Содержание

[Введение 5](#_Toc58708507)

[1. Спецификация языка программирования 6](#_Toc58708508)

[1.1. Характеристика языка программирования 6](#_Toc58708509)

[1.2. Определение алфавита языка программирования 6](#_Toc58708510)

[1.3. Применяемые сепараторы 7](#_Toc58708511)

[1.4. Применяемые кодировки 7](#_Toc58708512)

[1.5. Типы данных 7](#_Toc58708513)

[1.6. Преобразование типов данных 8](#_Toc58708514)

[1.7. Идентификаторы 9](#_Toc58708515)

[1.8. Литералы 9](#_Toc58708516)

[1.9. Объявление данных 9](#_Toc58708517)

[1.10. Инициализация данных 10](#_Toc58708518)

[1.11. Инструкции языка 10](#_Toc58708519)

[1.12. Операции языка 10](#_Toc58708520)

[1.13. Выражение и их вычисление 11](#_Toc58708521)

[1.14. Конструкции языка 11](#_Toc58708522)

[1.15. Область видимости идентификаторов 12](#_Toc58708523)

[1.16. Семантические проверки 12](#_Toc58708524)

[1.17. Распределение оперативной памяти на этапе выполнения 13](#_Toc58708525)

[1.18. Стандартная библиотека и её состав 13](#_Toc58708526)

[1.19. Ввод и вывод данных 14](#_Toc58708527)

[1.20. Точка входа 14](#_Toc58708528)

[1.21. Препроцессор 14](#_Toc58708529)

[1.22. Соглашения о вызовах 14](#_Toc58708530)

[1.23. Объектный код 15](#_Toc58708531)

[1.24. Классификация сообщений транслятора 15](#_Toc58708532)

[1.25. Контрольный пример 15](#_Toc58708533)

[2. Структура транслятора 16](#_Toc58708534)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 16](#_Toc58708535)

[2.2 Перечень протоколов, формируемых транслятором и их содержимое 17](#_Toc58708536)

[2.3 Протоколы, формируемые транслятором 17](#_Toc58708537)

[3 Разработка лексического анализатора 18](#_Toc58708538)

[3.1 Структура лексического анализатора 18](#_Toc58708539)

[3.2 Контроль входных символов 19](#_Toc58708540)

[3.3 Удаление избыточных символов 19](#_Toc58708541)

[3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов 20](#_Toc58708542)

[3.5 Основные структуры данных 22](#_Toc58708543)

[3.6 Принцип обработки ошибок 23](#_Toc58708544)

[3.7 Структура и перечень сообщений лексического анализатора 23](#_Toc58708545)

[3.8 Параметры лексического анализатора 24](#_Toc58708546)

[3.9 Алгоритм лексического анализа 24](#_Toc58708547)

[3.10 Контрольный пример 24](#_Toc58708549)

[4. Разработка синтаксического анализатора 25](#_Toc58708550)

[4.1 Структура синтаксического анализатора 25](#_Toc58708551)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 25](#_Toc58708552)

[4.3 Построение конечного магазинного автомата 26](#_Toc58708553)

[4.4 Основные структуры данных 27](#_Toc58708554)

[4.5 Описание алгоритма синтаксического разбора 27](#_Toc58708555)

[4.6 Структура и перечень сообщений синтаксического анализатора 27](#_Toc58708556)

[4.7. Параметры синтаксического анализатора и режимы его работы 27](#_Toc58708557)

[4.8. Принцип обработки ошибок 28](#_Toc58708558)

[4.9. Контрольный пример 28](#_Toc58708559)

[5 Разработка семантического анализатора 29](#_Toc58708560)

[5.1 Структура семантического анализатора 29](#_Toc58708561)

[5.2 Функции семантического анализатора 29](#_Toc58708562)

[5.3 Структура и перечень сообщений семантического анализатора 29](#_Toc58708563)

[5.4 Принцип обработки ошибок 30](#_Toc58708564)

[5.5 Контрольный пример 30](#_Toc58708565)

[6 Преобразование выражений 31](#_Toc58708566)

[6.1 Выражения, допускаемые языком 31](#_Toc58708567)

[6.2 Польская запись 31](#_Toc58708568)

[6.3 Программная реализация обработки выражений 31](#_Toc58708569)

[6.4 Контрольный пример 32](#_Toc58708570)

[7. Генерация кода 33](#_Toc58708571)

[7.1 Структура генератора кода 33](#_Toc58708572)

[7.2 Представление типов данных в оперативной памяти 33](#_Toc58708573)

[7.3 Статическая библиотека 34](#_Toc58708574)

[7.4 Особенности алгоритма генерации кода 34](#_Toc58708575)

[7.5 Входные параметры генератора кода 34](#_Toc58708576)

[7.6 Контрольный пример 35](#_Toc58708577)

[8. Тестирование транслятора 36](#_Toc58708578)

[8.1 Общие положения 36](#_Toc58708579)

[8.2 Результаты тестирования 36](#_Toc58708580)

[Заключение 37](#_Toc58708581)

[Список использованных источников 38](#_Toc58708582)

[Приложение А 39](#_Toc58708583)

[Приложение Б 41](#_Toc58708585)

[Приложение В 42](#_Toc58708586)

[Приложение Г 46](#_Toc58708587)

[Приложение Д 48](#_Toc58708588)

[Приложение Е 52](#_Toc58708589)

Приложение Ж 54

**Введение**

Целью курсового проекта является разработка транслятора для своего языка программирования: KES-2022. Этот язык программирования предназначен для выполнения простейших операций и арифметических действий над числами. Главная задача транслятора заключается в том, чтобы сделать программу, написанную языке программирования KES-2022, понятной компьютеру. В данном курсовом проекте мой исходный код транслируется на язык ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разбратка структуры транслятора;

– разработка лексического и семантического анализаторов;

– разработка синтаксического анализатора;

– преобразование выражений;

– генерация кода на язык ассемблера;

– тестирование транслятора.

Решения каждой из поставленных задач будут приведены в соответствующих главах курсового проекта, а именно:

В первой главе работы определена спецификация языка программирования, т.е. описан синтаксис и семантика языка.

Во второй главе работы представлена структура транслятора, т.е. перечислены компоненты транслятора, их назначение и принципы взаимодействия, перечень входных параметров, перечень протоколов, формируемых транслятором и их содержимое.

В третьей главе работы показана разработка лексического анализатора, порождающего таблицы лексем и идентификаторов.

В четвертой главе работы рассказывается о синтаксическом анализаторе, который выполняет синтаксический разбор текста с распечаткой протокола разбора и дерева разбора на основе таблицы лексем.

В пятой главе описан семантический анализатор, показана его работа (распечатка выданных сообщений в трёх примерах на разных этапах).

В шестой главе решены вопросы преобразования выражений, допускаемых языком и приведена часть протокола для контрольного примера, отображающая результаты преобразования выражений в польский формат.

В седьмой главе представлена генерация кода, где из промежуточного представления порождается код на целевом языке.

В восьмой главе описывается тестирование транслятора.

# **1. Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

Язык программирования KES-2022 является процедурным, строго- типизированным, не объектно-ориентированным, компилируемым.

**1.2 Определение алфавита языка программирования**

Алфавит языка KES-2022 состоит из следующих множеств символов:

* латинские символы верхнего и нижнего регистра: {A, B, C, …, Z, a, b, c, …, z}
* цифры: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
* специальные символы: {+, -, \*,/,%, >, <,&,!}
* знаки пунктуации языка: {(), {},[], ;, =}
* пробел, символ табуляции, символ перехода на новую строку

## **1.3 Применяемые сепараторы**

Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |
| --- | --- |
| Сепаратор | Область применения |
| ‘пробел' | Служит для разделения. Допускается везде кроме названий идентификаторов и ключевых слов |
| ; | Разделитель программных конструкций |
| { } | Блок функции |
| = | Оператор присваивания |
| ( ) | Блок фактических или формальных параметров функции, а также приоритет арифметических операций |
| , | Разделитель параметров функций |
| +,-,\*,/ ,%,++,--,\*\*,// | Арифметические операции |
| [] | Блок условной конструкции/цикла |
| > < & ! | Логические операции (операции сравнения: больше, меньше, проверка на равенство, на неравенство), используемые в условии цикла/условной конструкции. |

## **1.4 Применяемые кодировки**

Для написания программ язык KES-2022 использует кодировку Windows-1251, содержащую английский алфавит, а также некоторые специальные символы, такие как { } [ ] ( ) , ; ~ + - / \* > < & !.



Рисунок 1.1 – Таблица кодировки Windows-1251

## **1.5 Типы данных**

В языке KES-2022 есть 3 типа данных: знаковый целочисленный, строковый и символьный. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка KES-2022

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| integer | Фундаментальный тип данных. Предусмотрен для объявления целочисленных данных (4 байта). Принимает значения в диапазоне от -231 до 231-1. Автоматически инициализируется нулевым значением. Инициализация по умолчанию: значение 0.  Поддерживаемые операции:  + (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  \* (бинарный) – оператор умножения;  / (бинарный) – оператор деления;  %(бинарный) – оператор остаток от деления; |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
|  | = (бинарный) – оператор присваивания.  В качестве условия цикла/условного оператора поддерживаются следующие логические операции:  > (бинарный) – оператор «больше»;  <(бинарный) – оператор «меньше»;  & (бинарный) – оператор проверки на равенство;  ! (бинарный) – оператор проверки на неравенство. |
| string | Символы, заключённые в ' ' (одинарные кавычки), инициализируются пустой строкой, строковые переменные. Максимальное число символов в литерале 255.  Инициализация по умолчанию: строка нулевой длины ' '. Операции над данными строкового типа: присваивание строковому идентификатору значения другого строкового идентификатора, строкового литерала или значения строковой функции |
| symbol | Фундаментальный тип данных. Используется для работы с символами, каждый из которых занимает 1 байт. Максимальное количество символов – 1.  Инициализация по умолчанию: символ нулевой длины ' '. Операции над данными строкового типа: присваивание символьному идентификатору значения другого символьного идентификатора, символьного литерала или значения символьной функции, а также использование библиотечных функций. |

## **1.6 Преобразование типов данных**

В языке программирования KES-2022 преобразование типов данных не поддерживается. Язык является строго типизированным.

## **1.7 Идентификаторы**

Идентификаторы должны содержать символы латинского алфавита нижнего регистра. Идентификаторы должны начинаться только с символов латинского алфавита, могут содержать цифры. Максимальная длина идентификатора равна восьми символам. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Префикс занимает 8 дополнительных символов. В случае превышения заданной длины, идентификаторы усекаются до длины, равной 16 символов (8 символов на имя идентификатора, 8 символов на префикс). Данные правила действуют для всех типов идентификаторов. Идентификаторы не должны совпадать с ключевыми словами.

Форма Бэкуса-Наура для записи идентификаторов:

<буква> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<идентификатор> ::= <буква> {(<буква> |<цифры> )}

## **1.8 Литералы**

В языке существует 3 типа литералов: целого, строкового и cимвольного типов. Краткое описание литералов представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Последовательность цифр 0…9 с предшествующим знаком минус или без него (знак минус не должен отделяться пробелом) |
| Строковые литералы | Набор символов (от 1 до 255), заключённых в ' ' |
| Символьные литералы | Символ , заключенный в ' ' |

Представление строковых литералов в форме Бэкуса-Наура:

<цифра> ::= { 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 }

<буква> ::= { A | B | C | … | Z | a | b | c | … | z |}

<символ> ::= { = | + | - | \* | / | ; | , | ? | ! | ( | ) | { | } | . }

<строковый литерал> ::= { (<буква> | <цифра> | <символ>) }

## **1.9 Объявление данных**

Для объявления переменной используется ключевое слово type, после которого указывается типа данных и имя идентификатора. Допускается инициализация при объявлении.

Пример объявления числового типа с инициализацией:

**type integer** num1 = -1

Пример объявления переменной строкового типа с инициализацией:

**type string** str1= 'hello world'

Пример объявления переменной символьного типа с инициализацией:

**type symbol** ch1= 'c'

Для объявления функций используется ключевое слово function, перед которым указывается тип функции, после которого имя функции. Далее обязателен список параметров и тело функции.

## **1.10 Инициализация данных**

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. При объявлении без инициализации предусмотрены значения по умолчанию: значение 0 для типа **integer** и строка длины 0 ('') для типа **string** и **symbol**.

## **1.11 Инструкции языка**

Инструкции языка программирования KES-2022 представлены в общем виде в таблице 1.4.

Таблица 1.4 – Инструкции языка программирования KES-2022

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Объявление переменной | **type** <тип данных> <идентификатор>. |
| Объявление переменной с явной инициализацией | **type** <тип данных> <идентификатор> = <значение>|<выражение>;  Значение – литерал, идентификатор, вызов функции соответствующего типа данных |
| Объявление функции | <тип данных> **function** <идентификатор> (<тип данных> <идентификатор>, …)  {  / программный блок /  **return** <идентификатор/литерал>.  } |
| Вызов функции | <идентификатор> (<идентификатор>, ...) |
| Присвоение значения | <идентификатор> = <значение>; |
| Вывод данных | **write** <идентификатор/литерал>; |
| Вывод данных с переходом на новую строку | **writeline** <идентификатор/литерал>; |
| Возвращаемое значение | **return** <литерал/идентификатор>. |

## **1.12 Операции языка**

В языке KES-2022 предусмотрены следующие операции с данными. Приоритетность операции умножения выше приоритета операций сложения и вычитания. Для установки наивысшего приоритета используются круглые скобки. Операции языка представлены в таблице 1.5.

Таблица 1.5 – Операции языка программирования KES-2022

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические операции языка | + – сложение  - – вычитание  \* – умножение |

## Продолжение таблицы 1.5

|  |  |
| --- | --- |
|  | / – деление  % – остаток от деления  = – присваивание |
| Логические | > – больше  < – меньше  & – проверка на равенство  ! – проверка на неравенство |

## **1.13 Выражения и их вычисления**

В языке KES-2022 существует ряд правил составления выражений и их вычислений:

1) Выражения вычисляются слева направо;

2) Круглые скобки используются для смены приоритета операций;

3) Выражение должно записываться в строку без переносов;

4) Не допускается запись несколько подряд идущих арифметических операций;

5) Допускается использовать в выражении вызов функции, вычисляющей и возвращающей целочисленное значение.

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

## **1.14 Конструкции языка**

Ключевые программные конструкции языка программирования KES-2022 представлены в таблице 1.6.

Таблица 1.6 – Программные конструкции языка KES-2022

|  |  |
| --- | --- |
| Конструкция | Запись на языке KES-2022 |
| Главная функция (точка входа в приложение) | main  {…} |
| Функция | <тип данных> function <идентификатор> (<тип> <идентификатор>, ...)  {…  return <идентификатор/литерал>;  }return < идентификатор/литерал >;  } |
| Цикл | while <идентификатор1> <оператор> <идентификатор2>  do [ … ] |

## **1.15 Область видимости идентификаторов**

Область видимости переменных: сверху вниз. Переменные, объявленные в одной функции, недоступны в другой. Все объявления и операции с переменными происходят внутри программного блока. Каждая переменная, объявленная внутри блока функции получают соответствующий постфикс – название функции, внутри которой они находятся.

Объявление глобальных переменных не предусмотрено.

## **1.16 Семантические проверки**

В языке программирования KES-2022 выполняются следующие семантические проверки:

* Наличие функции main – точки входа в программу;
* Единственность точки входа;
* Переопределение идентификаторов;
* Использование идентификаторов без их объявления;
* Проверка соответствия типа функции и возвращаемого параметра;
* Правильность передаваемых в функцию параметров: количество, типы;
* Правильность строковых выражений;
* Превышение размера строковых и числовых литералов;
* Правильность составленного условия цикла/условного оператора.

## **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 и копии строковых литералов. Локальная область видимости в исходном коде определяется за счет использования правил именования идентификаторов и регулируется их постфиксами, что и обуславливает их локальность на уровне исходного кода.

## **1.18 Стандартная библиотека и её состав**

Функции стандартной библиотеки с описанием представлены в таблице 1.7. Стандартная библиотека написана на языке программирования C++.

Таблица 1.7 – Состав стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Функция | Возвращаемое значение | Принимаемые параметры | Описание |
| random | integer | integer a | Генерирует случайное число в диапазоне от -a до а и возвращает это число. |

Продолжение таблицы 1.7

|  |  |  |  |
| --- | --- | --- | --- |
| pow | integer | integer a, integer b | Возводит число a в степень b и возвращает результат. |
| length | length | string a | Возвращает длину строки a. |

Стандартная библиотека написана на языке С++, подключается к транслированному коду на этапе генерации кода. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для манипулирования выводом, недоступные конечному пользователю. Для вывода предусмотрен оператор **write** и **writeline**. Эти функции представлены в таблице 1.8.

Таблица 1.8 Дополнительные функции стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| outnum | void | int value | Функция для вывода в стандартный поток значения целочисленного идентификатора/литерала. |
| outstr | void | char\* line | Функция для вывода в стандартный поток значения строкового, символьного идентификатора/литерала. |
| outstrline | void | char\* line | Функция для вывода в стандартный поток значения строкового, символьного идентификатора/литерала и перехода на новую строку. |
| outnumline | void | int value | Функция для вывода в стандартный поток значения целочисленного идентификатора/литерала и перехода на новую строку. |

**1.19 Ввод и вывод данных**

Ввод данных не поддерживается языком программирования KES-2022.

Вывод данных осуществляется с помощью операторов **write** и **writeline**. Допускается использование операторов **write** и **writeline** с литералами и идентификаторами.

Функции, управляющие выводом данных, реализованы на языке С++ и вызываются из транслированного кода, конечному пользователю недоступны. Пользовательские команды **write** и **writeline** в транслированном коде будут заменены вызовом нужных библиотечных функций. Библиотека, содержащая нужные процедуры, подключается на этапе генерации кода.

## **1.20 Точка входа**

В языке KES-2022 каждая программа должна содержать главную функцию (точку входа) **main**, с первой инструкции которой начнётся последовательное выполнение команд программы.

## **1.21 Препроцессор**

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке KES-2022 отсутствует.

## **1.22 Соглашения о вызовах**

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

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## **1.23 Объектный код**

KES-2022 транслируется в язык ассемблера.

## **1.24 Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке KES-2022 и выявления её транслятором в текущий файл протокола выводится сообщение. Их классификация сообщений приведена в таблице 1.9.

Таблица 1.9 Классификация ошибок

|  |  |
| --- | --- |
| Номера ошибок | Характеристика |
| 0 – 200 | Системные ошибки |
| 200 – 299 | Ошибки лексического анализа |
| 300 – 399 | Ошибки семантического анализа |
| 600 – 699 | Ошибки синтаксического анализа |
| 400 – 499, 700 – 999 | Зарезервированные коды ошибок |

## **1.25 Контрольный пример**

Контрольный пример демонстрирует главные особенности языка KES-2022: его фундаментальные типы, основные структуры, функции, процедуры, использование функций статической библиотеки. Исходный код контрольного примера представлен в приложении А.

# **2. Структура транслятора**

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

Транслятор преобразует программу, написанную на языке KES-2022 в программу на языке ассемблера. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.



Рис. 2.1 – Структура транслятора языка программирования KES-2022

Лексический анализ – первая фаза трансляции. Назначением лексического анализатора является нахождение ошибок лексики языка и формирование таблицы лексем и таблицы идентификаторов.

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

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций. Входным параметром для синтаксического анализа является таблица лексем. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора.

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке KES-2022 в код на языке Ассемблера.

**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 KES-2022

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке KES-2022, имеющий расширение .txt | Не предусмотрено |
| -log:<путь к log-файлу> | Файл с расширение .log, определяет файлы, содержащие результат работы программы: | <имя\_in\_файла>.log |
| -out:<путь к out-файлу> | Файл для записи результата работы транслятора | <имя\_in\_файла>.asm |

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

Таблица с перечнем протоколов, формируемых транслятором языка KES-2022 и их назначением представлена в таблице 2.2

Таблица 2.2 – Протоколы, формируемые транслятором языка KES-2022

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл содержит таблицу лексем и таблицу идентификаторов, протокол работы синтаксического анализатора и дерево разбора, полученные на этапе лексического и синтаксического анализа, а также результат работы алгоритма преобразования выражений к польской записи. |
| Выходной файл, заданный параметром "-out:" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

# **3. Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке KES-2022. На выходе формируется таблица лексем и таблица идентификаторов.

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



Рисунок 3.1 – Структура лексического анализатора KES-2022

## **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 ниже.

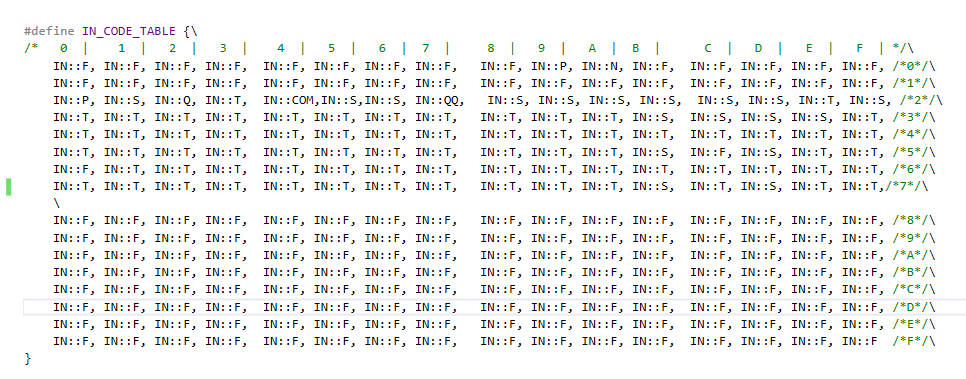


Рисунок 3.2. – Таблица контроля входных символов

Принцип работы таблицы заключается в соответствии значения каждого элемента значению в таблице ASCII.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, I – игнорируемый символ, Q – строковый литерал, QQ – символьный литерал, S – сепаратор, N –перевод строки, R – пробел(табуляция), COM – комментарии.

## **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, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов**

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие токенов и лексем приведено в таблице 3.1.

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

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| integer, string,symbol | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 8 символов. |
| Литерал | l | Литерал любого доступного типа. |
| function | f | Объявление функции. |
| Функции стандартной библиотеки | p | Ключевое слово для стандартных функций. |
| return | e | Выход из функции/процедуры. |
| main | m | Главная функция. |
| type | n | Объявление переменной. |
| write | o | Вывод данных. |
| writeline | ^ | Вывод данных c переводом строки. |
| is | ? | Указывает начало условного оператора. |
| while | c | Указывает начало цикла. |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| writeline | ^ | Вывод данных c переводом строки. |
| is | ? | Указывает начало условного оператора. |
| while | c | Указывает начало цикла. |
| true | r | Истинная ветвь условного оператора. |
| false | w | Ложная ветвь условного оператора. |
| do | d | Указывает на начало тела цикла. |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| [ | [ | Начало блока цикла/условного выражения. |
| ] | ] | Конец блока цикла/условного выражения. |
| { | { | Начало тела функции. |
| } | } | Конец тела функции |
| ( | ( | Передача параметров в функцию, приоритет операций. |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | = | Знак присваивания. |

Продолжение таблицы 3.2

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| +  -  \*  /  % | +  -  \*  /  % | Знаки операций. |
| >  <  &  ! | >  <  &  ! | Знаки логических операторов |

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 автомата изображена в приложении Б и листинге 3.4.

Пример реализации графа конечного автомата для токена string представлен в листинге 3.2.

#define GRAPH\_WRITELINE 10, \

FST::NODE(1, FST::RELATION('w', 1)), \

FST::NODE(1, FST::RELATION('r', 2)), \

FST::NODE(1, FST::RELATION('i', 3)), \

FST::NODE(1, FST::RELATION('t', 4)), \

FST::NODE(1, FST::RELATION('e', 5)), \

FST::NODE(1, FST::RELATION('l', 6)), \

FST::NODE(1, FST::RELATION('i', 7)), \

FST::NODE(1, FST::RELATION('n', 8)), \

FST::NODE(1, FST::RELATION('e', 9)), \

FST::NODE()

Листинг 3.2 - Пример реализации графа конечного автомата для writeline

## **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, номер строки в исходном коде (sn), и номер в таблице идентификаторов, если лексема является идентификаторов (idxTI). Код C++ со структурой таблицы лексем представлен в листинге 3.3

struct Entry //строка таблицы лексем

{

unsigned char lexema; //лексема

int sn; //номер строки в исходном тексте

int idxTI; //индекс в таблице идентификаторов или LT\_TI\_NULLIDX

Entry();

Entry(unsigned char lexema, int snn, int indxTI=NULLDX\_IT);

};

struct LexTable //экземпляр таблицы лексем

{

int maxsize; //емкость таблицы лексем < LT\_MAXSIZE

int size; //текущий размер таблицы лексем < maxsize

Entry\* table; //массив строк таблицы лексем

};

Листинг 3.3 - 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

struct Entry

{

int idxfirstLE;

char id[SCOPED\_ID\_MAXSIZE];

IDDATATYPE iddatatype;

IDTYPE idtype;

union

{

int vint;

struct

{

int len;

char str[STR\_MAXSIZE - 1];

} vstr;

char symbol;

struct{

int count;

IDDATATYPE \*types;

} params;

}value;

Entry(){…}

Entry(char\*id, int idxLT, IDDATATYPE datatype, IDTYPE idtype){…}

};

struct IdTable

{

int maxsize;

int size;

Entry\* table; //массив строк таблицы id

};

Листинг 3.4 - Структура таблицы идентификаторов

## **3.6 Принцип обработки ошибок**

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. При возникновении первой ошибки, дальнейшая обработка прекращается. Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами.

## **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. Если в процессе анализа находятся 1 ошибка, то анализ останавливается. Перечень сообщений представлен в таблице 3.2.

Таблица 3.2 – Перечень сообщений лексического анализатора

|  |  |
| --- | --- |
| Номер сообщения | Текст сообщения |
| 200 | Недопустимый символ в исходном файле(-in) |
| 202 | Превышен размер таблицы лексем |
| 203 | Превышен размер таблицы идентификаторов |
| 301 | Отсутствует точка входа main |
| 302 | Несколько точек входа main |

## **3.8 Параметры лексического анализатора**

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

## **3.9 Алгоритм лексического анализа**

Алгоритм работы лексического анализатора выглядит следующим образом:

* проверяет входной поток символов программы на исходном языке на допустимость, удаляет лишние пробелы и добавляет сепаратор для вычисления номера строки для каждой лексемы;
* для выделенной части входного потока выполняется функция распознавания лексемы;
* при успешном распознавании информация о выделенной лексеме заносится в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к первому этапу;
* при неуспешном распознавании выдается сообщение об ошибке.
* Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «string» представлен на рисунке 3.2, где S0 – начальное, а S6 – конечное состояние автомата.

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «string» представлен на рисунке 3.2, где S0 – начальное, а S6 – конечное состояние автомата.

s t r i n g

## **3.10 Контрольный пример**

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

# **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. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов, а выходной – дерево разбора. Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора.

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

В синтаксическом анализаторе транслятора языка KES-2022 используется контекстно-свободная грамматика , где

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

N – множество нетерминальных символов,

P – множество правил языка

S – начальный символ грамматики, являющийся нетерминалом.

Эта грамматика имеет ослабленную нормальную форму Грейбах.

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

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

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

Таблица правил переходов и описание нетерминальных символов представлена в приложении Е.

**4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку, описание которой представлено в таблице 4.2.

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ S) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

## **4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата, выполняющего разбор исходной ленты, и структуру грамматики Грейбах, описывающей правила языка KES-2022. Данные структуры представлены в приложении Г.

## **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. После 3 исключений синтаксический анализатор завершает работу.

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

Перечень сообщений синтаксического анализатора представлен в таблице

Таблица 4.2 – Перечень сообщений синтаксического анализатора

|  |  |
| --- | --- |
| Номер сообщения | Текст сообщения |
| 600 | Неверная структура программы |
| 601 | Не найден список параметров функции |
| 602 | Ошибка в теле функции |
| 603 | Ошибка в теле процедуры |
| 604 | Ошибка в списке параметров функции |
| 605 | Ошибка в вызове функции/выражении |
| 606 | Ошибка в списке фактических параметров функции |
| 607 | Неверный арифметический оператор |
| 608 | Неверное выражение. Ожидаются только идентификаторы/литералы |
| 609 | Ошибка в арифметическом выражении |
| 610 | Недопустимая синтаксическая конструкция |

Рис. 4.2 – Сообщения синтаксического анализатора

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

Входным параметром синтаксического анализатора является таблица лексем, полученная на этапе лексического анализа, а также правила контекстно-свободной грамматики в форме Грейбах.

Результатом работы синтаксического анализатора является дерево разбора.

## **4.8 Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

– синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем;

– если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка, которая записывается в протокол работы и программа останавливается.

**4.9 Контрольный пример**

Результатом работы синтаксического анализатора является дерево разбора. Дерево разбора представлено в приложении Г.

# **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 семантические правила языка (семантики языка).

## **5.3 Перечень сообщений семантического анализатора**

Сообщения, формируемые семантическим анализатором, представлены в таблице 5.1

Таблица 5.1 – Сообщения семантического анализатора

|  |  |
| --- | --- |
| Номер сообщения | Текст сообщения |
| 300 | Идентификатор не объявлен |
| 303 | Не указан тип идентификатора при объявлении |
| 304 | Отсутствует ключевое слово при объявлении |
| 305 | Попытка переопределения идентификатора |

Продолжение таблицы 5.1

|  |  |
| --- | --- |
| 306 | Превышено максимальное количество параметров функции |
| 307 | Слишком много параметров в вызове функции |
| 308 | Кол-во ожидаемых функцией и передаваемых параметров не совпадают |
| 309 | Несовпадение типов передаваемых параметров |
| 310 | Использование пустого строкового литерала недопустимо |
| 311 | Обнаружен символ \'\"\'. Возможно, не закрыт строковый литерал |
| 312 | Превышен размер строкового литерала |
| 313 | Недопустимый целочисленный литерал |
| 314 | Типы данных в выражении не совпадают |
| 315 | Тип функции и возвращаемого значения не совпадают |
| 316 | Недопустимое строковое выражение после \'=\' |
| 317 | Не закрыт блок main |

## **5.4 Принцип обработки ошибок**

В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением. Анализ останавливается после нахождения 1 ошибки.

## **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{  integer x = 9;  write x;  } | Ошибка N304: Семантическая ошибка: В объявлении отсутствует ключевое слово type Строка: 2 |
| main{  type integer x = 9;  type integer y = x; } | Ошибка N314: Семантическая ошибка: Типы данных в выражении не совпадают Строка: 3 |
| main{  type integer x = 9;  }  main{  type string y = "qwerty";} | Ошибка N302: Семантическая ошибка: Обнаружено несколько точек входа main |

## **6. Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке KES-2022 допускаются вычисления выражений целочисленного типа данных с поддержкой вызова функций внутри выражений. Приоритет операций представлен на таблице 6.1.

Таблица 6.1 – Приоритеты операций

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

## **6.2 Польская запись**

Все выражения языка KES-2022 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 представлен ниже.

1. исходная строка: выражение;
2. результирующая строка: польская запись;
3. стек: пустой;
4. исходная строка просматривается слева направо;
5. операнды переносятся в результирующую строку;
6. операция записывается в стек, если стек пуст;
7. операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
8. отрывающая скобка помещается в стек;
9. 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 Д.

## **6.4 Контрольный пример**

Пример преобразования выражений из контрольного примера к обратной польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления и преобразования к ассемблерному коду. В приложении Д приведена изменённая таблица лексем, отображающая результат преобразования выражений в польский формат.

Таблица 6.2. – Преобразование выражений к ПОЛИЗ

|  |  |
| --- | --- |
| Выражение | Обратная польская запись для выражения |
| i[2]=(((l[3]+l[4])-i[0])\*l[5])/l[6]; | i[2]=l[3]l[4]+i[0]-l[5]\*l[6]/ |
| i[23]=(i[23]+l[26])\*l[26] | i[23]=i[23]l[26]+l[26]\* |
| i[3]=(((l[4]+l[5])-i[0])\*l[6]) | i[3]=l[4]l[5]+i[0]-l[6]\* |

# **7. Генерация кода**

## **7.1 Структура генератора кода**

В языке KES-2022 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 KES-2022 представлена на рисунке 7.1.

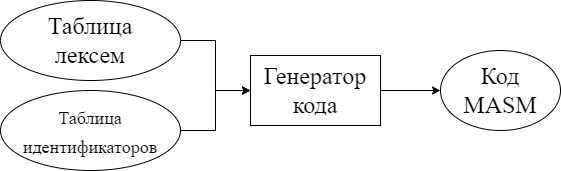


Рисунок 7.1 Структура генератора кода

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

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Соответствия между типами данных идентификаторов на языке KES-2022 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка KES-2022 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке S-2022 | Тип идентификатора на языке ассемблера | Пояснение |
| integer | sdword | Хранит целочисленный тип данных. |
| string,simbol | dword | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |

## **7.3 Статическая библиотека**

В языке KES-2022 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++.

Объявление функций статической библиотеки генерируется автоматически в коде ассемблера. Список функций и их описание приведены в разделе 1.18.

## **7.4 Особенности алгоритма генерации кода**

В языке KES-2022 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2

Рис. 7.2 – Структура генератора кода

## **7.5 Входные параметры, управляющие генерацией кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного кода программы на языке KES-2022. Результаты генератора кода выводятся в файл с расширением .asm.

## **7.6 Контрольный пример**

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Ж. Результат работы контрольного примера приведён на рисунке 7.3.

|  |
| --- |
| String: Don’t give up  Length of string:12  Factorial:479001600  Random number: -10  New random number: 10  Number of digit in factorial: 9 |

Рисунок 7.3 – Результат работы контрольного примера

# **8. Тестирование транслятора**

## **8.1 Общие положения**

При возникновении ошибки на каком-либо этапе трансляции, она обрабатывается в главном файле программы: ошибка выводится на консоль и записывается в файл логирования. Если ошибок обнаружено не было, текст консоли окрасится в зеленый цвет и будет выведена надпись об успешной компиляции файла

## **8.2 Результаты тестирования**

В языке KES-2022 не разрешается использовать запрещённые входным алфавитом символы. Результаты тестирования языка KES-2022 представлены в таблице 8.1

Таблица 8.1 – Тестирование проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main | Ошибка 200: Лексическая ошибка: Недопустимый символ в исходном файле(-in) Строка: 1 |
| main{  type integer i = a;  } | Ошибка 300: Семантическая ошибка: Необъявленный идентификатор Строка: 2 |
| integer function a(){  return 1;  } | Ошибка 301: Семантическая ошибка: Отсутствует точка входа main |
| main{  integer i = 3;  } | Ошибка 304: Семантическая ошибка: В объявлении отсутствует ключевое слово type Строка: 2 |
| main{type integer i;  type integer i;} | Ошибка 305: Семантическая ошибка: Попытка переопределения идентификатора Строка: 3 |
| integer function func(integer a)  {return 1;}  main{  type integer i = func(1,2);} | Ошибка 308: Семантическая ошибка: Кол-во ожидаемых функцией и передаваемых параметров не совпадают Строка: 4 |
| integer function func(integer a)  {return 1;}  main{  type integer i = func("string");  } | Ошибка 309: Семантическая ошибка: Несовпадение типов передаваемых параметров Строка: 4 |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| main{type string i = "sdj;  } | Ошибка 311: Семантическая ошибка: Обнаружен символ '"'. Возможно, не закрыт строковый литерал Строка: 2 |
| main{  type integer i = ~3~+3;} | Ошибка 314: Семантическая ошибка: Типы данных в выражении не совпадают Строка: 2 |
| integer function out(){  return; }  main{  out();} | Ошибка 315: Семантическая ошибка: Тип функции и возвращаемого значения не совпадают Строка: 2 |
| main{  is ~3~ < 4  true[write 3;]  } | Ошибка 317: Семантическая ошибка: Неверное условное выражение Строка: 1 |
| main{  type integer s = 1/0;  } | Ошибка 318: Семантическая ошибка: Деление на ноль Строка: 2 |
| main{  is 1 < 2  true[return 1;]  } | Ошибка 320: Семантическая ошибка: Недопустим return в main Строка: 3 |
| main type integer x; } | Ошибка 600: строка 1, Синтаксическая ошибка: Неверная структура программы |
| string function fi({}  main{  } | Ошибка 601: строка 1, Синтаксическая ошибка: Не найден список параметров функции |
| integer function func(){  return 1;  }  main{  type integer s = func(]);  } | Ошибка 605: строка 5, Синтаксическая ошибка: Ошибка в списке фактических параметров функции |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| main{  type integer s = 1 ++2;  } | Ошибка 610: строка 2, Синтаксическая ошибка: Неверный арифметический оператор |
| main{  type integer s = 1]2;  } | Ошибка 611: строка 2, Синтаксическая ошибка: Ошибка в арифметическом выражении |
| main{ type integer s  } | Ошибка 612: строка 2, Синтаксическая ошибка: Недопустимая синтаксическая конструкция |
| main{  while 1 < 2  do[return;]} | Ошибка 613: строка 3, Синтаксическая ошибка: Недопустимая синтаксическая конструкция в теле цикла/условного выражения |

## 

## **Заключение**

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования KES-2022 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка KES-2022;
2. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
3. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
4. Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
5. Осуществлена программная реализация синтаксического анализатора;
6. Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;

7) Разработан транслятор кода на язык ассемблера;

8) Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка KES-2022 включает:

1. 3 типа данных;
2. Поддержка операторов вывода ;
3. Возможность вызова функций стандартной библиотеки;
4. Наличие 5 арифметических операторов для вычисления выражений;
5. Поддержка функций, процедур, 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.

## **Литература**

1) Курс лекций по предмету «Конструирование программного обеспечения» Наркевич А.С.

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

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

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

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

## **Приложение А**

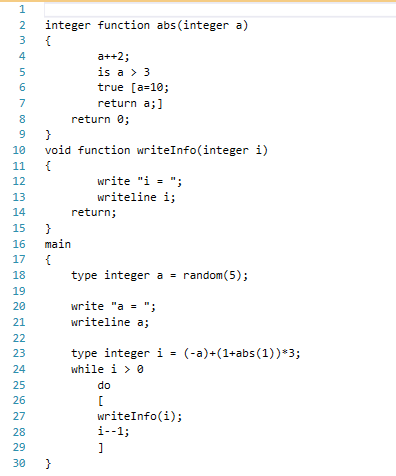


Рисунок 1- Исходный код программы на языке KES-2022

## **Приложение Б**

namespace FST

{

struct RELATION

{

char symbol;

short nnode;

RELATION(

char c = 0x00,

short ns = 0

);

};

struct NODE

{

short n\_relation;

RELATION\* relations;

NODE();

NODE(

short n,

RELATION rel, ...

);

};

struct FST

{

unsigned char\* string;

short position;

short nstates;

NODE\* nodes;

short\* rstates;

FST(short ns,NODE n, …);

FST(unsigned char\* 5, FST&fst);

FST(

unsigned char\*s,

short ns,

NODE n, …

);

};

}

Листинг 1 – Конечный автомат

## **Приложение В**

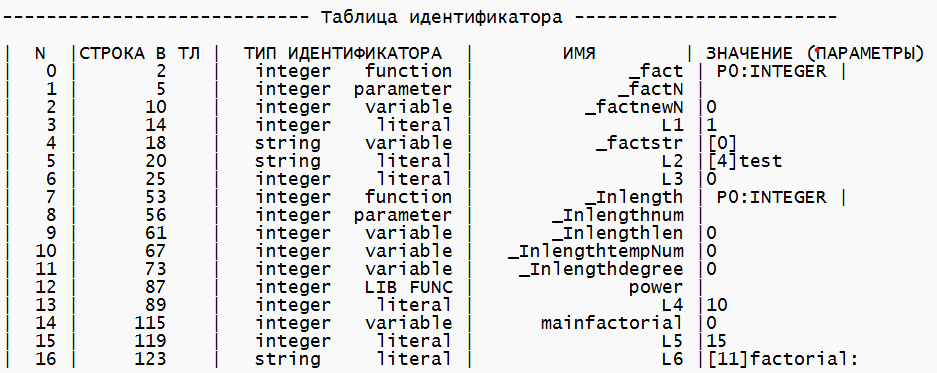


Рисунок 1 - Таблица идентификаторов

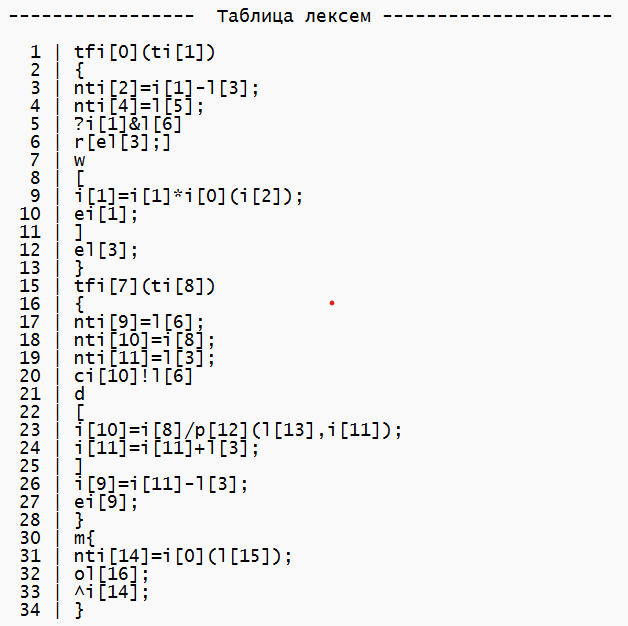
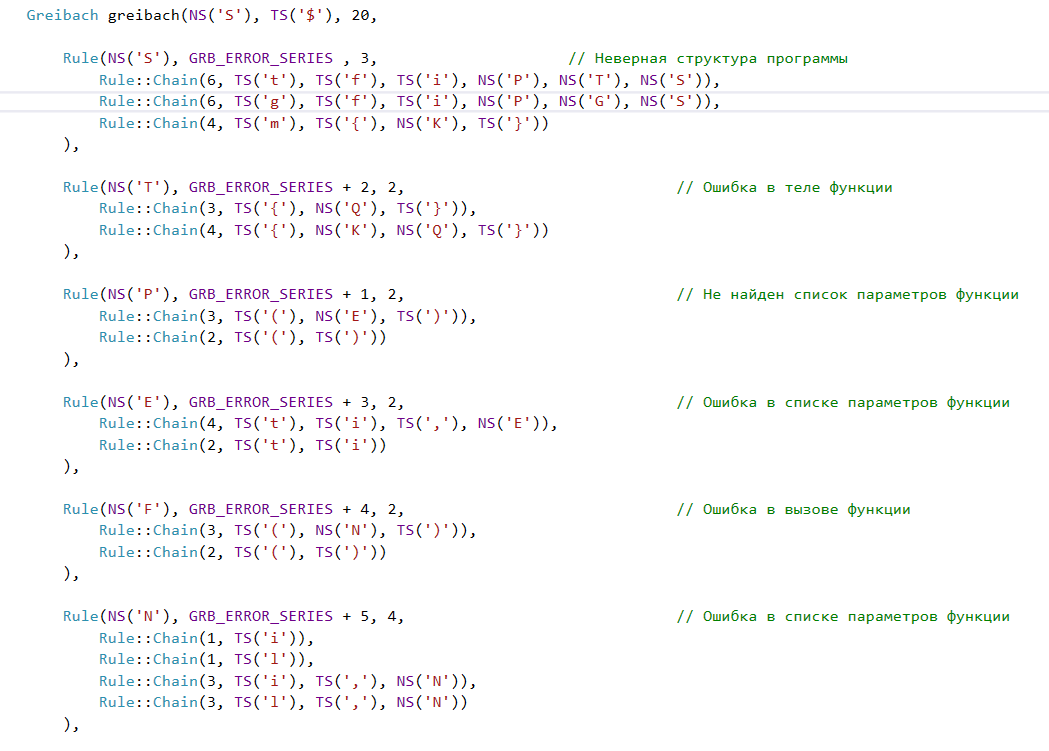
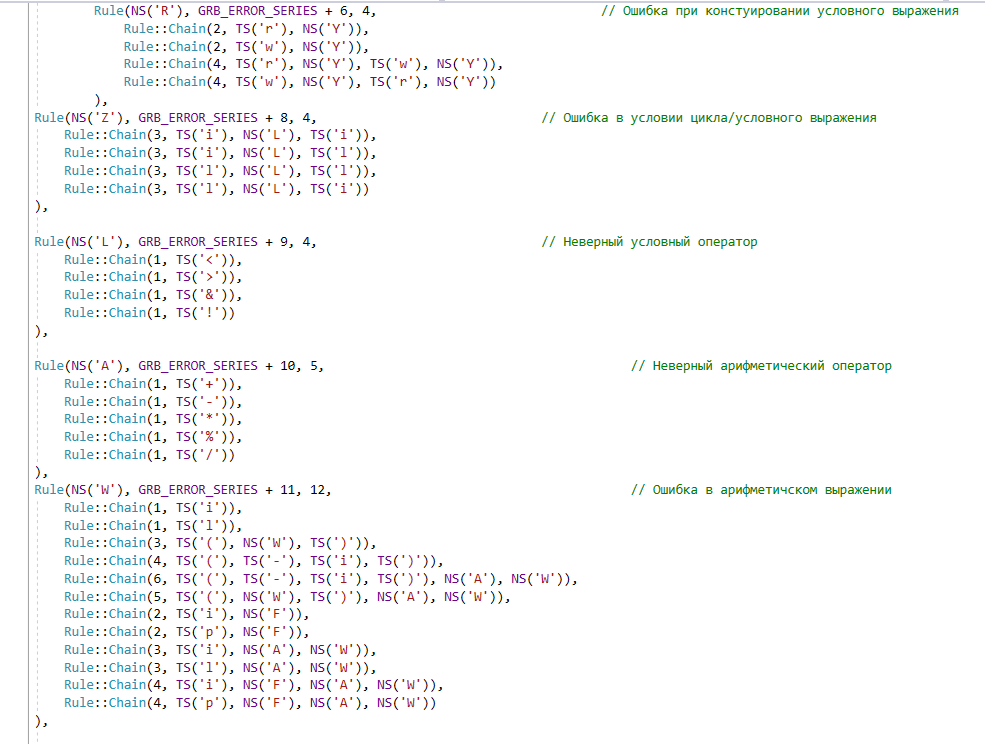
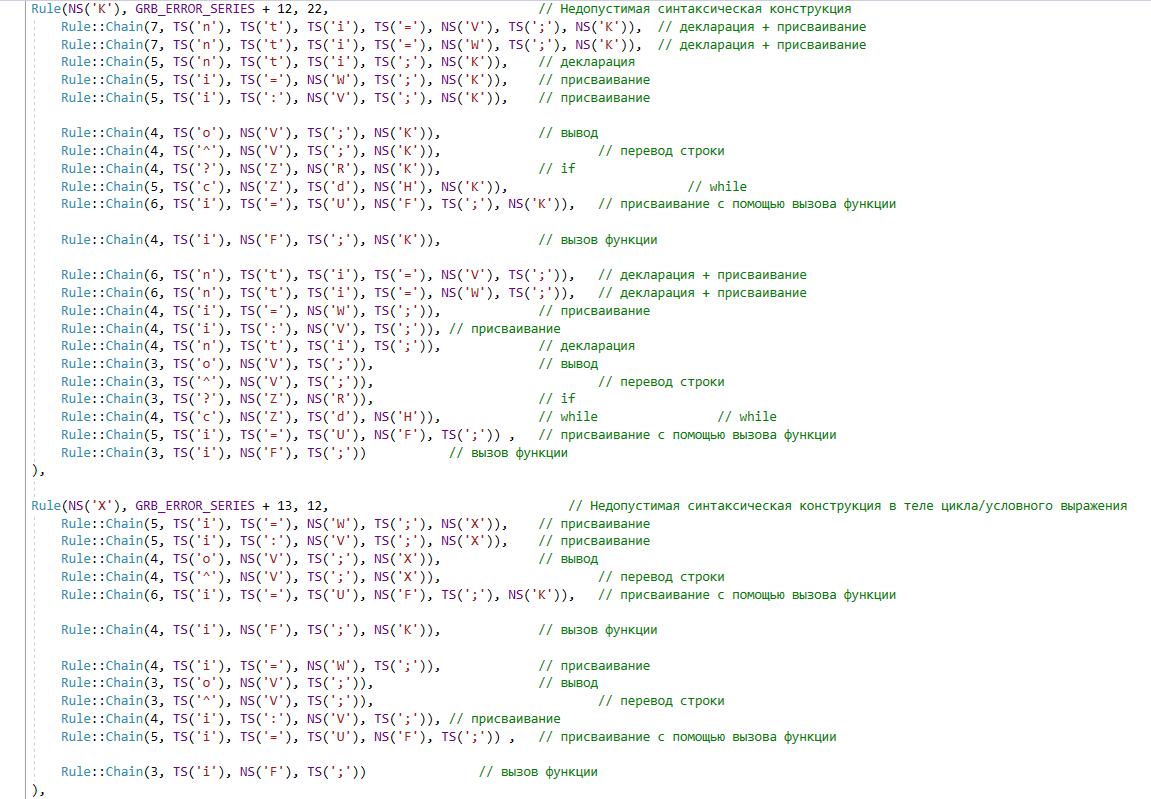


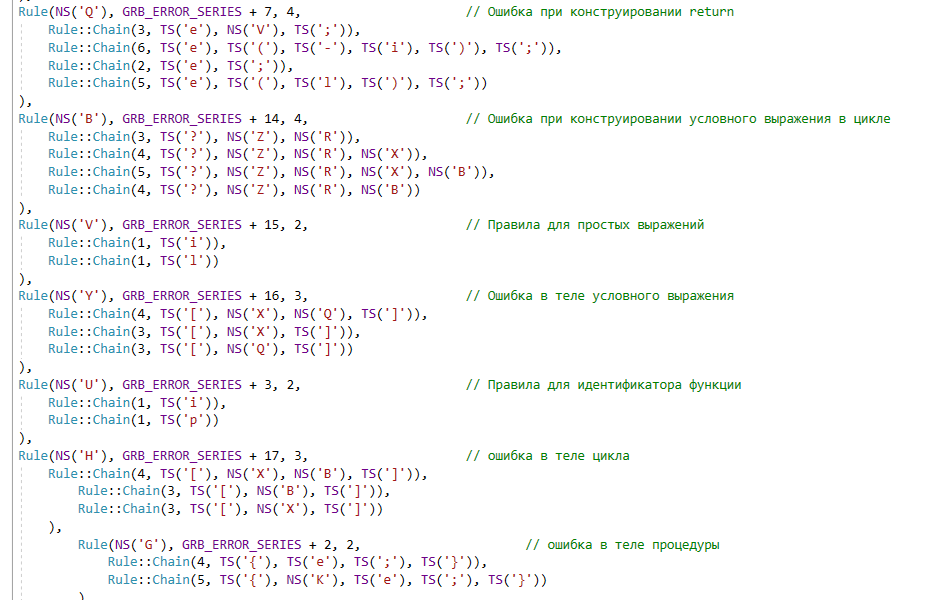
Рисунок 2 –Таблица лексем

## **Приложение Г**

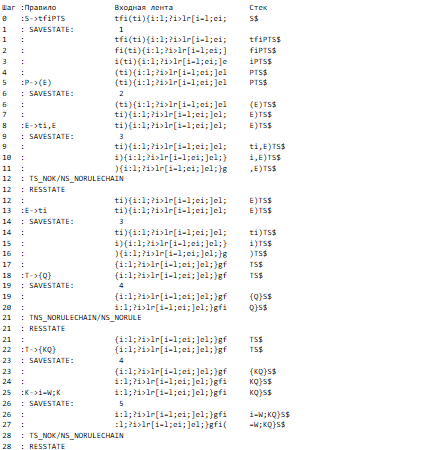


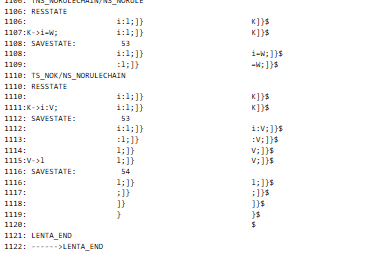




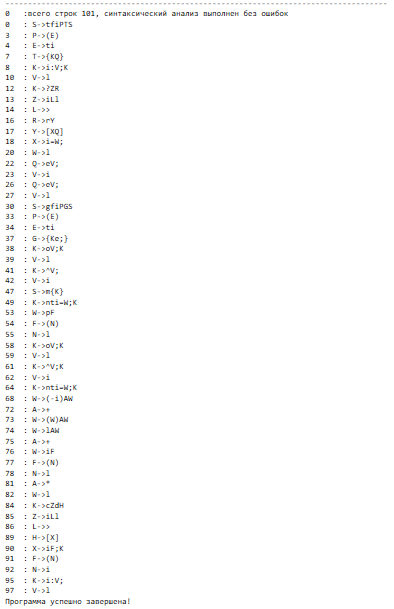


Листинг 1 – Структура грамматики Грейбах



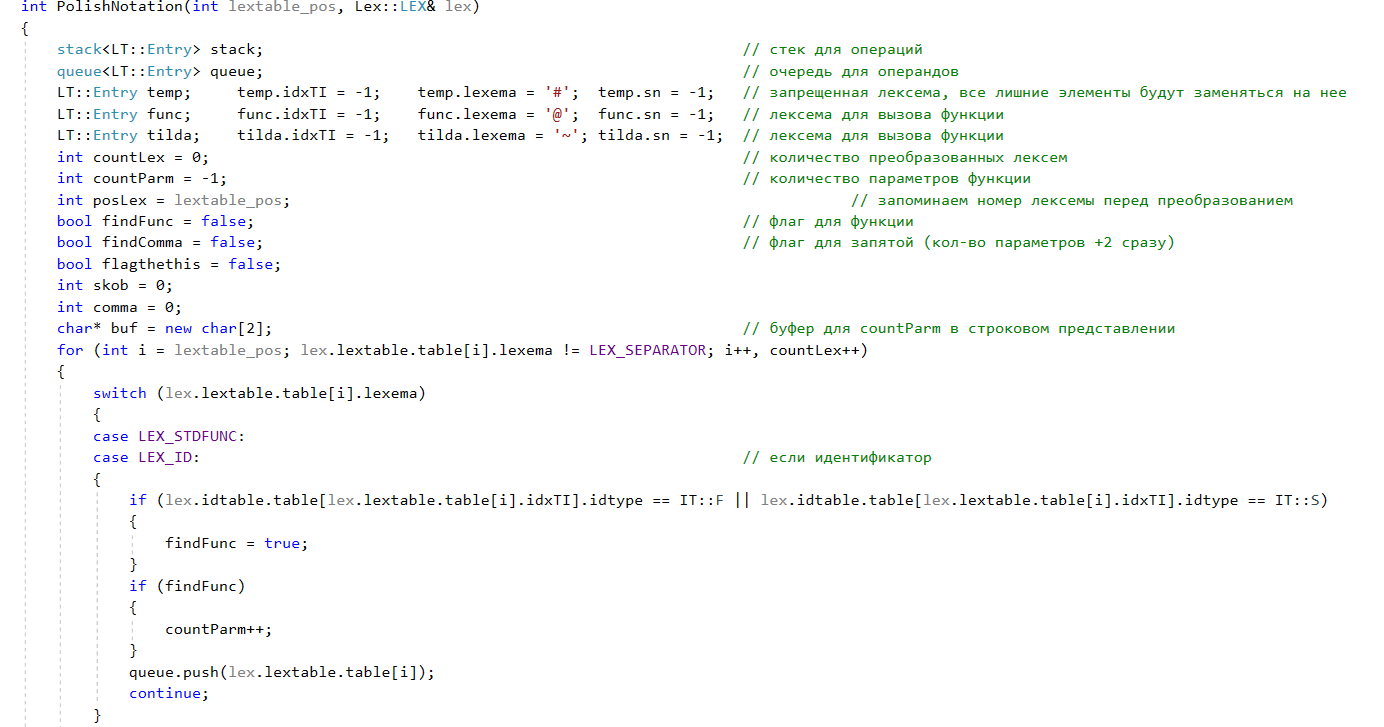


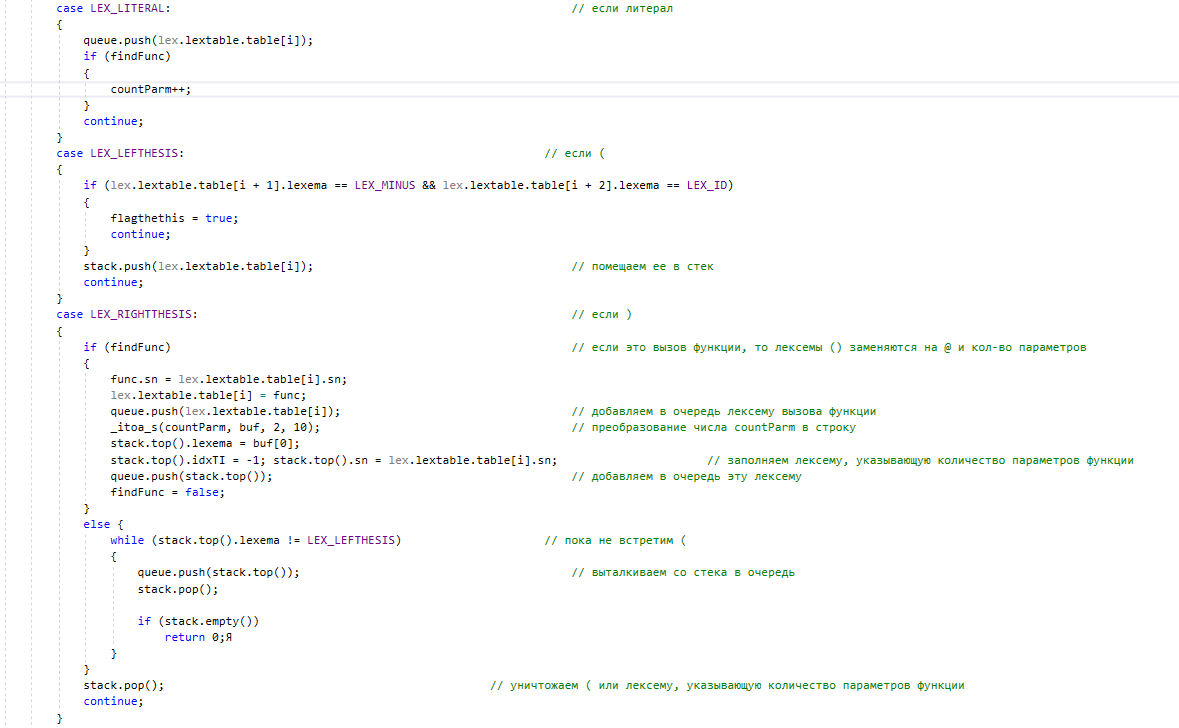
Листинг 2 – Разбор исходного кода синтаксическим анализатором

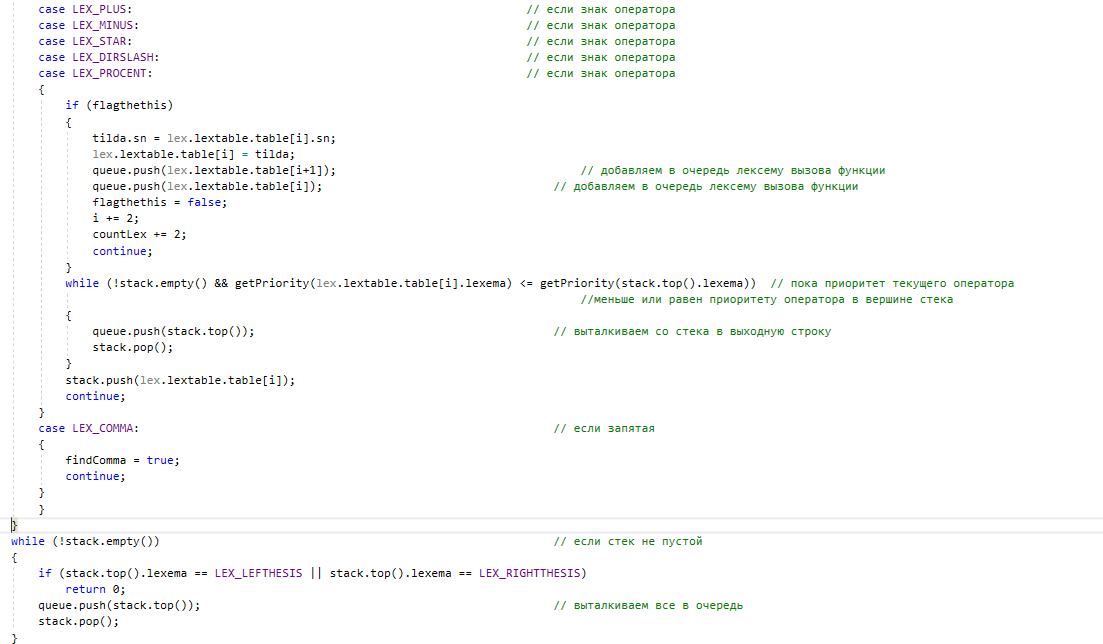


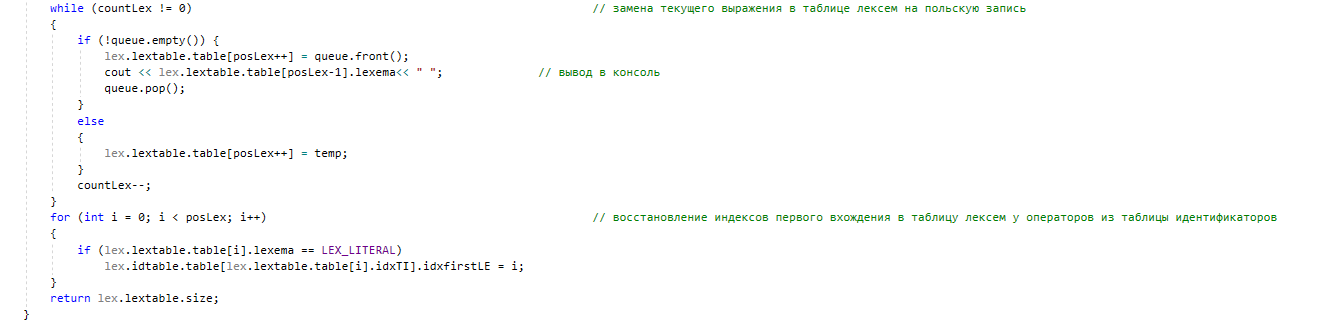
Листинг 3 – Дерево разбора

## **Приложение Д**







Листинг 4 – Реализация преобразования выражений к обратной польской записи

## 

## **Приложение Е**

Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| S | S->tfiPTS  S->m{K} | Стартовые правила, описывающее общую структуру программы |
| P | P->(E)  P->() | Правила для параметров объявляемых функций |
| T | T->{Q;}  T->[KQ;] | Правила для тела функций |
| Q | Q->eV;  Q->e(l);  Q->e(-i); | Правила для возврата значений |
| E | E->ti,E  E->ti | Правила для списка параметров функции |
| F | F->(N)  F->() | Правила для вывозов функций(в т.ч. и в выражениях) |
| N | N->i  N>l  N->l,N  N->i,N | Правила для параметров вызываемых функций |
| R | R->rY  R->wY  R->rYwY  R->wYrY | Правила составления условного оператора |
| Z | Z->iLi  Z->iLl  Z->lli | Правила для условия цикла/условного оператора |
| L | L-><  L->>  L->&  L->! | Правила для логических операторов |
| A | A->+  A->-  A->\*  A->/  A->% | Правила для арифметических операторов |
| V | V->l  V->i | Правила для простых выражений |

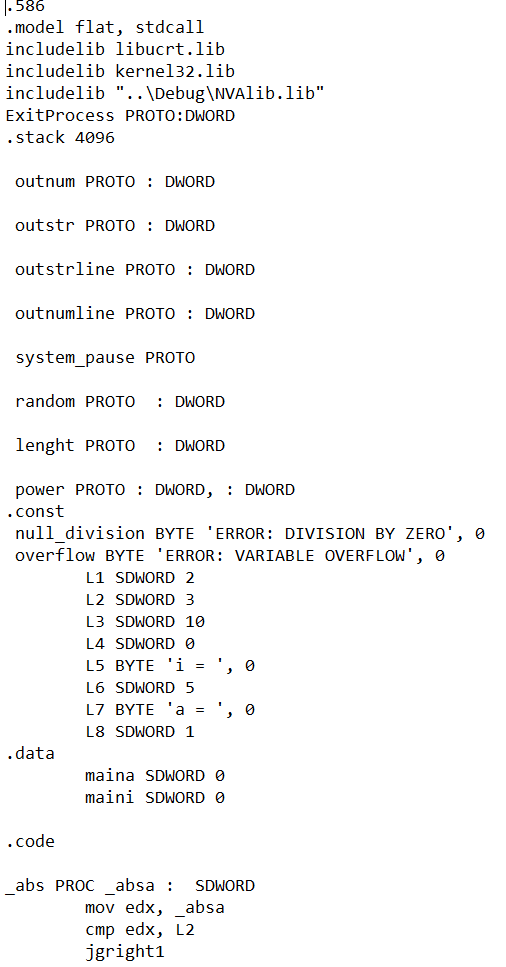
## Продолжение таблицы 1

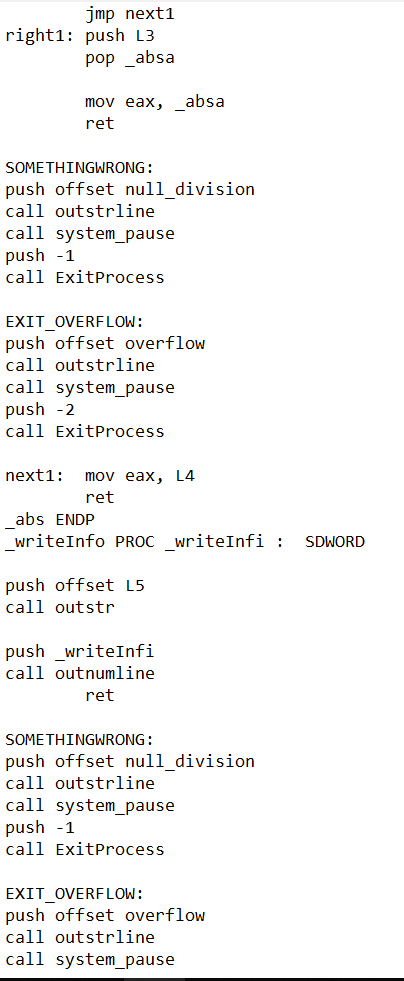
|  |  |  |
| --- | --- | --- |
| Y | Y->[XQ]  Y->[X]  Y->[Q] | Правила для тела условного выражения |
| W | W->l  W->i  W->(W)  W->(-i)  W->(-i)AW  W->(W)AW  W->iF  W->pF  W->iAW  W->lAW  W->iFAW  W->pFAW | Правила для сложных выражений |
| K | K->nti=V;K  K->nti=WK  K->nti;K  K->i=W;K  K->oV;K  K->^VZ;K  K->?ZdHK  K->cZdHK  K->i=UF;K  K->nti=V;  K->nti=W;  K->i=W;  K->nti;  K->oV;  K->^V;  K->?ZR  K->cZdH  K->i=UF; | Программные конструкции |
| X | X->i=W;X  X->oV;X  X->^V;X  X->i=UF; | Программные конструкции внутри цикла/условного оператора |

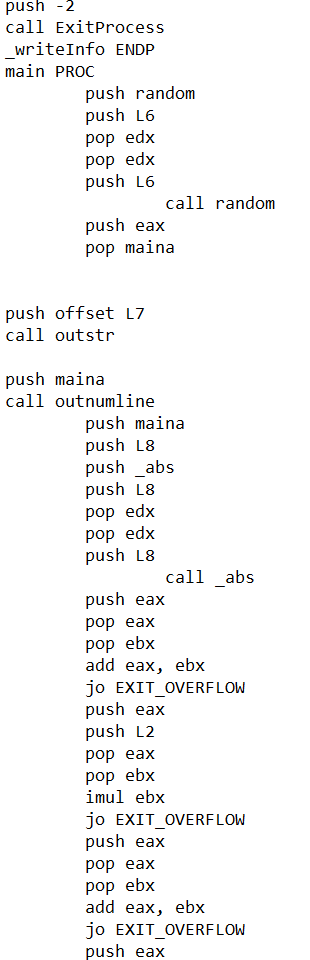
## Продолжение таблицы 1

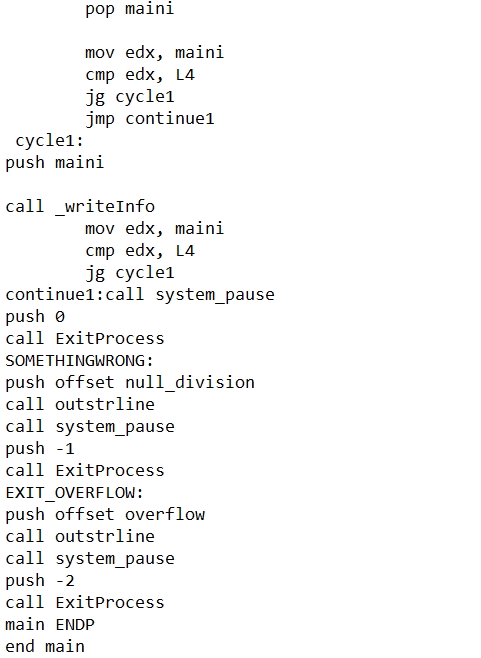
|  |  |  |
| --- | --- | --- |
| U | U->i  U->p | Правила для идентификатора функции |
| B | B->?ZR  B->?ZRX  B->?ZRXB  B->?ZRB | Правила для конструировании условного выражения в цикле |
| H | H->[XB]  H->[B]  H->[X] | Правила для тела цикла |

## **Приложение Ж**









Листинг 5 – Результат генерации ассемблерного кода