023 преобразование типов не предусмотрено.

## **1.7 Идентификаторы**

Идентификаторы – имена, задаваемые в программе для переменных, типов и функций. Идентификаторы не должны совпадать с ключевыми словами. Имя идентификатора создается по следующим правилам:

* cостоит из символов [a-z], [A-Z], [0-9] и ‘\_’;
* длина идентификаторов не должна превышать 20 символов.

## **1.8 Литералы**

Литералы – неизменяемые значения, или константы. Являются значением переменных. В языке программирования KAD-2023 реализованы строковые и целочисленные литералы. Литералы являются rvalue, выражениями, которые не представляют собой объекты, которые занимают идентифицируемое место в памяти.   
 rvalue - это то, что можно только присваивать, например, литералы или результаты выражений.

## **1.9 Объявление данных**

Объявление переменных осуществляется с помощью ключевого слова var. Объявление функций осуществляется с помощью ключевого слова function.

## **1.10 Инициализация данных**

Объектами-инициализаторами могут быть только идентификаторы или литералы. При объявлении предусмотрены значения по умолчанию: значение 0 для типа short и строка длины 0 (“”) для типа str. Также возможна инициализация непосредственно при объявлении переменной.

## **1.11 Инструкции языка**

В языке программирования KAD-2023 реализованы некоторые инструкции (таблица 1.2).

Таблица 1.2 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Запись инструкции |
| Объявление переменной | <тип данных> var <идентификатор>; |
| Вывод | write(<литерал>|<идентификатор>); |
| Возврат значения | return <литерал>|<идентификатор>; |
| Присваивание | <идентификатор> = <выражение>|<литерал>|<идентфикатор>; |
| Вызов функции | <тип данных> var <идентификатор> = <идентификатор>([параметр][,]); |

Функции в языке KAD-2023 вызываются только через присваивание функции к переменной.

## **1.12 Операции языка**

В языке программирования KAD-2023 реализованны некоторые операции.

Арифметические операции:

* Сложение (+)
* Вычитание (-)
* Умножение (\*)
* Деление нацело (/)
* Деление по модулю (%)

Операции сравнения:

* Больше (>)
* Меньше (<)

## **1.13 Выражения и их вычисления**

Выражение — комбинация значений или переменных, [констант](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D0%B0%D0%BD%D1%82%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), п[еременных](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [операций](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), которая может быть [интерпретирована](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80) в соответствии с правилами конкретного языка. Выражения в языке программирования KAD-2023 относятся только к целочисленному типу.

Выражения составляются по следующим правилам:

* Просмотр выражения идет слева направо;
* Приоритет последовательности операция можно изменить с помощью круглых скобок ();
* Выражение может иметь вызов функции;
* Выражение считывается до символа сепаратора точки с запятой.

## **1.14 Конструкции языка**

Программные конструкции языка, реализованные в языке программирования KAD-2023 (таблица 1.3).

Таблица 1.3 – Конструкции языка

|  |  |  |
| --- | --- | --- |
| Конструкция | Описание | Реализация |
| Главная функция | Начало работы кода | main { <инструкции языка> } |

Таблица 1.3 (продолжение) – Конструкции языка

|  |  |  |
| --- | --- | --- |
| Конструкция | Описание | Реализация |
| Функция | Именованный блок кода | <тип> function <идентификатор> ([<тип данных> param <идентификатор>][, <тип данных> param <идентификатор>] ) {  <инструкции языка> return <идентификатор>|<литерал> } |
| Цикл | Повторение блока кода определенное количество раз | repeat(<выражение>) { <инструкции языка> } |

Конструкция языка – синтаксически допустимая часть программы, которая может быть сформирована из одного или нескольких лексических ключевых слов в соответствии с правилами языка программирования.

## **1.15 Область видимости идентификаторов**

Область видимости идентификаторов может быть локальной и глобальной. Локальные идентификаторы – идентификаторы, объявленные внутри функции. Глобальные идентификаторы – идентификаторы, объявленные вне функций.

## **1.16 Семантические проверки**

В языке программирования KAD-2023 есть следующиеправила семантической проверки исходного текста языка (таблица 1.4)

Таблица 1.4 – Семантические проверки языка KAD-2023

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Деление на ноль |
| 2 | Совпадение типов данных |
| 3 | Корректность строковых выражений справа от знака равно |
| 4 | Проверка на возвращающее значение из функции |
| 5 | Передаваемые параметру в функцию при вызове |
| 6 | Корректность условного выражения |
| 7 | Переопределение типов данных |
| 8 | Выход за пределы области видимости |

Таблица 1.4 (продолжение) – Семантические проверки языка KAD-2023

|  |  |
| --- | --- |
| Номер | Правило |
| 9 | Усечение слишком длинного значения string-литерала |
| 10 | Округление слишком большого значения short-литерала |
| 11 | Точка входа main |
| 12 | Проверка типа идентификаторов |
| 13 | Проверка лексем |
| 14 | Если ошибка возникает на этапе лексического анализа, синтаксический  анализ не выполняется |
| 15 | При возникновении ошибки в процессе синтаксического анализа,  ошибочная фраза игнорируется (предполагается, что ее нет) и  осуществляется попытка разбора следующей фразы. Граница фразы –  точка с запятой. |
| 16 | Если 3 подряд фразы не разобраны, то работа транслятора останавливается |

Назначение семантического анализа – проверка смысловой правильности конструкций языка программирования.

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

Транслированный код использует две области памяти. В сегмент констант заносятся все литералы. В сегмент данных заносятся переменные, параметры функций. Локальная область видимости в исходном коде определяется за счет использования переменной, хранящей имя родительского блока, что и обуславливает их локальность на уровне ассемблерного кода, в языке нет глобальных переменных, все глобальные функции имеют имя родительского блока “global”.

## **1.18 Стандартная библиотека и ее состав**

В стандартной библиотеке языка программирования KAD-2023 реализованы некоторые функции (таблица 1.5)

Таблица 1.5 – Функции стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Назначение | Параметры | Возвращаемое значение |
| copy | Копирование n символов строки str2 в строку str1 | str param str1, str param str2, short param n | str |

Таблица 1.5 (продолжение) – Функции стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Назначение | Параметры | Возвращаемое значение |
| lenght | Вычисление длины строки | str | short |
| random | Вывод случайного числа в диапазоне от a до b | short param a, short param b | short |
| getLocalTimeAndDate | Вывод текущей даты и времени | - | str |

Функции стандартной библиотеки добавляются в таблицу идентификаторов на этапе лексического анализа.

## **1.19 Ввод и вывод данных**

Вывод в языке программирования KAD-2023 осуществляется с помощью ключевого слова “write”. При каждом использовании write вывод начинается с новой строки.

## **1.20 Точка входа**

Точкой входа является ключевое слово “main”.

## **1.21 Препроцессор**

Препроцессор в языке программирования KAD-2023 не предусмотрен.

## **1.22 Соглашения о вызовах**

Используется соглашение stdcall, все параметры передаются в стек справа налево, память освобождается вызываемым кодом.

## **1.23 Объектный код**

Код на языке KAD-2023 транслируется в исходный код на языке ассемблера.

## **1.24 Классификация сообщений**

Сообщения об ошибках трансляции классифицируются следующим образом:

Таблица 1.6 – Номера ошибок и их характеристика

|  |  |
| --- | --- |
| Номера ошибок | Характеристика |
| 0 – 99 | Системные ошибки |
| 100 – 104 | Ошибки входных параметров |
| 105 – 109 | Ошибки при открытии файла |
| 110 – 119 | Ошибки при чтении файла |
| 120 – 140 | Ошибки лексического анализа |
| 600 – 610 | Ошибки синтаксического анализа |
| 700 – 710 | Ошибки семантического анализа |

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

## **1.25 Контрольный пример**

Контрольный пример показывает работу всех функций и показывает особенности языка KAD-2023. Исходный код контрольного примера представлен в приложенииА.

# **2 Структура транслятора**

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

В языке KAD-2023 исходный код транслируется в язык Assembler. Транслятор языка разделён на отдельные части, которые взаимодействуют между собой и выполняют отведённые им функции, которые представлены в пункте 2.1. Для того чтобы получить ассемблерный код, используется выходные данные работы лексического анализатора, а именно таблица лексем и таблица идентификаторов. Для указания выходных файлов используются входные параметры транслятора, которые описаны в таблице 2.1. Структура транслятора языка KAD-2023 приведена на рисунке 1.

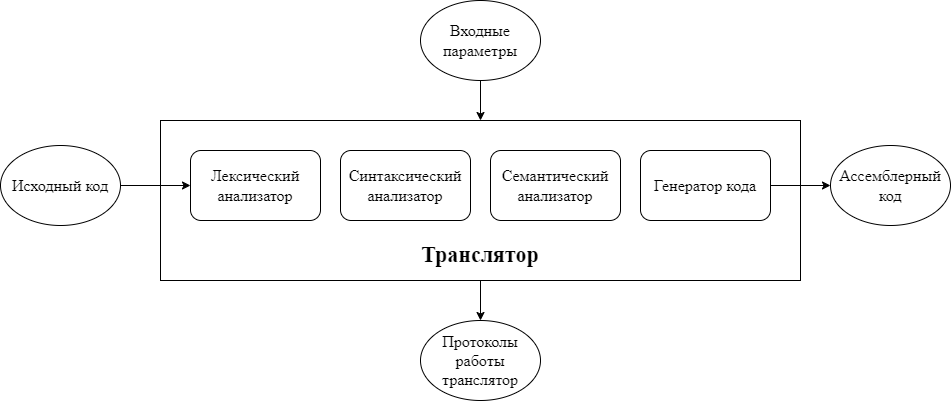


Рисунок 2.1 – Структура транслятора языка KAD-2023

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся последовательность символов входного языка. Он производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив отдельных слов (в теории компиляции вместо термина «слово» часто используют термин «токен»). Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация. Таблица лексем (ТЛ) и таблица идентификаторов (ТИ) являются входом для следующей фазы компилятора – синтаксического анализа (разбора, парсера).

Цели лексического анализатора:

* убрать все лишние пробелы;
* выполнить распознавание лексем;
* построить таблицу лексем и таблицу идентификаторов;
* при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

Синтаксический анализатор – часть компилятора, выполняющая синтаксический анализ, то есть проверку исходного кода на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией является дерево разбора

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

Генератор кода – часть транслятора, выполняющая генерацию ассемблерного кода 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 параметров 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 KAD-2023

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

Входные параметры указываются через командную строку вручную.

## **2.3 Перечень протоколов, формируемых транслятором и их содержимое**

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 KAD-2023

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования KAD-2023. Содержит информацию про входные параметры, общем количестве символов и строк(исходные данные), протокол работы синтаксического анализатора, полученный на этапе синтаксического анализа. |
| Выходной файл с названием "Table.id.txt" | Файл содержит таблицу идентификаторов, сформированную во время лексического анализа. |
| Выходной файл с названием "Table.lex.txt" | Файл содержит таблицу лексем, сформированную во время лексического анализа. |
| Выходной файл с названием "PolishNotation.lex.txt" | Файл содержит таблицу лексем, выражения в которой написаны обратной польской нотацией. |
| Выходной файл, заданный параметром "-out:" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

Файлы таблиц создаются в папке Tables. Файлы log и out создаются в корневой папке.

# **3 Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся исходный код входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка. Лексический анализатор производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив токенов.

Функции лексического анализатора:

* удаление «пустых» символов и комментариев. Если «пустые» символы (пробелы, знаки табуляции и перехода на новую строку) и комментарии будут удалены лексическим анализатором, синтаксический анализатор никогда не столкнется с ними (альтернативный способ, состоящий в модификации грамматики для включения «пустых» символов и комментариев в синтаксис, достаточно сложен для реализации);
* распознавание идентификаторов и ключевых слов;
* распознавание констант;
* распознавание разделителей и знаков операций.

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



Рисунок 3.1 – Структура лексического анализатора

Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация.

## **3.2 Контроль входных символов**

Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. Таблица входных символов представлена на рисунке 3.2, категории входных символов представлены в таблице 3.1.

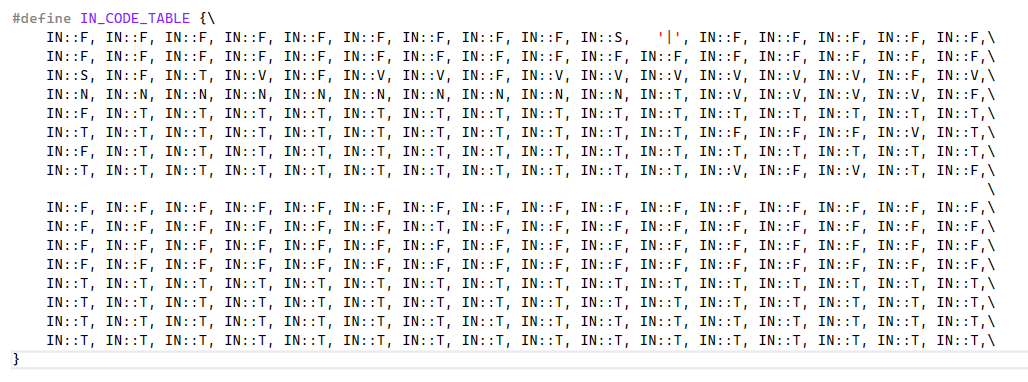


Рисунок 3.2 – Таблица контроля входных символов

Таблица 3.1 - Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Значение в таблице входных символов | Символы |
| Разрешенный | T |
| Запрещенный | F |
| Игнорируемый | I |
| Пробел, табуляция | S |
| Операторы | V |
| Цифры | N |

Если символ во входном файле не отмечен в таблице контроля, лексический анализатор выдаст соответствующую ошибку.

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на токены.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем файл с исходным кодом программы;
2. Встреча пробела или знака табуляции является своего рода встречей символа-сепаратора;
3. В отличие от других символов-сепараторов не записываем в очередь лексем эти символы, т.е. игнорируем.

## **3.4 Перечень ключевых слов**

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие токенов и лексем приведено в таблице 3.2.

Таблица 3.2 - Соответствие токенов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| short, str | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 20 символов. |
| Литерал | l | Литерал любого доступного типа. |
| function | f | Объявление функции. |
| var | v | Объявление переменной |
| param | p | Объявление параметра функции |
| return | r | Выход из функции/процедуры. |
| main | m | Главная функция. |
| write | w | Вывод данных |
| repeat | ~ | Указывает на начало тела цикла. |
| ; | ; | 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. |

Таблица 3.2 (окончание) - Соответствие токенов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| +  -  \*  /  % | +  -  \*  /  % | Знаки операций. |
| >  < | >  < | Знаки логических операторов. |
| # | нет | Комментарии. |

Пример реализации таблицы лексем представлен в приложении Б.

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся токен и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Структура конечного изображена на рисунке 3.3.

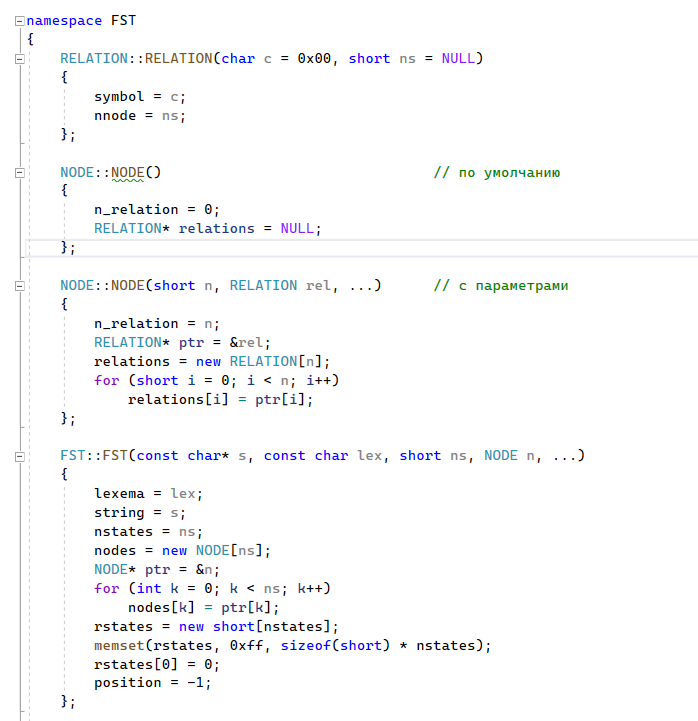


Рисунок 3.3 – Структура конечного автомата

Пример графа перехода конечного автомата изображен на рисунке 3.4.

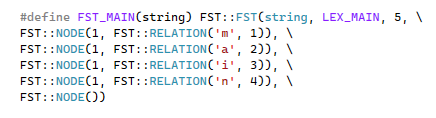


Рисунок 3.4 – Пример реализации графа КА для токена main

Пример работы графа перехода представлен на рисунке 3.6.

## **3.5 Основные структуры данных**

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Таблица лексем содержит номер лексемы, лексему (lexema), полученную при разборе, знак оператора (sign), номер строки в исходном коде (sn), номер токена (st), и номер в таблице идентификаторов, если лексема является идентификатором (idxTI). Код C++ со структурой таблицы лексем представлен на рисунке 3.5.

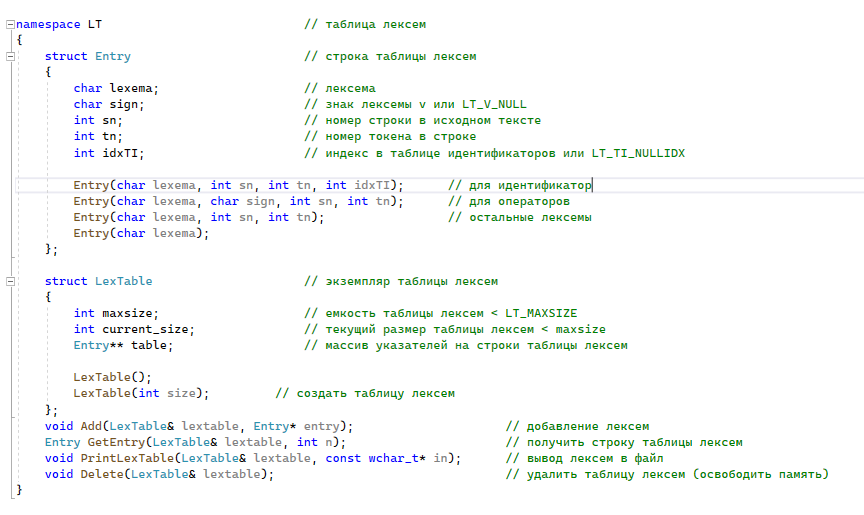


Рисунок 3.5 - Структура таблицы лексем

Таблица идентификаторов содержит имя идентификатора (id), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype) и значение (или параметры функций) (value). Код C++ со структурой таблицы идентификаторов представлен в приложении \_.

## **3.6 Принцип обработки ошибок**

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. Перечень сообщений об ошибках лексического анализатора представлен на рисунке 3.6.

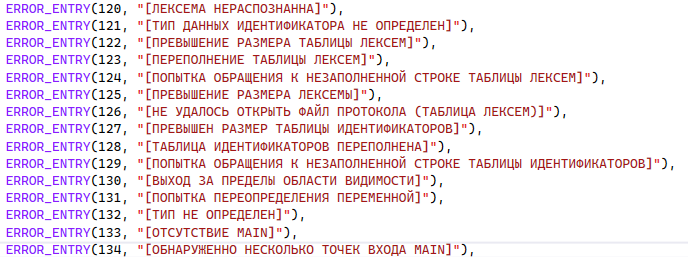


Рисунок 3.6 - Структура таблицы лексем

При возникновении сообщения, лексический анализатор выбрасывает исключение – работа программы останавливается.

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

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением.

## **3.8 Параметры лексического анализатора**

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся в два отдельных файла – Table.lex.txt и Table.id.txt .

## **3.9 Алгоритм лексического анализа**

1. Проверяет входной поток символов программы на исходном языке на допустимость, удаляет лишние пробелы и добавляет сепаратор для вычисления номера строки для каждой лексемы;
2. Для выделенной части входного потока выполняется функция распознавания лексемы;
3. При успешном распознавании информация о выделенной лексеме заносится в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к первому этапу;
4. Формирует протокол работы;
5. При неуспешном распознавании выдается сообщение об ошибке.

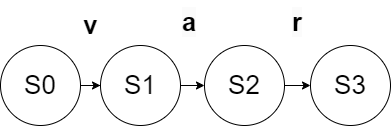


Рисунок 3.6 - Пример графа переходов для цепочки var

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «var» представлен на рисунке 3.2, где S0 – начальное, а S3 – конечное состояние автомата.

## **3.10 Контрольный пример**

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

4 Разработка синтаксического анализатора

## **4.1 Структура синтаксического анализатора**

Синтаксический анализатор: часть компилятора, выполняющая синтаксический анализ, то есть исходный код проверяется на соответствие правилам грамматики.

Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

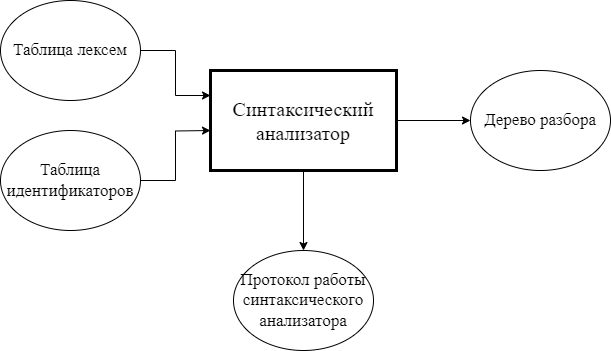


Рисунок 4.1 Структура синтаксического анализатора

Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией– дерево разбора

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

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

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

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

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

S – начальный символ грамматики, являющийся нетерминалом.

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

Описание нетерминальных символов содержится в таблице 1 приложения В.

**4.3 Построение конечного магазинного автомата**

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

Таблица 4.1 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит представляет из себя множества терминальных и нетерминальных символов, описание которых содержится в таблица 3.1 и 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $) |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики |
|  | Начальное состояние магазина автомата | Символ маркера дна стека $ |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

Автомат с магазинной памятью – это, по существу, недетерминированный конечный автомат, имеющий дополнительную потенциально неограниченную ленту памяти (магазин).

## **4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка KAD-2023. Данные структуры в приложении В.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата, следующий:

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

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

Перечень сообщений синтаксического анализатора представлен на рисунке 4.3.

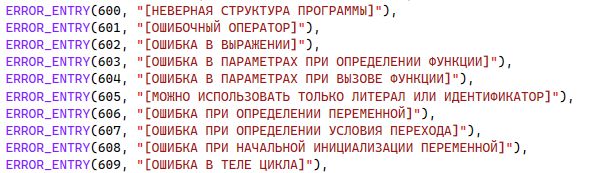


Рисунок 4.2 - Сообщения синтаксического анализатора

На этапе синтаксического анализа определено 10 потенциально возможных ошибок, каждая из которых обрабатывается и в случае возникновения – срабатывает и выводится.

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

Входной информацией для синтаксического анализатора является таблица лексем и идентификаторов. Кроме того, используется описание грамматики в форме Грейбах. Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью выводятся в журнал работы программы.

## **4.8 Принцип обработки ошибок**

Синтаксический анализатор выполняет разбор исходной последовательности лексем до тех пор, пока не дойдёт до конца цепочки лексем или не найдёт ошибку. Тогда анализ останавливается и выводится сообщение об ошибке (если она найдена). Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

## **4.9 Контрольный пример**

Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью приведены в приложении В.

# **5 Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализатор принимает на свой вход результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов и результат работы синтаксического анализатора, то есть дерево разбора, и последовательно ищет необходимые ошибки. Некоторые проверки (такие как проверка на единственность точки входа, проверка на предварительное объявление переменной) осуществляются в процессе лексического анализа. Общая структура обособленно работающего (не параллельно с лексическим анализом) семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1 Структура семантического анализатора

## **5.2 Функции семантического анализатора**

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

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

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.1.

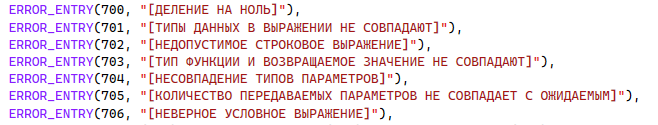


Рисунок 5.2 Перечень сообщений семантического анализатора.

На этапе семантического анализа предусмотрено возникновение 6 возможных ошибок.

## **5.4 Принцип обработки ошибок**

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением.

## **5.5 Контрольный пример**

Соответствие примеров некоторых ошибок в исходном коде и диагностических сообщений об ошибках приведено в таблице 5.1.

Таблица 5.1 - Примеры диагностики ошибок

|  |  |
| --- | --- |
| Исходный код | Текст сообщения |
| main{  short var x;  x = 5 / 0;  } | Ошибка 700: [ДЕЛЕНИЕ НА НОЛЬ] |
| main{  short var x = 15;  x = x + “string”;  } | Ошибка 701: [ТИПЫ ДАННЫХ В ВЫРАЖЕНИИ НЕ СОВПАДАЮТ] |
| main{  str var y;  y = "test2" + “test3”;  } | Ошибка 702: [НЕДОПУСТИМОЕ СТРОКОВОЕ ВЫРАЖЕНИЕ] |

Общая информация про ошибки выводится непосредственно в консоль. Более подробная – в текстовые файлы.

6 Вычисление выражений

## **6.1 Выражения, допускаемые языком**

В языке программирования KAD-2023 поддерживаются только выражения между идентификаторами и литералами целочисленного типа. Приоритет операций представлен на таблице 6.1.

Таблица 6.1 - Приоритеты операций

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| % | 4 |
| / | 4 |
| \* | 4 |
| - | 3 |
| + | 3 |
| , | 2 |
| ( ) | 1 |

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

Все выражения языка KAD-2023 преобразовываются к обратной польской записи.

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Существует два типа польской записи: прямая и обратная, также известные как префиксная и постфиксная. Отличие их от классического, инфиксного способа заключается в том, что знаки операций пишутся не между, а, соответственно, до или после аргументов.

Алгоритм построения польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку;
* операция записывается в стек, если стек пуст;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.

## **6.3 Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений к польской записи представлена в приложении Г.

## **6.4 Контрольный пример**

Пример преобразования выражений из контрольных примеров к обратной польской записи представлен в таблице 6.2.

Таблица 6.2 - Преобразование выражений к ПОЛИЗ

|  |  |
| --- | --- |
| Выражение | Обратная польская запись для выражения |
| i+l | Il+ |
| ((i+l)\*l/l-l)%l | il+l\*l/l-l%#### |
| i(l,l) | ill@2# |

ПОЛИЗ, или, как её еще называют, обратная польская запись, это способ бесско­боч­ного представления выражений (не только арифметических), в которых операнды пред­шествуют операции.

# **7 Генерация кода**

## **7.1 Структура генератора кода**

В языке KAD-2023 генерация кода является заключительным этапом трансляции. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. В соответствии с таблицей лексем строится выходной файл на языке ассемблера, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода KAD-2023 представлена на рисунке 7.1.



Рисунок 7.1 – Структура генератора кода

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

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

Таблица 7.1 – Соответствия типов идентификаторов языка KAD-2023 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке KAD-2023 | Тип идентификатора на языке ассемблера | Пояснение |
| short | sword | Хранит целочисленный 2-ух байтный тип данных. |

Таблица 7.1 – Соответствия типов идентификаторов языка KAD-2023 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке KAD-2023 | Тип идентификатора на языке ассемблера | Пояснение |
| str | dword | Хранит указатель на начало строки. Строка должна завершаться символом нуль-терминатор. |

В то время как идентификатор str имеет тип dword, литерал этой строки хранится как тип данных byte.

## **7.3 Статическая библиотека**

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

Таблица 7.2 – Функции статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| void write\_str(char\* str) | Вывод на консоль строки str |
| void write\_short(short num) | Вывод на консоль целочисленной переменной num |
| short length (char\* str) | Вычисление длины строки |
| char\* copy (char\* str1, char\* str2, short count) | Копирование определенного количества count символов из str2 в строку str1 |
| short random(short start, short end) | Генерирует случайное значение в диапазоне от start до end |
| char\* getLocalTimeAndDate() | Возвращает текущее локальное время и дату |

Объявление функций статической библиотеки генерируется автоматически.

## **7.4 Особенности алгоритма генерации кода**

В языке KAD-2023 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2.



Рисунок 7.2 Структура генератора кода

## **7.5 Входные параметры генератора кода**

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

## **7.6 Контрольный пример**

Результат работы кода, сгенерированного на основе контрольного примера, представлен на рисунке 7.3

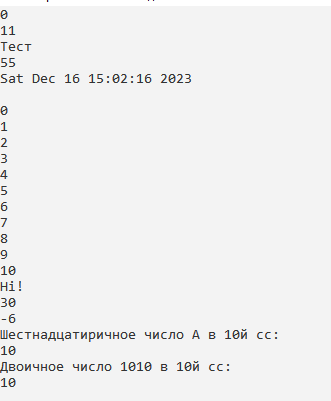


Рисунок 7.3 – Результат работы кода контрольного примера

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Д.

# **8 Тестирование транслятора**

## **8.1 Тестирование проверки на допустимость символов**

В языке KAD-2023 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 { $ } | Ошибка 110: [НЕДОПУСТИМЫЙ СИМВОЛ В ФАЙЛЕ -IN], строка 1, столбец 8 |
| main {  write "test  } | Ошибка 111: [НЕХВАТКА КАВЫЧКИ], строка 3, столбец 16 |

## **8.2 Тестирование лексического анализатора**

На этапе лексического анализа в языке KAD-2023 могут возникнуть ошибки, описанные в пункте 3.7. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 - Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main {  test123  } | Ошибка на этапе лексического анализатора  Ошибка 120: [ЛЕКСЕМА НЕРАСПОЗНАННА], строка 3, лексема 1 |
| main {  var test;  } | Ошибка на этапе лексического анализатора  Ошибка 121: [ТИП ДАННЫХ ИДЕНТИФИКАТОРА НЕ ОПРЕДЕЛЕН], строка 3, лексема 2 |
| main {  short var test;  short var test;  } | Ошибка на этапе лексического анализатора  Ошибка 131: [ПОПЫТКА ПЕРЕОПРЕДЕЛЕНИЯ ПЕРЕМЕННОЙ], строка 4, лексема 3 |
| main {  short test;  } | Ошибка на этапе лексического анализатора  Ошибка 132: [ТИП НЕ ОПРЕДЕЛЕН], строка 3, лексема 2 |

Таблица 8.2 - Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| short function FindMaxLen(str param x, str param y){} | Ошибка 133: [ОТСУТСТВИЕ MAIN] |
|  |  |
| main {}  main {  short var test;  } | Ошибка 134: [ОБНАРУЖЕННО НЕСКОЛЬКО ТОЧЕК ВХОДА MAIN] |

## **8.3 Тестирование синтаксического анализатора**

На этапе синтаксического анализа в языке KAD-2023 могут возникнуть ошибки, описанные в пункте 4.6. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 - Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| function main  {  short var test;  } | 600: строка 1, [НЕВЕРНАЯ СТРУКТУРА ПРОГРАММЫ] |
| main  {  return 1;  } | 601: строка 4, [ОШИБОЧНЫЙ ОПЕРАТОР] |
| main  {  short var test;  test = 1 + - 2;  } | 602: строка 4, [ОШИБКА В ВЫРАЖЕНИИ] |

Таблица 8.3 (продолжение) - Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| short function FindMaxLen(str param x, )  {  }  main {} | 603: строка 1, [ОШИБКА В ПАРАМЕТРАХ ПРИ ОПРЕДЕЛЕНИИ ФУНКЦИИ] |
| short function FindMax(short param x)  { return 1;}  main  {  short var res;  res = FindMax(1, )  } | 604: строка 6, [ОШИБКА В ПАРАМЕТРАХ ПРИ ВЫЗОВЕ ФУНКЦИИ] |
| short function FindMax(short param x)  { return +;}  main  {  short var res;  res = FindMax(1)  } | 605: строка 2, [МОЖНО ИСПОЛЬЗОВАТЬ ТОЛЬКО ЛИТЕРАЛ ИЛИ ИДЕНТИФИКАТОР] |
| main  {  short str var test;  } | 606: строка 3, [ОШИБКА ПРИ ОПРЕДЕЛЕНИИ ПЕРЕМЕННОЙ] |
| main  {  if {1 > 0)  then {}  } | 607: строка 3, [ОШИБКА ПРИ ОПРЕДЕЛЕНИИ УСЛОВИЯ ПЕРЕХОДА] |

Таблица 8.3 (окончание) - Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| main  {  short var a = -;  } | 608: строка 3, [ОШИБКА ПРИ НАЧАЛЬНОЙ ИНИЦИАЛИЗАЦИИ ПЕРЕМЕННОЙ] |
| main{  short var i = 0;  repeat(i < 10){  return 1;  }  } | 609: строка 3, [ОШИБКА В ТЕЛЕ ЦИКЛА] |

## **8.4 Тестирование семантического анализатора**

Итоги тестирования семантического анализатора на корректное обнаружение семантических ошибок приведены в таблице 8.4.

Таблица 8.4 - Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Текст сообщения |
| main{  short var x = 125;  x = x + “test”;  } | Ошибка 701: [ТИПЫ ДАННЫХ В ВЫРАЖЕНИИ НЕ СОВПАДАЮТ] |
| main{  short var x;  x = 5 / 0;  } | Ошибка 700: [ДЕЛЕНИЕ НА НОЛЬ] |
| main{  str var y;  y = "test2" + “test3”;  } | Ошибка 702: [НЕДОПУСТИМОЕ СТРОКОВОЕ ВЫРАЖЕНИЕ] |

Таблица 8.4 - Тестирование семантического анализатора

|  |  |
| --- | --- |
| str function Test(int param x){  return 0;  }  main {} | Ошибка 703: [ТИП ФУНКЦИИ И ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ НЕ СОВПАДАЮТ] |
| short function Test(short param x)  {return 0;}  main {  short var temp;  temp = Test("test");  } | Ошибка 704: [НЕСОВПАДЕНИЕ ТИПОВ ПАРАМЕТРОВ] |
| short function Test(short param x)  {return 0;}  main {  short var temp;  temp = Test(1, 2);  } | Ошибка 705: [КОЛИЧЕСТВО ПЕРЕДАВАЕМЫХ ПАРАМЕТРОВ НЕ СОВПАДАЕТ С ОЖИДАЕМЫМ] |
| main{  repeat(“str1” < “str2”){  }  } | Ошибка 706: [НЕВЕРНОЕ УСЛОВНОЕ ВЫРАЖЕНИЕ] |

Семантический анализ в языке KAD-2023 содержит множество проверок по семантическим правилам, описанным в пункте 1.16.

**Заключение**

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования KAD-2023 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

Сформулирована спецификация языка KAD-2023;

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

Окончательная версия языка KAD-2023 включает:

1. 2 типа данных;
2. Поддержка оператора вывода;
3. Возможность вызова 5 функций стандартной библиотеки;
4. Наличие 5 арифметических операторов для вычисления выражений;
5. Поддержка функций, операторов цикла и условия;
6. Структурированная и классифицированная система для обработки ошибок пользователя.

Проделанная работа позволила получить необходимое представление о структурах и процессах, использующихся при построении трансляторов, а также основные различия и преимущества тех или иных средств трансляции.

# **Список использованных источников**

1. Вирт Н. Построение компиляторов/ Пер. с англ. Борисов Е. В., Чернышов Л. Н. – М.: ДМК Пресс, 2010. – 192 с.: ил.

2. Грис Д. Конструирование компиляторов для цифровых вычислительных машин.: Пер. с англ. — М.: Мир, 1975.

3. Костельцев А. В. Построение интерпретаторов и компиляторов. СПб: Наука и Техника, 2001. — 224 стр. с ил.

4. Пратт Т. Языки программирования: разработка и реализация. Пер. с англ. — М.: Мир, 1979.

5. Хантер Р. Проектирование и конструирование компиляторов/ Пер. с англ.: Предисл. В. М. Савинкова. — М.: Финансы и статистика, 1984. — 232 с., ил.

6. Хендрикс Д. Компилятор языка Си для микроЭВМ: Пер. с англ. — М.: радио и связь, 1989. — 240 с.: ил.

# **Приложение А**

Листинг 1. Код программы на языке KAD-2023

short function sum(short param first, short param second, short param third){

short var sumRes = first + second + third;

return sumRes;

}

short function differenceInLength(str param strokeOne, str param strokeTwo){

short var lenOne = lenght(strokeOne);

short var lenTwo = lenght(strokeTwo);

short var res = lenOne - lenTwo;

return res;

}

str function getHi(){

str var hi = "Hi!";

write hi;

return hi;

}

main{

#Пример работы арифметических операций

short var a = -3;

a = ((a + 7) \* 6 / 3 - 4) % 2;

write a;

#Пример работы функции вычисления длины строки

str var strB = "Тест строки";

short var b = lenght(strB);

write b;

#Пример функции копирования строки

str var strBCopy = copy(strBCopy, strB, 4);

write strBCopy;

#Функция получения случайного числа

short var rand = random(0, 100);

write rand;

#Функция получения локальной даты

str var date = getLocalTimeAndDate();

write date;

#Пример работы цикла

short var i = 0;

repeat(i<10){

write i;

i = i + 1;

}

str var h = getHi();

Листинг 1 (продолжение). Код программы на языке KAD-2023

#Вызов функции и передача целочисленных параметров

a = 10;

b = 15;

short var c = 5;

short var sumABC = sum(a, b, c);

write sumABC;

#Вызов функции и передача строковых параметров

str var firstStroke = "Hello";

str var secondStroke = "How are you";

short var difBetween = differenceInLength(firstStroke, secondStroke);

write difBetween;

#Шестнадцатиричное представление числа

short var hexNum = hA;

write "Шестнадцатиричное число A в 10й сс:";

write hexNum;

#Двоичное прелставление числа

short var binNum = b1010;

write "Двоичное число 1010 в 10й сс:";

write binNum;

}

# **Приложение Б**

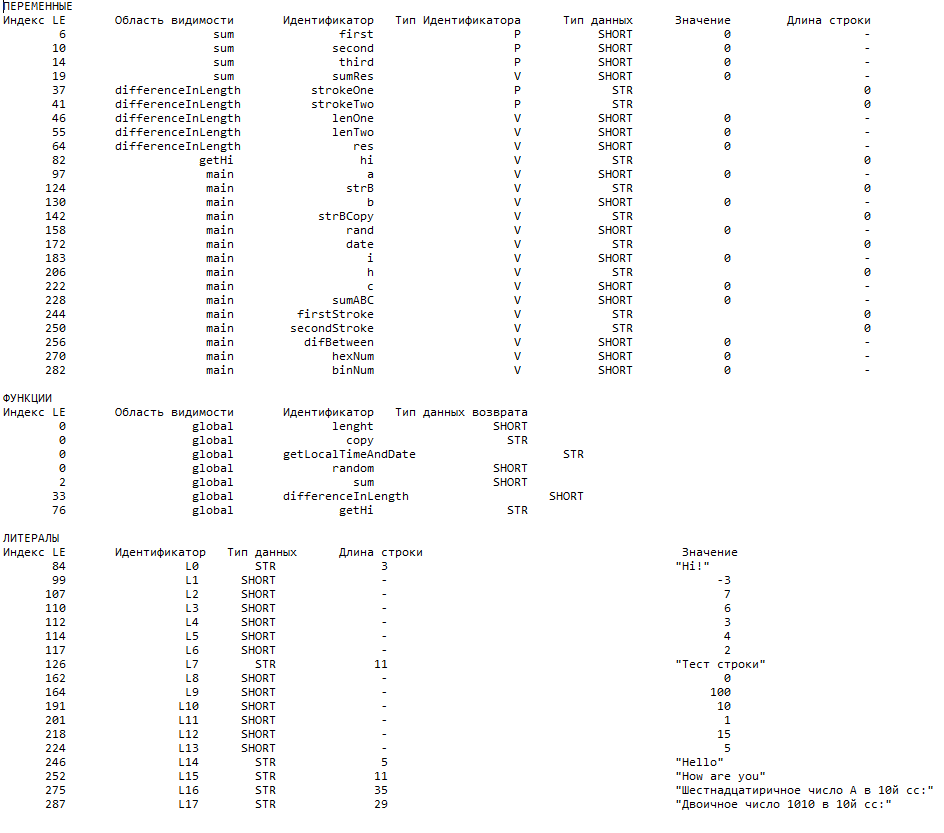
****

Рисунок 1. - Таблица идентификаторов после контрольного примера

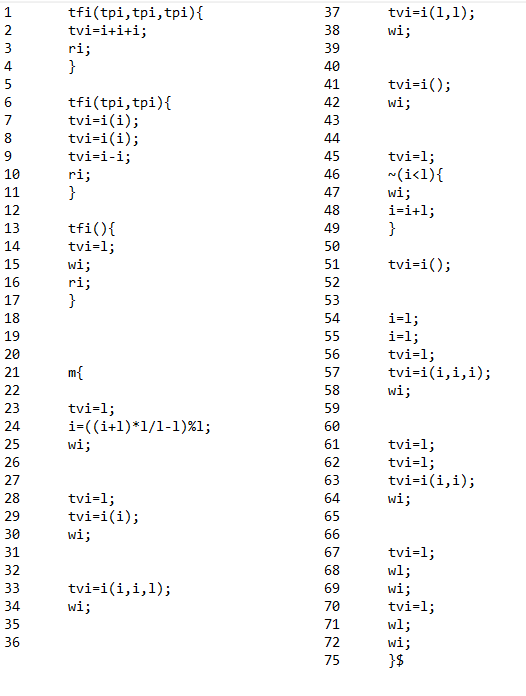
****

Рисунок 2. - Таблица лексем после контрольного примера

# **Приложение В**

Листинг 1. Грамматика языка KAD-2022

|  |
| --- |
| Greibach greibach(NS('S'), TS('$'), 12,  Rule(NS('S'), GRB\_ERROR\_SERIES + 0, //Неверная структура программы  3, // m{N}S | tfi(F){NrU;}S | пустой переход  Rule::Chain(5, TS('m'), TS('{'), NS('N'), TS('}'), NS('S')),  Rule::Chain(13, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('U'), TS(';'), TS('}'), NS('S')),  Rule::Chain()  ),  Rule(NS('N'), GRB\_ERROR\_SERIES + 1, //ошибочный оператор  5, // tY;N | i=E;N | wU;N | ~K{N}N | ?KJN | пустой переход  Rule::Chain(4, TS('t'), NS('Y'), TS(';'), NS('N')),  Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(4, TS('w'), NS('E'), TS(';'), NS('N')),  Rule::Chain(4, TS('~'), NS('K'), NS('Z'), NS('N')),  Rule::Chain()  ),  Rule(NS('E'), GRB\_ERROR\_SERIES + 2, // Ошибка в выражении  5, // iM | lM | (E)M | i(W)M  Rule::Chain(2, TS('i'), NS('M')),  Rule::Chain(2, TS('l'), NS('M')),  Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M')),  Rule::Chain(4, TS('i'), TS('('), TS(')'), NS('M'))  ),  Rule(NS('M'), GRB\_ERROR\_SERIES + 2, // Ошибка в выражении  6, // #E | пустой переход  Rule::Chain(2, TS('+'), NS('E')),  Rule::Chain(2, TS('-'), NS('E')),  Rule::Chain(2, TS('\*'), NS('E')),  Rule::Chain(2, TS('/'), NS('E')),  Rule::Chain(2, TS('%'), NS('E')),  Rule::Chain()  ),  Rule(NS('F'), GRB\_ERROR\_SERIES + 3, // Ошибка в параметрах при определении функции  3, // tP | tP,F  Rule::Chain(2, TS('t'), NS('P')),  Rule::Chain(4, TS('t'), NS('P'), TS(','), NS('F')),  Rule::Chain()  ),  Rule(NS('P'), GRB\_ERROR\_SERIES + 3, // Ошибка в параметрах при определении функции  1, // pi  Rule::Chain(2, TS('p'), TS('i'))  ), |

Листинг 1 (продолжение). Грамматика языка KAD-2022

|  |
| --- |
| Rule(NS('W'), GRB\_ERROR\_SERIES + 4, // Ошибка в параметрах при вызове функции  4, // i | l | i,W | l,W  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('i'), TS(','), NS('W')),  Rule::Chain(3, TS('l'), TS(','), NS('W'))  ),  Rule(NS('U'), GRB\_ERROR\_SERIES + 5, // Только литерал или идентификатор  6, // l | i  Rule::Chain(1, TS('l')),  Rule::Chain(1, TS('i')),  Rule::Chain(2, TS('l'), NS('U')),  Rule::Chain(2, TS('i'), NS('U')),  Rule::Chain(3, TS('i'), TS(','), NS('U')),  Rule::Chain(3, TS('l'), TS(','), NS('U'))  ),  Rule(NS('Y'), GRB\_ERROR\_SERIES + 6, // Ошибка при определении переменной  2, // vi || vi = L  Rule::Chain(2, TS('v'), TS('i')),  Rule::Chain(4, TS('v'), TS('i'), TS('='), NS('L'))  ),  Rule(NS('K'), GRB\_ERROR\_SERIES + 7, // Ошибка при определении условия перехода  2, // (U<U) | (U>U)  Rule::Chain(5, TS('('), NS('U'), TS('<'), NS('U'), TS(')')),  Rule::Chain(5, TS('('), NS('U'), TS('>'), NS('U'), TS(')'))  ),  Rule(NS('L'), GRB\_ERROR\_SERIES + 8, // Ошибка при начальной инициализации  25, //vi = l || vi = i || vi = i(i)  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(4, TS('i'), TS('('), NS('U'), TS(')')),  Rule::Chain(3, TS('i'), TS('('), TS(')')),  Rule::Chain(2, TS('i'), NS('L')),  Rule::Chain(2, TS('l'), NS('L')),  Rule::Chain(2, TS('+'), TS('i')),  Rule::Chain(2, TS('+'), TS('l')),  Rule::Chain(3, TS('+'), TS('i'), NS('L')),  Rule::Chain(3, TS('+'), TS('l'), NS('L')),  Rule::Chain(2, TS('-'), TS('i')),  Rule::Chain(2, TS('-'), TS('l')), |

Листинг 1 (окончание). Грамматика языка KAD-2022

|  |
| --- |
| Rule::Chain(3, TS('-'), TS('i'), NS('L')),  Rule::Chain(3, TS('-'), TS('l'), NS('L')),  Rule::Chain(2, TS('\*'), TS('i')),  Rule::Chain(2, TS('\*'), TS('l')),  Rule::Chain(3, TS('\*'), TS('i'), NS('L')),  Rule::Chain(3, TS('\*'), TS('l'), NS('L')),  Rule::Chain(2, TS('/'), TS('i')),  Rule::Chain(2, TS('/'), TS('l')),  Rule::Chain(3, TS('/'), TS('i'), NS('L')),  Rule::Chain(3, TS('/'), TS('l'), NS('L')),  Rule::Chain(2, TS('%'), TS('i')),  Rule::Chain(2, TS('%'), TS('l')),  Rule::Chain(3, TS('%'), TS('i'), NS('L')),  Rule::Chain(3, TS('%'), TS('l'), NS('L'))  ),  Rule(NS('Z'), GRB\_ERROR\_SERIES + 9, //ошибка в теле цикла  1,  Rule::Chain(3, TS('{'), NS('N'), TS('}'))  ) |

Листинг 2. Структура магазинного автомата

|  |
| --- |
| struct MFSTState // состояние автомата (для сохранения)  {  short posInLent; // позиция на ленте  short nRule; // номер текущего правила  short nRuleChain; // номер текущей цепочки, текущего правила  MFSTSTSTACK st; // стек автомата  MFSTState();  MFSTState(short posInLent, // позиция на ленте  MFSTSTSTACK a\_steck, // стек автомата  short currentChain); // номер текущей цепочки, текущего правила  MFSTState(short posInLent, // позиция на ленте  MFSTSTSTACK a\_stack, // стек автомата  short currentRule, // номер текущего правила  short currentChain); // номер текущей цепочки, текущего правила  };  struct MFST // магазинный автомат  {  enum RC\_STEP //код возврата функции step  {  NS\_OK, // найдено правило и цепочка, цепочка записана в стек  NS\_NORULE, // не найдено правило грамматики (ошибка в грамматике)  NS\_NORULECHAIN, // не найдена походящая цепочка правила (ошибка в исходном коде)  NS\_ERROR, // неизвесный нетерминальный символ грамматики  TS\_OK, // тек. символ ленты == вершине стека, продвинулась лента, pop стека  TS\_NOK, // тек. символ ленты != вершине стека, восстановленно состояние  LENTA\_END, // теущая позиция ленты >= lenta\_size  SURPRISE // неожиданный код возврата (ошибка в step)  };  struct MFST\_Diagnosis // диагностика  {  short posInLent; // позиция на ленте  RC\_STEP rc\_step; // код завершения шага  short ruleNum; // номер правила  short nrule\_chain; // номер цепочки правила  MFST\_Diagnosis();  MFST\_Diagnosis(short posInLent, RC\_STEP rc\_step, short ruleNum, short ruleChainNum);  } diagnosis[MFST\_DIAGN\_NUMBER]; // последние самые глубокие сообщения  GRBALPHABET\* lenta; // перекодированная (TS/NS) лента (из LEX)  short currentPosInLent; // текущая позиция на ленте  short currentRule; // номер текущего правила |

Листинг 2 (продолжение). Структура магазинного автомата

|  |
| --- |
| short currentRuleChain; // номер текущей цепочки, текущего правила  short lenta\_size; // размер ленты  GRB::Greibach grebach; // грамматика Грейбах  LT::LexTable lexTable;  MFSTSTSTACK st; // стек автомата  std::stack<MFSTState> storestate; // стек для сохранения состояний  MFST();  MFST(const LT::LexTable& lexTable, GRB::Greibach grebach);  char\* getCSt(char\* buf); //получить содержимое стека  char\* getCLenta(char\* buf, short pos, short n = 25); //лента: n символов, начиная с pos  char\* getDiagnosis(short n, char\* buf); //получить n-ую строку диагностики или '\0'  bool savestate(std::ostream\* stream); //сохранить состояние автомата  bool resetstate(std::ostream\* stream); //восстановить состояние автомата  bool push\_chain(GRB::Rule::Chain chain);  RC\_STEP step(std::ostream\* stream); //выполнить шаг автомата  bool start(std::ostream\* stream); //запустить автомат  bool savedDiagnosis(RC\_STEP prc\_step);  void printRules(std::ostream\* stream); //вывести последовательность правил  struct Deducation // вывод  {  short stepsCount; // количество шагов в выводе  short\* nRules; // номер правила грамматики  short\* nChainsOfRules; // номер цепочек правил грамматики  Deducation()  {  this->stepsCount = 0;  this->nRules = 0;  this->nChainsOfRules = 0;  }  }deducation;  bool saveoutputTree(); // сохранить дерево вывода  }; |

Листинг 3. Структура грамматики Грейбаха

|  |
| --- |
| struct Greibach // грамматика Грейбах  {  short size; // количество правил  GRBALPHABET startN; // стартовый символ  GRBALPHABET stbottomT; // дно стека  Rule\* rules; // множество правил  Greibach()  {  size = 0;  startN = 0;  stbottomT = 0;  rules = 0;  };  Greibach(GRBALPHABET pstartN, // стартовый символ  GRBALPHABET pstbottomT, // дно стека  short psize, // количество правил  Rule r, ...); // правила  short getRule( // получить правило, возвращается номер правила или -1  GRBALPHABET pnn, // левый символ правила  Rule& prule); // возвращаемое правило грамматики  Rule getRule(short n); // получить правило по номеру  }; |

Листинг 4. Разбор исходного кода синтаксическим анализатором

|  |
| --- |
| 0 :S->tfi(F){NrU;}S tfi(tpi,tpi,tpi){tvi=i+i+ S$  1 : SAVESTATE: 1  1 : tfi(tpi,tpi,tpi){tvi=i+i+ tfi(F){NrU;}S$  2 : fi(tpi,tpi,tpi){tvi=i+i+i fi(F){NrU;}S$  3 : i(tpi,tpi,tpi){tvi=i+i+i; i(F){NrU;}S$  4 : (tpi,tpi,tpi){tvi=i+i+i;r (F){NrU;}S$  5 : tpi,tpi,tpi){tvi=i+i+i;ri F){NrU;}S$  6 :F->tP tpi,tpi,tpi){tvi=i+i+i;ri F){NrU;}S$  7 : SAVESTATE: 2  7 : tpi,tpi,tpi){tvi=i+i+i;ri tP){NrU;}S$  8 : pi,tpi,tpi){tvi=i+i+i;ri; P){NrU;}S$  9 :P->pi pi,tpi,tpi){tvi=i+i+i;ri; P){NrU;}S$  10 : SAVESTATE: 3  10 : pi,tpi,tpi){tvi=i+i+i;ri; pi){NrU;}S$  11 : i,tpi,tpi){tvi=i+i+i;ri;} i){NrU;}S$  12 : ,tpi,tpi){tvi=i+i+i;ri;}m ){NrU;}S$  13 : TNS\_NS\_NORULECHAIN/NS\_NORULE  13 : RESTATE  13 : pi,tpi,tpi){tvi=i+i+i;ri; P){NrU;}S$  14 : TNS\_NS\_NORULECHAIN/NS\_NORULE  14 : RESTATE  14 : tpi,tpi,tpi){tvi=i+i+i;ri F){NrU;}S$  15 :F->tP,F tpi,tpi,tpi){tvi=i+i+i;ri F){NrU;}S$  16 : SAVESTATE: 2  16 : tpi,tpi,tpi){tvi=i+i+i;ri tP,F){NrU;}S$  17 : pi,tpi,tpi){tvi=i+i+i;ri; P,F){NrU;}S$  18 :P->pi pi,tpi,tpi){tvi=i+i+i;ri; P,F){NrU;}S$  19 : SAVESTATE: 3  19 : pi,tpi,tpi){tvi=i+i+i;ri; pi,F){NrU;}S$  20 : i,tpi,tpi){tvi=i+i+i;ri;} i,F){NrU;}S$  21 : ,tpi,tpi){tvi=i+i+i;ri;}m ,F){NrU;}S$  22 : tpi,tpi){tvi=i+i+i;ri;}m{ F){NrU;}S$  23 :F->tP tpi,tpi){tvi=i+i+i;ri;}m{ F){NrU;}S$  24 : SAVESTATE: 4  24 : tpi,tpi){tvi=i+i+i;ri;}m{ tP){NrU;}S$  25 : pi,tpi){tvi=i+i+i;ri;}m{t P){NrU;}S$  26 :P->pi pi,tpi){tvi=i+i+i;ri;}m{t P){NrU;}S$  27 : SAVESTATE: 5  27 : pi,tpi){tvi=i+i+i;ri;}m{t pi){NrU;}S$  28 : i,tpi){tvi=i+i+i;ri;}m{tv i){NrU;}S$  29 : ,tpi){tvi=i+i+i;ri;}m{tvi ){NrU;}S$  30 : TNS\_NS\_NORULECHAIN/NS\_NORULE  30 : RESTATE  30 : pi,tpi){tvi=i+i+i;ri;}m{t P){NrU;}S$  31 : TNS\_NS\_NORULECHAIN/NS\_NORULE  31 : RESTATE  31 : tpi,tpi){tvi=i+i+i;ri;}m{ F){NrU;}S$  32 :F->tP,F tpi,tpi){tvi=i+i+i;ri;}m{ F){NrU;}S$  33 : SAVESTATE: 4  33 : tpi,tpi){tvi=i+i+i;ri;}m{ tP,F){NrU;}S$  34 : pi,tpi){tvi=i+i+i;ri;}m{t P,F){NrU;}S$  35 :P->pi pi,tpi){tvi=i+i+i;ri;}m{t P,F){NrU;}S$  36 : SAVESTATE: 5  36 : pi,tpi){tvi=i+i+i;ri;}m{t pi,F){NrU;}S$  37 : i,tpi){tvi=i+i+i;ri;}m{tv i,F){NrU;}S$  38 : ,tpi){tvi=i+i+i;ri;}m{tvi ,F){NrU;}S$  39 : tpi){tvi=i+i+i;ri;}m{tvi= F){NrU;}S$  40 :F->tP tpi){tvi=i+i+i;ri;}m{tvi= F){NrU;}S$  41 : SAVESTATE: 6  41 : tpi){tvi=i+i+i;ri;}m{tvi= tP){NrU;}S$  42 : pi){tvi=i+i+i;ri;}m{tvi=l P){NrU;}S$  43 :P->pi pi){tvi=i+i+i;ri;}m{tvi=l P){NrU;}S$  44 : SAVESTATE: 7  44 : pi){tvi=i+i+i;ri;}m{tvi=l pi){NrU;}S$  45 : i){tvi=i+i+i;ri;}m{tvi=l; i){NrU;}S$  46 : ){tvi=i+i+i;ri;}m{tvi=l;i ){NrU;}S$  47 : {tvi=i+i+i;ri;}m{tvi=l;i= {NrU;}S$  48 : tvi=i+i+i;ri;}m{tvi=l;i=( NrU;}S$  49 :N->tY;N tvi=i+i+i;ri;}m{tvi=l;i=( NrU;}S$  50 : SAVESTATE: 8  50 : tvi=i+i+i;ri;}m{tvi=l;i=( tY;NrU;}S$  51 : vi=i+i+i;ri;}m{tvi=l;i=(( Y;NrU;}S$  52 :Y->vi vi=i+i+i;ri;}m{tvi=l;i=(( Y;NrU;}S$  53 : SAVESTATE: 9  53 : vi=i+i+i;ri;}m{tvi=l;i=(( vi;NrU;}S$  54 : i=i+i+i;ri;}m{tvi=l;i=((i i;NrU;}S$  55 : =i+i+i;ri;}m{tvi=l;i=((i+ ;NrU;}S$  56 : TNS\_NS\_NORULECHAIN/NS\_NORULE  56 : RESTATE  56 : vi=i+i+i;ri;}m{tvi=l;i=(( Y;NrU;}S$  57 :Y->vi=L vi=i+i+i;ri;}m{tvi=l;i=(( Y;NrU;}S$  58 : SAVESTATE: 9  58 : vi=i+i+i;ri;}m{tvi=l;i=(( vi=L;NrU;}S$  59 : i=i+i+i;ri;}m{tvi=l;i=((i i=L;NrU;}S$  60 : =i+i+i;ri;}m{tvi=l;i=((i+ =L;NrU;}S$  61 : i+i+i;ri;}m{tvi=l;i=((i+l L;NrU;}S$  62 :L->i i+i+i;ri;}m{tvi=l;i=((i+l L;NrU;}S$  63 : SAVESTATE: 10  63 : i+i+i;ri;}m{tvi=l;i=((i+l i;NrU;}S$  64 : +i+i;ri;}m{tvi=l;i=((i+l) ;NrU;}S$  65 : TNS\_NS\_NORULECHAIN/NS\_NORULE  65 : RESTATE  65 : i+i+i;ri;}m{tvi=l;i=((i+l L;NrU;}S$  66 :L->i(U) i+i+i;ri;}m{tvi=l;i=((i+l L;NrU;}S$  67 : SAVESTATE: 10  67 : i+i+i;ri;}m{tvi=l;i=((i+l i(U);NrU;}S$  68 : +i+i;ri;}m{tvi=l;i=((i+l) (U);NrU;}S$  69 : TNS\_NS\_NORULECHAIN/NS\_NORULE  69 : RESTATE  69 : i+i+i;ri;}m{tvi=l;i=((i+l L;NrU;}S$  70 :L->i() i+i+i;ri;}m{tvi=l;i=((i+l L;NrU;}S$  71 : SAVESTATE: 10  71 : i+i+i;ri;}m{tvi=l;i=((i+l i();NrU;}S$  72 : +i+i;ri;}m{tvi=l;i=((i+l) ();NrU;}S$  73 : TNS\_NS\_NORULECHAIN/NS\_NORULE  73 : RESTATE  73 : i+i+i;ri;}m{tvi=l;i=((i+l L;NrU;}S$  74 :L->iL i+i+i;ri;}m{tvi=l;i=((i+l L;NrU;}S$  75 : SAVESTATE: 10  75 : i+i+i;ri;}m{tvi=l;i=((i+l iL;NrU;}S$  76 : +i+i;ri;}m{tvi=l;i=((i+l) L;NrU;}S$  77 :L->+i +i+i;ri;}m{tvi=l;i=((i+l) L;NrU;}S$  78 : SAVESTATE: 11  78 : +i+i;ri;}m{tvi=l;i=((i+l) +i;NrU;}S$  79 : i+i;ri;}m{tvi=l;i=((i+l)\* i;NrU;}S$  80 : +i;ri;}m{tvi=l;i=((i+l)\*l ;NrU;}S$  81 : TNS\_NS\_NORULECHAIN/NS\_NORULE  81 : RESTATE  81 : +i+i;ri;}m{tvi=l;i=((i+l) L;NrU;}S$  82 :L->+l +i+i;ri;}m{tvi=l;i=((i+l) L;NrU;}S$  83 : SAVESTATE: 11  83 : +i+i;ri;}m{tvi=l;i=((i+l) +l;NrU;}S$  84 : i+i;ri;}m{tvi=l;i=((i+l)\* l;NrU;}S$  85 : TNS\_NS\_NORULECHAIN/NS\_NORULE  85 : RESTATE  85 : +i+i;ri;}m{tvi=l;i=((i+l) L;NrU;}S$  86 :L->+iL +i+i;ri;}m{tvi=l;i=((i+l) L;NrU;}S$  87 : SAVESTATE: 11  87 : +i+i;ri;}m{tvi=l;i=((i+l) +iL;NrU;}S$  88 : i+i;ri;}m{tvi=l;i=((i+l)\* iL;NrU;}S$  89 : +i;ri;}m{tvi=l;i=((i+l)\*l L;NrU;}S$  90 :L->+i +i;ri;}m{tvi=l;i=((i+l)\*l L;NrU;}S$  91 : SAVESTATE: 12  91 : +i;ri;}m{tvi=l;i=((i+l)\*l +i;NrU;}S$  92 : i;ri;}m{tvi=l;i=((i+l)\*l/ i;NrU;}S$  93 : ;ri;}m{tvi=l;i=((i+l)\*l/l ;NrU;}S$  94 : ri;}m{tvi=l;i=((i+l)\*l/l- NrU;}S$  95 :N-> ri;}m{tvi=l;i=((i+l)\*l/l- NrU;}S$  96 : SAVESTATE: 13  96 : ri;}m{tvi=l;i=((i+l)\*l/l- rU;}S$  97 : i;}m{tvi=l;i=((i+l)\*l/l-l U;}S$  98 :U->i i;}m{tvi=l;i=((i+l)\*l/l-l U;}S$  99 : SAVESTATE: 14  99 : i;}m{tvi=l;i=((i+l)\*l/l-l i;}S$  100 : ;}m{tvi=l;i=((i+l)\*l/l-l) ;}S$  101 : }m{tvi=l;i=((i+l)\*l/l-l)% }S$  102 : m{tvi=l;i=((i+l)\*l/l-l)%l S$  103 :S->m{N}S m{tvi=l;i=((i+l)\*l/l-l)%l S$  104 : SAVESTATE: 15  104 : m{tvi=l;i=((i+l)\*l/l-l)%l m{N}S$  105 : {tvi=l;i=((i+l)\*l/l-l)%l; {N}S$  106 : tvi=l;i=((i+l)\*l/l-l)%l;w N}S$  107 :N->tY;N tvi=l;i=((i+l)\*l/l-l)%l;w N}S$  108 : SAVESTATE: 16  108 : tvi=l;i=((i+l)\*l/l-l)%l;w tY;N}S$  109 : vi=l;i=((i+l)\*l/l-l)%l;wi Y;N}S$  110 :Y->vi vi=l;i=((i+l)\*l/l-l)%l;wi Y;N}S$  111 : SAVESTATE: 17  111 : vi=l;i=((i+l)\*l/l-l)%l;wi vi;N}S$  112 : i=l;i=((i+l)\*l/l-l)%l;wi; i;N}S$  113 : =l;i=((i+l)\*l/l-l)%l;wi;t ;N}S$  114 : TNS\_NS\_NORULECHAIN/NS\_NORULE  114 : RESTATE  114 : vi=l;i=((i+l)\*l/l-l)%l;wi Y;N}S$  115 :Y->vi=L vi=l;i=((i+l)\*l/l-l)%l;wi Y;N}S$  116 : SAVESTATE: 17  116 : vi=l;i=((i+l)\*l/l-l)%l;wi vi=L;N}S$  117 : i=l;i=((i+l)\*l/l-l)%l;wi; i=L;N}S$  118 : =l;i=((i+l)\*l/l-l)%l;wi;t =L;N}S$  119 : l;i=((i+l)\*l/l-l)%l;wi;tv L;N}S$  120 :L->l l;i=((i+l)\*l/l-l)%l;wi;tv L;N}S$  121 : SAVESTATE: 18  121 : l;i=((i+l)\*l/l-l)%l;wi;tv l;N}S$  122 : ;i=((i+l)\*l/l-l)%l;wi;tvi ;N}S$  123 : i=((i+l)\*l/l-l)%l;wi;tvi= N}S$  124 :N->i=E;N i=((i+l)\*l/l-l)%l;wi;tvi= N}S$  125 : SAVESTATE: 19  125 : i=((i+l)\*l/l-l)%l;wi;tvi= i=E;N}S$  126 : =((i+l)\*l/l-l)%l;wi;tvi=l =E;N}S$  127 : ((i+l)\*l/l-l)%l;wi;tvi=l; E;N}S$  128 :E->(E)M ((i+l)\*l/l-l)%l;wi;tvi=l; E;N}S$  129 : SAVESTATE: 20  129 : ((i+l)\*l/l-l)%l;wi;tvi=l; (E)M;N}S$  130 : (i+l)\*l/l-l)%l;wi;tvi=l;t E)M;N}S$  131 :E->(E)M (i+l)\*l/l-l)%l;wi;tvi=l;t E)M;N}S$  132 : SAVESTATE: 21  132 : (i+l)\*l/l-l)%l;wi;tvi=l;t (E)M)M;N}S$  133 : i+l)\*l/l-l)%l;wi;tvi=l;tv E)M)M;N}S$  134 :E->iM i+l)\*l/l-l)%l;wi;tvi=l;tv E)M)M;N}S$  135 : SAVESTATE: 22  135 : i+l)\*l/l-l)%l;wi;tvi=l;tv iM)M)M;N}S$  136 : +l)\*l/l-l)%l;wi;tvi=l;tvi M)M)M;N}S$  137 :M->+E +l)\*l/l-l)%l;wi;tvi=l;tvi M)M)M;N}S$  138 : SAVESTATE: 23  138 : +l)\*l/l-l)%l;wi;tvi=l;tvi +E)M)M;N}S$  139 : l)\*l/l-l)%l;wi;tvi=l;tvi= E)M)M;N}S$  140 :E->lM l)\*l/l-l)%l;wi;tvi=l;tvi= E)M)M;N}S$  141 : SAVESTATE: 24  141 : l)\*l/l-l)%l;wi;tvi=l;tvi= lM)M)M;N}S$  142 : )\*l/l-l)%l;wi;tvi=l;tvi=i M)M)M;N}S$  143 :M-> )\*l/l-l)%l;wi;tvi=l;tvi=i M)M)M;N}S$  144 : SAVESTATE: 25  144 : )\*l/l-l)%l;wi;tvi=l;tvi=i )M)M;N}S$  145 : \*l/l-l)%l;wi;tvi=l;tvi=i( M)M;N}S$  146 :M->\*E \*l/l-l)%l;wi;tvi=l;tvi=i( M)M;N}S$  147 : SAVESTATE: 26  147 : \*l/l-l)%l;wi;tvi=l;tvi=i( \*E)M;N}S$  148 : l/l-l)%l;wi;tvi=l;tvi=i(i E)M;N}S$  149 :E->lM l/l-l)%l;wi;tvi=l;tvi=i(i E)M;N}S$  150 : SAVESTATE: 27  150 : l/l-l)%l;wi;tvi=l;tvi=i(i lM)M;N}S$  151 : /l-l)%l;wi;tvi=l;tvi=i(i) M)M;N}S$  152 :M->/E /l-l)%l;wi;tvi=l;tvi=i(i) M)M;N}S$  153 : SAVESTATE: 28  153 : /l-l)%l;wi;tvi=l;tvi=i(i) /E)M;N}S$  154 : l-l)%l;wi;tvi=l;tvi=i(i); E)M;N}S$  155 :E->lM l-l)%l;wi;tvi=l;tvi=i(i); E)M;N}S$  156 : SAVESTATE: 29  156 : l-l)%l;wi;tvi=l;tvi=i(i); lM)M;N}S$  157 : -l)%l;wi;tvi=l;tvi=i(i);w M)M;N}S$  158 :M->-E -l)%l;wi;tvi=l;tvi=i(i);w M)M;N}S$  159 : SAVESTATE: 30  159 : -l)%l;wi;tvi=l;tvi=i(i);w -E)M;N}S$  160 : l)%l;wi;tvi=l;tvi=i(i);wi E)M;N}S$  161 :E->lM l)%l;wi;tvi=l;tvi=i(i);wi E)M;N}S$  162 : SAVESTATE: 31  162 : l)%l;wi;tvi=l;tvi=i(i);wi lM)M;N}S$  163 : )%l;wi;tvi=l;tvi=i(i);wi; M)M;N}S$  164 :M-> )%l;wi;tvi=l;tvi=i(i);wi; M)M;N}S$  165 : SAVESTATE: 32  165 : )%l;wi;tvi=l;tvi=i(i);wi; )M;N}S$  166 : %l;wi;tvi=l;tvi=i(i);wi;t M;N}S$  167 :M->%E %l;wi;tvi=l;tvi=i(i);wi;t M;N}S$  168 : SAVESTATE: 33  168 : %l;wi;tvi=l;tvi=i(i);wi;t %E;N}S$  169 : l;wi;tvi=l;tvi=i(i);wi;tv E;N}S$  170 :E->lM l;wi;tvi=l;tvi=i(i);wi;tv E;N}S$  171 : SAVESTATE: 34  171 : l;wi;tvi=l;tvi=i(i);wi;tv lM;N}S$  172 : ;wi;tvi=l;tvi=i(i);wi;tvi M;N}S$  173 :M-> ;wi;tvi=l;tvi=i(i);wi;tvi M;N}S$  174 : SAVESTATE: 35  174 : ;wi;tvi=l;tvi=i(i);wi;tvi ;N}S$  175 : wi;tvi=l;tvi=i(i);wi;tvi= N}S$  176 :N->wE;N wi;tvi=l;tvi=i(i);wi;tvi= N}S$  177 : SAVESTATE: 36  177 : wi;tvi=l;tvi=i(i);wi;tvi= wE;N}S$  178 : i;tvi=l;tvi=i(i);wi;tvi=i E;N}S$  179 :E->iM i;tvi=l;tvi=i(i);wi;tvi=i E;N}S$  180 : SAVESTATE: 37  180 : i;tvi=l;tvi=i(i);wi;tvi=i iM;N}S$  181 : ;tvi=l;tvi=i(i);wi;tvi=i( M;N}S$  182 :M-> ;tvi=l;tvi=i(i);wi;tvi=i( M;N}S$  183 : SAVESTATE: 38  183 : ;tvi=l;tvi=i(i);wi;tvi=i( ;N}S$  184 : tvi=l;tvi=i(i);wi;tvi=i(i N}S$  185 :N->tY;N tvi=l;tvi=i(i);wi;tvi=i(i N}S$  186 : SAVESTATE: 39  186 : tvi=l;tvi=i(i);wi;tvi=i(i tY;N}S$  187 : vi=l;tvi=i(i);wi;tvi=i(i, Y;N}S$  188 :Y->vi vi=l;tvi=i(i);wi;tvi=i(i, Y;N}S$  189 : SAVESTATE: 40  189 : vi=l;tvi=i(i);wi;tvi=i(i, vi;N}S$  190 : i=l;tvi=i(i);wi;tvi=i(i,i i;N}S$  191 : =l;tvi=i(i);wi;tvi=i(i,i, ;N}S$  192 : TNS\_NS\_NORULECHAIN/NS\_NORULE  192 : RESTATE  192 : vi=l;tvi=i(i);wi;tvi=i(i, Y;N}S$  193 :Y->vi=L vi=l;tvi=i(i);wi;tvi=i(i, Y;N}S$  194 : SAVESTATE: 40  194 : vi=l;tvi=i(i);wi;tvi=i(i, vi=L;N}S$  195 : i=l;tvi=i(i);wi;tvi=i(i,i i=L;N}S$  196 : =l;tvi=i(i);wi;tvi=i(i,i, =L;N}S$  197 : l;tvi=i(i);wi;tvi=i(i,i,l L;N}S$  198 :L->l l;tvi=i(i);wi;tvi=i(i,i,l L;N}S$  199 : SAVESTATE: 41  199 : l;tvi=i(i);wi;tvi=i(i,i,l l;N}S$  200 : ;tvi=i(i);wi;tvi=i(i,i,l) ;N}S$  201 : tvi=i(i);wi;tvi=i(i,i,l); N}S$  202 :N->tY;N tvi=i(i);wi;tvi=i(i,i,l); N}S$  203 : SAVESTATE: 42  203 : tvi=i(i);wi;tvi=i(i,i,l); tY;N}S$  204 : vi=i(i);wi;tvi=i(i,i,l);w Y;N}S$  205 :Y->vi vi=i(i);wi;tvi=i(i,i,l);w Y;N}S$  206 : SAVESTATE: 43  206 : vi=i(i);wi;tvi=i(i,i,l);w vi;N}S$  207 : i=i(i);wi;tvi=i(i,i,l);wi i;N}S$  208 : =i(i);wi;tvi=i(i,i,l);wi; ;N}S$  209 : TNS\_NS\_NORULECHAIN/NS\_NORULE  209 : RESTATE  209 : vi=i(i);wi;tvi=i(i,i,l);w Y;N}S$  210 :Y->vi=L vi=i(i);wi;tvi=i(i,i,l);w Y;N}S$  211 : SAVESTATE: 43  211 : vi=i(i);wi;tvi=i(i,i,l);w vi=L;N}S$  212 : i=i(i);wi;tvi=i(i,i,l);wi i=L;N}S$  213 : =i(i);wi;tvi=i(i,i,l);wi; =L;N}S$  214 : i(i);wi;tvi=i(i,i,l);wi;t L;N}S$  215 :L->i i(i);wi;tvi=i(i,i,l);wi;t L;N}S$  216 : SAVESTATE: 44  216 : i(i);wi;tvi=i(i,i,l);wi;t i;N}S$  217 : (i);wi;tvi=i(i,i,l);wi;tv ;N}S$  218 : TNS\_NS\_NORULECHAIN/NS\_NORULE  218 : RESTATE  218 : i(i);wi;tvi=i(i,i,l);wi;t L;N}S$  219 :L->i(U) i(i);wi;tvi=i(i,i,l);wi;t L;N}S$  220 : SAVESTATE: 44  220 : i(i);wi;tvi=i(i,i,l);wi;t i(U);N}S$  221 : (i);wi;tvi=i(i,i,l);wi;tv (U);N}S$  222 : i);wi;tvi=i(i,i,l);wi;tvi U);N}S$  223 :U->i i);wi;tvi=i(i,i,l);wi;tvi U);N}S$  224 : SAVESTATE: 45  224 : i);wi;tvi=i(i,i,l);wi;tvi i);N}S$  225 : );wi;tvi=i(i,i,l);wi;tvi= );N}S$  226 : ;wi;tvi=i(i,i,l);wi;tvi=l ;N}S$  227 : wi;tvi=i(i,i,l);wi;tvi=l; N}S$  228 :N->wE;N wi;tvi=i(i,i,l);wi;tvi=l; N}S$  229 : SAVESTATE: 46  229 : wi;tvi=i(i,i,l);wi;tvi=l; wE;N}S$  230 : i;tvi=i(i,i,l);wi;tvi=l;~ E;N}S$  231 :E->iM i;tvi=i(i,i,l);wi;tvi=l;~ E;N}S$  232 : SAVESTATE: 47  232 : i;tvi=i(i,i,l);wi;tvi=l;~ iM;N}S$  233 : ;tvi=i(i,i,l);wi;tvi=l;~( M;N}S$  234 :M-> ;tvi=i(i,i,l);wi;tvi=l;~( M;N}S$  235 : SAVESTATE: 48  235 : ;tvi=i(i,i,l);wi;tvi=l;~( ;N}S$  236 : tvi=i(i,i,l);wi;tvi=l;~(i N}S$  237 :N->tY;N tvi=i(i,i,l);wi;tvi=l;~(i N}S$  238 : SAVESTATE: 49  238 : tvi=i(i,i,l);wi;tvi=l;~(i tY;N}S$  239 : vi=i(i,i,l);wi;tvi=l;~(i< Y;N}S$  240 :Y->vi vi=i(i,i,l);wi;tvi=l;~(i< Y;N}S$  241 : SAVESTATE: 50  241 : vi=i(i,i,l);wi;tvi=l;~(i< vi;N}S$  242 : i=i(i,i,l);wi;tvi=l;~(i<l i;N}S$  243 : =i(i,i,l);wi;tvi=l;~(i<l) ;N}S$  244 : TNS\_NS\_NORULECHAIN/NS\_NORULE  244 : RESTATE  244 : vi=i(i,i,l);wi;tvi=l;~(i< Y;N}S$  245 :Y->vi=L vi=i(i,i,l);wi;tvi=l;~(i< Y;N}S$  246 : SAVESTATE: 50  246 : vi=i(i,i,l);wi;tvi=l;~(i< vi=L;N}S$  247 : i=i(i,i,l);wi;tvi=l;~(i<l i=L;N}S$  248 : =i(i,i,l);wi;tvi=l;~(i<l) =L;N}S$  249 : i(i,i,l);wi;tvi=l;~(i<l){ L;N}S$  250 :L->i i(i,i,l);wi;tvi=l;~(i<l){ L;N}S$  251 : SAVESTATE: 51  251 : i(i,i,l);wi;tvi=l;~(i<l){ i;N}S$  252 : (i,i,l);wi;tvi=l;~(i<l){w ;N}S$  253 : TNS\_NS\_NORULECHAIN/NS\_NORULE  253 : RESTATE  253 : i(i,i,l);wi;tvi=l;~(i<l){ L;N}S$  254 :L->i(U) i(i,i,l);wi;tvi=l;~(i<l){ L;N}S$  255 : SAVESTATE: 51  255 : i(i,i,l);wi;tvi=l;~(i<l){ i(U);N}S$  256 : (i,i,l);wi;tvi=l;~(i<l){w (U);N}S$  257 : i,i,l);wi;tvi=l;~(i<l){wi U);N}S$  258 :U->i i,i,l);wi;tvi=l;~(i<l){wi U);N}S$  259 : SAVESTATE: 52  259 : i,i,l);wi;tvi=l;~(i<l){wi i);N}S$  260 : ,i,l);wi;tvi=l;~(i<l){wi; );N}S$  261 : TNS\_NS\_NORULECHAIN/NS\_NORULE  261 : RESTATE  261 : i,i,l);wi;tvi=l;~(i<l){wi U);N}S$  262 :U->iU i,i,l);wi;tvi=l;~(i<l){wi U);N}S$  263 : SAVESTATE: 52  263 : i,i,l);wi;tvi=l;~(i<l){wi iU);N}S$  264 : ,i,l);wi;tvi=l;~(i<l){wi; U);N}S$  265 : TNS\_NS\_NORULECHAIN/NS\_NORULE  265 : RESTATE  265 : i,i,l);wi;tvi=l;~(i<l){wi U);N}S$  266 :U->i,U i,i,l);wi;tvi=l;~(i<l){wi U);N}S$  267 : SAVESTATE: 52  267 : i,i,l);wi;tvi=l;~(i<l){wi i,U);N}S$  268 : ,i,l);wi;tvi=l;~(i<l){wi; ,U);N}S$  269 : i,l);wi;tvi=l;~(i<l){wi;i U);N}S$  270 :U->i i,l);wi;tvi=l;~(i<l){wi;i U);N}S$  271 : SAVESTATE: 53  271 : i,l);wi;tvi=l;~(i<l){wi;i i);N}S$  272 : ,l);wi;tvi=l;~(i<l){wi;i= );N}S$  273 : TNS\_NS\_NORULECHAIN/NS\_NORULE  273 : RESTATE  273 : i,l);wi;tvi=l;~(i<l){wi;i U);N}S$  274 :U->iU i,l);wi;tvi=l;~(i<l){wi;i U);N}S$  275 : SAVESTATE: 53  275 : i,l);wi;tvi=l;~(i<l){wi;i iU);N}S$  276 : ,l);wi;tvi=l;~(i<l){wi;i= U);N}S$  277 : TNS\_NS\_NORULECHAIN/NS\_NORULE  277 : RESTATE  277 : i,l);wi;tvi=l;~(i<l){wi;i U);N}S$  278 :U->i,U i,l);wi;tvi=l;~(i<l){wi;i U);N}S$  279 : SAVESTATE: 53  279 : i,l);wi;tvi=l;~(i<l){wi;i i,U);N}S$  280 : ,l);wi;tvi=l;~(i<l){wi;i= ,U);N}S$  281 : l);wi;tvi=l;~(i<l){wi;i=i U);N}S$  282 :U->l l);wi;tvi=l;~(i<l){wi;i=i U);N}S$  283 : SAVESTATE: 54  283 : l);wi;tvi=l;~(i<l){wi;i=i l);N}S$  284 : );wi;tvi=l;~(i<l){wi;i=i+ );N}S$  285 : ;wi;tvi=l;~(i<l){wi;i=i+l ;N}S$  286 : wi;tvi=l;~(i<l){wi;i=i+l; N}S$  287 :N->wE;N wi;tvi=l;~(i<l){wi;i=i+l; N}S$  288 : SAVESTATE: 55  288 : wi;tvi=l;~(i<l){wi;i=i+l; wE;N}S$  289 : i;tvi=l;~(i<l){wi;i=i+l;} E;N}S$  290 :E->iM i;tvi=l;~(i<l){wi;i=i+l;} E;N}S$  291 : SAVESTATE: 56  291 : i;tvi=l;~(i<l){wi;i=i+l;} iM;N}S$  292 : ;tvi=l;~(i<l){wi;i=i+l;}i M;N}S$  293 :M-> ;tvi=l;~(i<l){wi;i=i+l;}i M;N}S$  294 : SAVESTATE: 57  294 : ;tvi=l;~(i<l){wi;i=i+l;}i ;N}S$  295 : tvi=l;~(i<l){wi;i=i+l;}i= N}S$  296 :N->tY;N tvi=l;~(i<l){wi;i=i+l;}i= N}S$  297 : SAVESTATE: 58  297 : tvi=l;~(i<l){wi;i=i+l;}i= tY;N}S$  298 : vi=l;~(i<l){wi;i=i+l;}i=l Y;N}S$  299 :Y->vi vi=l;~(i<l){wi;i=i+l;}i=l Y;N}S$  300 : SAVESTATE: 59  300 : vi=l;~(i<l){wi;i=i+l;}i=l vi;N}S$  301 : i=l;~(i<l){wi;i=i+l;}i=l; i;N}S$  302 : =l;~(i<l){wi;i=i+l;}i=l;i ;N}S$  303 : TNS\_NS\_NORULECHAIN/NS\_NORULE  303 : RESTATE  303 : vi=l;~(i<l){wi;i=i+l;}i=l Y;N}S$  304 :Y->vi=L vi=l;~(i<l){wi;i=i+l;}i=l Y;N}S$  305 : SAVESTATE: 59  305 : vi=l;~(i<l){wi;i=i+l;}i=l vi=L;N}S$  306 : i=l;~(i<l){wi;i=i+l;}i=l; i=L;N}S$  307 : =l;~(i<l){wi;i=i+l;}i=l;i =L;N}S$  308 : l;~(i<l){wi;i=i+l;}i=l;i= L;N}S$  309 :L->l l;~(i<l){wi;i=i+l;}i=l;i= L;N}S$  310 : SAVESTATE: 60  310 : l;~(i<l){wi;i=i+l;}i=l;i= l;N}S$  311 : ;~(i<l){wi;i=i+l;}i=l;i=l ;N}S$  312 : ~(i<l){wi;i=i+l;}i=l;i=l; N}S$  313 :N->~KZN ~(i<l){wi;i=i+l;}i=l;i=l; N}S$  314 : SAVESTATE: 61  314 : ~(i<l){wi;i=i+l;}i=l;i=l; ~KZN}S$  315 : (i<l){wi;i=i+l;}i=l;i=l;t KZN}S$  316 :K->(U<U) (i<l){wi;i=i+l;}i=l;i=l;t KZN}S$  317 : SAVESTATE: 62  317 : (i<l){wi;i=i+l;}i=l;i=l;t (U<U)ZN}S$  318 : i<l){wi;i=i+l;}i=l;i=l;tv U<U)ZN}S$  319 :U->i i<l){wi;i=i+l;}i=l;i=l;tv U<U)ZN}S$  320 : SAVESTATE: 63  320 : i<l){wi;i=i+l;}i=l;i=l;tv i<U)ZN}S$  321 : <l){wi;i=i+l;}i=l;i=l;tvi <U)ZN}S$  322 : l){wi;i=i+l;}i=l;i=l;tvi= U)ZN}S$  323 :U->l l){wi;i=i+l;}i=l;i=l;tvi= U)ZN}S$  324 : SAVESTATE: 64  324 : l){wi;i=i+l;}i=l;i=l;tvi= l)ZN}S$  325 : ){wi;i=i+l;}i=l;i=l;tvi=l )ZN}S$  326 : {wi;i=i+l;}i=l;i=l;tvi=l; ZN}S$  327 :Z->{N} {wi;i=i+l;}i=l;i=l;tvi=l; ZN}S$  328 : SAVESTATE: 65  328 : {wi;i=i+l;}i=l;i=l;tvi=l; {N}N}S$  329 : wi;i=i+l;}i=l;i=l;tvi=l;t N}N}S$  330 :N->wE;N wi;i=i+l;}i=l;i=l;tvi=l;t N}N}S$  331 : SAVESTATE: 66  331 : wi;i=i+l;}i=l;i=l;tvi=l;t wE;N}N}S$  332 : i;i=i+l;}i=l;i=l;tvi=l;tv E;N}N}S$  333 :E->iM i;i=i+l;}i=l;i=l;tvi=l;tv E;N}N}S$  334 : SAVESTATE: 67  334 : i;i=i+l;}i=l;i=l;tvi=l;tv iM;N}N}S$  335 : ;i=i+l;}i=l;i=l;tvi=l;tvi M;N}N}S$  336 :M-> ;i=i+l;}i=l;i=l;tvi=l;tvi M;N}N}S$  337 : SAVESTATE: 68  337 : ;i=i+l;}i=l;i=l;tvi=l;tvi ;N}N}S$  338 : i=i+l;}i=l;i=l;tvi=l;tvi= N}N}S$  339 :N->i=E;N i=i+l;}i=l;i=l;tvi=l;tvi= N}N}S$  340 : SAVESTATE: 69  340 : i=i+l;}i=l;i=l;tvi=l;tvi= i=E;N}N}S$  341 : =i+l;}i=l;i=l;tvi=l;tvi=i =E;N}N}S$  342 : i+l;}i=l;i=l;tvi=l;tvi=i( E;N}N}S$  343 :E->iM i+l;}i=l;i=l;tvi=l;tvi=i( E;N}N}S$  344 : SAVESTATE: 70  344 : i+l;}i=l;i=l;tvi=l;tvi=i( iM;N}N}S$  345 : +l;}i=l;i=l;tvi=l;tvi=i(i M;N}N}S$  346 :M->+E +l;}i=l;i=l;tvi=l;tvi=i(i M;N}N}S$  347 : SAVESTATE: 71  347 : +l;}i=l;i=l;tvi=l;tvi=i(i +E;N}N}S$  348 : l;}i=l;i=l;tvi=l;tvi=i(i, E;N}N}S$  349 :E->lM l;}i=l;i=l;tvi=l;tvi=i(i, E;N}N}S$  350 : SAVESTATE: 72  350 : l;}i=l;i=l;tvi=l;tvi=i(i, lM;N}N}S$  351 : ;}i=l;i=l;tvi=l;tvi=i(i,i M;N}N}S$  352 :M-> ;}i=l;i=l;tvi=l;tvi=i(i,i M;N}N}S$  353 : SAVESTATE: 73  353 : ;}i=l;i=l;tvi=l;tvi=i(i,i ;N}N}S$  354 : }i=l;i=l;tvi=l;tvi=i(i,i, N}N}S$  355 :N-> }i=l;i=l;tvi=l;tvi=i(i,i, N}N}S$  356 : SAVESTATE: 74  356 : }i=l;i=l;tvi=l;tvi=i(i,i, }N}S$  357 : i=l;i=l;tvi=l;tvi=i(i,i,i N}S$  358 :N->i=E;N i=l;i=l;tvi=l;tvi=i(i,i,i N}S$  359 : SAVESTATE: 75  359 : i=l;i=l;tvi=l;tvi=i(i,i,i i=E;N}S$  360 : =l;i=l;tvi=l;tvi=i(i,i,i) =E;N}S$  361 : l;i=l;tvi=l;tvi=i(i,i,i); E;N}S$  362 :E->lM l;i=l;tvi=l;tvi=i(i,i,i); E;N}S$  363 : SAVESTATE: 76  363 : l;i=l;tvi=l;tvi=i(i,i,i); lM;N}S$  364 : ;i=l;tvi=l;tvi=i(i,i,i);w M;N}S$  365 :M-> ;i=l;tvi=l;tvi=i(i,i,i);w M;N}S$  366 : SAVESTATE: 77  366 : ;i=l;tvi=l;tvi=i(i,i,i);w ;N}S$  367 : i=l;tvi=l;tvi=i(i,i,i);wi N}S$  368 :N->i=E;N i=l;tvi=l;tvi=i(i,i,i);wi N}S$  369 : SAVESTATE: 78  369 : i=l;tvi=l;tvi=i(i,i,i);wi i=E;N}S$  370 : =l;tvi=l;tvi=i(i,i,i);wi; =E;N}S$  371 : l;tvi=l;tvi=i(i,i,i);wi;} E;N}S$  372 :E->lM l;tvi=l;tvi=i(i,i,i);wi;} E;N}S$  373 : SAVESTATE: 79  373 : l;tvi=l;tvi=i(i,i,i);wi;} lM;N}S$  374 : ;tvi=l;tvi=i(i,i,i);wi;}$ M;N}S$  375 :M-> ;tvi=l;tvi=i(i,i,i);wi;}$ M;N}S$  376 : SAVESTATE: 80  376 : ;tvi=l;tvi=i(i,i,i);wi;}$ ;N}S$  377 : tvi=l;tvi=i(i,i,i);wi;}$ N}S$  378 :N->tY;N tvi=l;tvi=i(i,i,i);wi;}$ N}S$  379 : SAVESTATE: 81  379 : tvi=l;tvi=i(i,i,i);wi;}$ tY;N}S$  380 : vi=l;tvi=i(i,i,i);wi;}$ Y;N}S$  381 :Y->vi vi=l;tvi=i(i,i,i);wi;}$ Y;N}S$  382 : SAVESTATE: 82  382 : vi=l;tvi=i(i,i,i);wi;}$ vi;N}S$  383 : i=l;tvi=i(i,i,i);wi;}$ i;N}S$  384 : =l;tvi=i(i,i,i);wi;}$ ;N}S$  385 : TNS\_NS\_NORULECHAIN/NS\_NORULE  385 : RESTATE  385 : vi=l;tvi=i(i,i,i);wi;}$ Y;N}S$  386 :Y->vi=L vi=l;tvi=i(i,i,i);wi;}$ Y;N}S$  387 : SAVESTATE: 82  387 : vi=l;tvi=i(i,i,i);wi;}$ vi=L;N}S$  388 : i=l;tvi=i(i,i,i);wi;}$ i=L;N}S$  389 : =l;tvi=i(i,i,i);wi;}$ =L;N}S$  390 : l;tvi=i(i,i,i);wi;}$ L;N}S$  391 :L->l l;tvi=i(i,i,i);wi;}$ L;N}S$  392 : SAVESTATE: 83  392 : l;tvi=i(i,i,i);wi;}$ l;N}S$  393 : ;tvi=i(i,i,i);wi;}$ ;N}S$  394 : tvi=i(i,i,i);wi;}$ N}S$  395 :N->tY;N tvi=i(i,i,i);wi;}$ N}S$  396 : SAVESTATE: 84  396 : tvi=i(i,i,i);wi;}$ tY;N}S$  397 : vi=i(i,i,i);wi;}$ Y;N}S$  398 :Y->vi vi=i(i,i,i);wi;}$ Y;N}S$  399 : SAVESTATE: 85  399 : vi=i(i,i,i);wi;}$ vi;N}S$  400 : i=i(i,i,i);wi;}$ i;N}S$  401 : =i(i,i,i);wi;}$ ;N}S$  402 : TNS\_NS\_NORULECHAIN/NS\_NORULE  402 : RESTATE  402 : vi=i(i,i,i);wi;}$ Y;N}S$  403 :Y->vi=L vi=i(i,i,i);wi;}$ Y;N}S$  404 : SAVESTATE: 85  404 : vi=i(i,i,i);wi;}$ vi=L;N}S$  405 : i=i(i,i,i);wi;}$ i=L;N}S$  406 : =i(i,i,i);wi;}$ =L;N}S$  407 : i(i,i,i);wi;}$ L;N}S$  408 :L->i i(i,i,i);wi;}$ L;N}S$  409 : SAVESTATE: 86  409 : i(i,i,i);wi;}$ i;N}S$  410 : (i,i,i);wi;}$ ;N}S$  411 : TNS\_NS\_NORULECHAIN/NS\_NORULE  411 : RESTATE  411 : i(i,i,i);wi;}$ L;N}S$  412 :L->i(U) i(i,i,i);wi;}$ L;N}S$  413 : SAVESTATE: 86  413 : i(i,i,i);wi;}$ i(U);N}S$  414 : (i,i,i);wi;}$ (U);N}S$  415 : i,i,i);wi;}$ U);N}S$  416 :U->i i,i,i);wi;}$ U);N}S$  417 : SAVESTATE: 87  417 : i,i,i);wi;}$ i);N}S$  418 : ,i,i);wi;}$ );N}S$  419 : TNS\_NS\_NORULECHAIN/NS\_NORULE  419 : RESTATE  419 : i,i,i);wi;}$ U);N}S$  420 :U->iU i,i,i);wi;}$ U);N}S$  421 : SAVESTATE: 87  421 : i,i,i);wi;}$ iU);N}S$  422 : ,i,i);wi;}$ U);N}S$  423 : TNS\_NS\_NORULECHAIN/NS\_NORULE  423 : RESTATE  423 : i,i,i);wi;}$ U);N}S$  424 :U->i,U i,i,i);wi;}$ U);N}S$  425 : SAVESTATE: 87  425 : i,i,i);wi;}$ i,U);N}S$  426 : ,i,i);wi;}$ ,U);N}S$  427 : i,i);wi;}$ U);N}S$  428 :U->i i,i);wi;}$ U);N}S$  429 : SAVESTATE: 88  429 : i,i);wi;}$ i);N}S$  430 : ,i);wi;}$ );N}S$  431 : TNS\_NS\_NORULECHAIN/NS\_NORULE  431 : RESTATE  431 : i,i);wi;}$ U);N}S$  432 :U->iU i,i);wi;}$ U);N}S$  433 : SAVESTATE: 88  433 : i,i);wi;}$ iU);N}S$  434 : ,i);wi;}$ U);N}S$  435 : TNS\_NS\_NORULECHAIN/NS\_NORULE  435 : RESTATE  435 : i,i);wi;}$ U);N}S$  436 :U->i,U i,i);wi;}$ U);N}S$  437 : SAVESTATE: 88  437 : i,i);wi;}$ i,U);N}S$  438 : ,i);wi;}$ ,U);N}S$  439 : i);wi;}$ U);N}S$  440 :U->i i);wi;}$ U);N}S$  441 : SAVESTATE: 89  441 : i);wi;}$ i);N}S$  442 : );wi;}$ );N}S$  443 : ;wi;}$ ;N}S$  444 : wi;}$ N}S$  445 :N->wE;N wi;}$ N}S$  446 : SAVESTATE: 90  446 : wi;}$ wE;N}S$  447 : i;}$ E;N}S$  448 :E->iM i;}$ E;N}S$  449 : SAVESTATE: 91  449 : i;}$ iM;N}S$  450 : ;}$ M;N}S$  451 :M-> ;}$ M;N}S$  452 : SAVESTATE: 92  452 : ;}$ ;N}S$  453 : }$ N}S$  454 :N-> }$ N}S$  455 : SAVESTATE: 93  455 : }$ }S$  456 : $ S$  457 :S-> $ S$  458 : SAVESTATE: 94  458 : $ $  459 :  460 : LENTA\_END  461 : ------>LENTA\_END |

Таблица 1 - Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| S | S-> m{N}S  S-> tfi(F){NrU;}S  S-> λ | Стартовые правила, описывающее общую структуру программы |
| N | N-> tY;N  N-> i=E;N  N-> wE;N  N-> ~KZN  N-> λ | Правила для операторов |
| U | U-> l  U-> i  U->lU  U->iU  U->i,U  U->l,U | Только литерал или идентификатор (один или несколько) |
| Y | Y-> vi  Y->vi=L | Правило определения переменной |
| E | E-> iM  E-> lM  E-> (E)M  E-> i(W)M | Правила выражений |
| M | M-> +E  M-> -E  M-> \*E  M-> /E  M-> %E  M-> λ | Правила арифметический операторов |
| F | F-> tP  F-> tP,F  F-> λ | Правила определения параметров функции |
| P | P-> pi | Правило определения параметров функции |
| W | W-> i  W-> l  W-> i,W  W-> l,W  W-> λ | Правила вызова функции с параметрами |
| L | L->I  L->l  L->i(U)  L->i()  L->iL  L->lL  L->+i  L->+l  L->+iL  L->+lL  L->-i  L->-l  L->-iL  L->-lL  L->\*i  L->\*l  L->\*iL  L->\*lL  L->/i  L->/l  L->/iL  L->/lL  L->%i  L->%l  L->%iL  L->%lL | Начальная инициализация |
| Z | Z->{N} | Тело цикла |

# **Приложение Г**

Листинг 1. Программная реализация механизма преобразования в ПОЛИЗ

|  |
| --- |
| bool polishNotation(int lextable\_pos, LT::LexTable& lextable, IT::IdTable& idtable)  {  stack<LT::Entry\*> stk; //создаем стек для хранения временных операций  queue<LT::Entry\*> result;  bool function = false;  int quantityParm = 0;  int i = ++lextable\_pos;  for (; lextable.table[i]->lexema != LEX\_SEMICOLON && (lextable.table[i]->lexema != LEX\_RIGHTHESIS || !stk.empty()); i++)  {  switch (lextable.table[i]->lexema)  {  case LEX\_ID: //операнды  case LEX\_LITERAL:  if (idtable.table[lextable.table[i]->idxTI]->idtype == IT::IDTYPE::F)  {  quantityParm = 0;  function = true;  result.push(lextable.table[i]);  break;  }  if (function && !quantityParm)  quantityParm++;  result.push(lextable.table[i]);  break;  case LEX\_PLUS:  case LEX\_MINUS:  case LEX\_TIMES:  case LEX\_DIVIDE:  case LEX\_MODULE:  if (stk.empty() || stk.top()->lexema == LEX\_LEFTHESIS)  stk.push(lextable.table[i]);  else  {  int prioritet = priority(lextable.table[i]->sign);  if (priority(stk.top()->sign) >= prioritet)  {  result.push(stk.top());  stk.pop();  }  stk.push(lextable.table[i]);  }  break;  case LEX\_LEFTHESIS:  stk.push(lextable.table[i]);  break;  case LEX\_RIGHTHESIS:  while (stk.top()->lexema != LEX\_LEFTHESIS)  {  result.push(stk.top());  stk.pop();  }  stk.pop();  if (function)  {  result.push(new LT::Entry('@'));  result.push(new LT::Entry('0' + quantityParm));  function = false;  }  break;  case LEX\_COMMA:  if (function)  quantityParm++;  while (stk.top()->lexema != LEX\_LEFTHESIS)  {  result.push(stk.top());  stk.pop();  }  break;  case LEX\_MORE:  case LEX\_LESS:  result.push(lextable.table[i]);  break;  }  }  while (!stk.empty())  {  result.push(stk.top());  stk.pop();  }  for (int j = lextable\_pos; j < i; j++)  {  if (!result.empty())  {  lextable.table[j] = result.front();  lextable.table[j]->sn = lextable.table[j - 1]->sn;  lextable.table[j]->tn = lextable.table[j - 1]->tn + 1;  result.pop();  }  else  {  lextable.table[j] = new LT::Entry('#', lextable.table[j]->sn = lextable.table[j - 1]->sn, lextable.table[j]->tn = lextable.table[j - 1]->tn + 1);  }  }  return true;  } |

# **Приложение Д**

Листинг 1. Исходный код на языке ассемблера для контрольного примера

|  |
| --- |
| .586 ; система команд (процессор Pentium)  .model flat, stdcall ; модель памяти, соглашение о вызовах  includelib kernel32.lib  includelib libucrt.lib  includelib "D:\KAD-2023-CP\KAD-2023\Debug\StaticLib.lib"  ExitProcess PROTO: dword ; прототип функции для завершения процесса Windows  EXTRN lenght: proc  EXTRN write\_short: proc  EXTRN write\_str : proc  EXTRN copy: proc  EXTRN getLocalTimeAndDate: proc  EXTRN random: proc  .stack 4096  .const ; сегмент констант - литералы  nulError byte 'error divided by zero', 0  nul sword 0, 0  L0 byte "Hi!", 0  L1 sword -3  L2 sword 7  L3 sword 6  L4 sword 3  L5 sword 4  L6 sword 2  L7 byte "Тест строки", 0  L8 sword 0  L9 sword 100  L10 sword 10  L11 sword 1  L12 sword 15  L13 sword 5  L14 byte "Hello", 0  L15 byte "How are you", 0  L16 byte "Шестнадцатиричное число A в 10й сс:", 0  L17 byte "Двоичное число 1010 в 10й сс:", 0  .data ; сегмент данных - переменные и параметры  sumRes\_sum sword 0  lenOne\_differenceInLength sword 0  lenTwo\_differenceInLength sword 0  res\_differenceInLength sword 0  hi\_getHi dword ?  a\_main sword 0  strB\_main dword ?  b\_main sword 0  strBCopy\_main dword ?  rand\_main sword 0  date\_main dword ?  i\_main sword 0  h\_main dword ?  c\_main sword 0  sumABC\_main sword 0  firstStroke\_main dword ?  secondStroke\_main dword ?  difBetween\_main sword 0  hexNum\_main sword 0  binNum\_main sword 0  .code ; сегмент кода  ;----------- sum ------------  sum PROC, first\_sum : sword, second\_sum : sword, third\_sum : sword  ; --- сохранить регистры ---  push ebx  push edx  ; ----------------------  push first\_sum  push second\_sum  pop bx  pop ax  add ax, bx  push ax  push third\_sum  pop bx  pop ax  add ax, bx  push ax  pop sumRes\_sum  ; --- восстановить регистры ---  pop edx  pop ebx  ; -------------------------  mov eax, dword ptr sumRes\_sum  ret  sum ENDP  ;------------------------------  ;----------- differenceInLength ------------  differenceInLength PROC, strokeOne\_differenceInLength : word, strokeTwo\_differenceInLength : word  ; --- сохранить регистры ---  push ebx  push edx  ; ----------------------  mov eax, dword ptr strokeOne\_differenceInLength  push eax  call lenght  push eax  pop lenOne\_differenceInLength  mov eax, dword ptr strokeTwo\_differenceInLength  push eax  call lenght  push eax  pop lenTwo\_differenceInLength  push lenOne\_differenceInLength  push lenTwo\_differenceInLength  pop bx  pop ax  sub ax, bx  push ax  pop res\_differenceInLength  ; --- восстановить регистры ---  pop edx  pop ebx  ; -------------------------  mov eax, dword ptr res\_differenceInLength  ret  differenceInLength ENDP  ;------------------------------  ;----------- getHi ------------  getHi PROC  ; --- сохранить регистры ---  push ebx  push edx  ; ----------------------  mov hi\_getHi, offset L0  push hi\_getHi  call write\_str  ; --- восстановить регистры ---  pop eax  pop edx  pop ebx  ; -------------------------  mov eax, dword ptr hi\_getHi  ret  getHi ENDP  ;------------------------------  ;----------- MAIN ------------  main PROC  push L1  pop a\_main  push a\_main  push L2  pop bx  pop ax  add ax, bx  push ax  push L3  pop bx  pop ax  imul ax, bx  push ax  push L4  pop bx  pop ax  cmp nul, bx  je errorExit  cdq  idiv bx  push ax  push L5  pop bx  pop ax  sub ax, bx  push ax  push L6  pop bx  pop ax  cdq  idiv bx  push dx  pop a\_main  push a\_main  call write\_short  mov strB\_main, offset L7  mov eax, dword ptr strB\_main  push eax  call lenght  push eax  pop b\_main  push b\_main  call write\_short  mov eax, dword ptr L5  push eax  mov eax, dword ptr strB\_main  push eax  mov eax, dword ptr strBCopy\_main  push eax  call copy  mov strBCopy\_main, eax  push strBCopy\_main  call write\_str  mov eax, dword ptr L9  push eax  mov eax, dword ptr L8  push eax  call random  push eax  pop rand\_main  push rand\_main  call write\_short  call getLocalTimeAndDate  mov date\_main, eax  push date\_main  call write\_str  push L8  pop i\_main  cyclenext0:  mov dx, i\_main  cmp dx, L10  jg cycle0  push i\_main  call write\_short  push i\_main  push L11  pop bx  pop ax  add ax, bx  push ax  pop i\_main  jmp cyclenext0  cycle0:  call getHi  mov h\_main, eax  push L10  pop a\_main  push L12  pop b\_main  push L13  pop c\_main  mov eax, dword ptr c\_main  push eax  mov eax, dword ptr b\_main  push eax  mov eax, dword ptr a\_main  push eax  call sum  push eax  pop sumABC\_main  push sumABC\_main  call write\_short  mov firstStroke\_main, offset L14  mov secondStroke\_main, offset L15  mov eax, dword ptr secondStroke\_main  push eax  mov eax, dword ptr firstStroke\_main  push eax  call differenceInLength  push eax  pop difBetween\_main  push difBetween\_main  call write\_short  push L10  pop hexNum\_main  push offset L16  call write\_str  push hexNum\_main  call write\_short  push L10  pop binNum\_main  push offset L17  call write\_str  push binNum\_main  call write\_short  jmp goodExit  errorExit:  push offset nulError  call write\_str  goodExit:  push 0  call ExitProcess  main ENDP  end main |

# **Приложение Е**

Графический материал. Граф дерева разбора.