МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ

ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

Специальность 6-05-0612-01 Программная инженерия

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора LPA-2024»

Выполнил студент Ленкевич Павел Андреевич

(Ф.И.О.)

Руководитель проекта Волчек Дарья Ивановна

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

Консультанты Волчек Дарья Ивановна

Нормоконтролер Волчек Дарья Ивановна

Курсовой проект защищен с оценкой

Минск 2024

**Содержание**

[Введение 5](#_Toc185135540)

[1. Спецификация языка программирования 6](#_Toc185135541)

[1.1 Характеристика языка программирования 6](#_Toc185135542)

[1.2 Определение алфавита языка программирования 6](#_Toc185135543)

[1.3 Применяемые сепараторы 6](#_Toc185135544)

[1.4 Применяемые кодировки 7](#_Toc185135545)

[1.5 Типы данных 7](#_Toc185135546)

[1.6 Преобразование типов данных 8](#_Toc185135547)

[1.7 Идентификаторы 8](#_Toc185135548)

[1.8 Литералы 8](#_Toc185135549)

[1.9 Объявление данных 9](#_Toc185135550)

[1.10 Инициализация данных 10](#_Toc185135551)

[1.11 Инструкции языка 10](#_Toc185135552)

[1.12 Операции языка 11](#_Toc185135553)

[1.13 Выражения и их вычисления 12](#_Toc185135554)

[1.14 Конструкции языка 12](#_Toc185135555)

[1.15 Область видимости идентификаторов 13](#_Toc185135556)

[1.16 Семантические проверки 13](#_Toc185135557)

[1.17 Распределение оперативной памяти на этапе выполнения 13](#_Toc185135558)

[1.18 Стандартная библиотека и её состав 13](#_Toc185135559)

[1.19 Ввод и вывод данных 14](#_Toc185135560)

[1.20 Точка входа 14](#_Toc185135561)

[1.21 Препроцессор 14](#_Toc185135562)

[1.22 Соглашение о вызовах 14](#_Toc185135563)

[1.23 Объектный код 14](#_Toc185135564)

[1.24 Классификация сообщений транслятора 14](#_Toc185135565)

[1.25 Контрольный пример 15](#_Toc185135566)

[2. Спецификация языка программирования 16](#_Toc185135567)

[2.1 Компоненты транслятора их назначение и принципы взаимодействия 16](#_Toc185135568)

[2.2 Перечень входных параметров транслятора 17](#_Toc185135569)

[2.3 Протоколы, формируемые транслятором 17](#_Toc185135570)

[3. Разработка лексического анализатора 18](#_Toc185135571)

[3.1 Структура лексического анализатора 18](#_Toc185135572)

[3.2 Контроль входных символов 18](#_Toc185135573)

[3.3 Удаление избыточных символов 19](#_Toc185135574)

[3.4 Перечень ключевых слов 19](#_Toc185135575)

[3.5 Основные структуры данных 20](#_Toc185135576)

[3.6 Структура и перечень сообщений лексического анализатора 22](#_Toc185135577)

[3.7 Принцип обработки ошибок 23](#_Toc185135578)

[3.8 Параметры лексического анализатора 23](#_Toc185135579)

[3.9 Алгоритм лексического анализа 23](#_Toc185135580)

[3.10 Контрольный пример 24](#_Toc185135581)

[4. Разработка синтаксического анализатора 25](#_Toc185135582)

[4.1 Структура синтаксического анализатора 25](#_Toc185135583)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 25](#_Toc185135584)

[4.3 Построение конечного магазинного автомата 28](#_Toc185135585)

[4.4 Основные структуры данных 29](#_Toc185135586)

[4.5 Описание алгоритма синтаксического разбора 29](#_Toc185135587)

[4.6 Структура и перечень сообщений синтаксического анализатора 29](#_Toc185135588)

[4.7 Параметры синтаксического анализатора и режимы его работы 30](#_Toc185135589)

[4.8 Принцип обработки ошибок 30](#_Toc185135590)

[4.9 Контрольный пример 30](#_Toc185135591)

[5. Разработка семантического анализатора 31](#_Toc185135592)

[5.1 Структура семантического анализатора 31](#_Toc185135593)

[5.2 Функции семантического анализатора 31](#_Toc185135594)

[5.3 Структура и перечень сообщений семантического анализатора 32](#_Toc185135595)

[5.4 Принцип обработки ошибок 32](#_Toc185135596)

[5.5 Принцип обработки ошибок 33](#_Toc185135597)

[6. Вычисление выражений 34](#_Toc185135598)

[6.1 Выражения, допускаемые языком 34](#_Toc185135599)

[6.2 Польская запись и принцип её построения 34](#_Toc185135600)

[6.3 Программная реализация обработки выражений 35](#_Toc185135601)

[6.4 Контрольный пример 36](#_Toc185135602)

[7. Генерация кода 37](#_Toc185135603)

[7.1 Структура генератора кода 37](#_Toc185135604)

[7.2 Представление типов данных в оперативной памяти 37](#_Toc185135605)

[7.3 Статическая библиотека 38](#_Toc185135606)

[7.4 Особенности алгоритма генерации кода 38](#_Toc185135607)

[7.5 Входные параметры, управляющие генерацией кода 39](#_Toc185135608)

[7.6 Входные параметры, управляющие генерацией кода 39](#_Toc185135609)

[8. Тестирование транслятора 40](#_Toc185135610)

[8.1 Общие положения 40](#_Toc185135611)

[8.2 Результаты тестирования 40](#_Toc185135612)

[Заключение 43](#_Toc185135613)

[Приложение А 45](#_Toc185135614)

[Приложение Б 49](#_Toc185135615)

[Приложение В 62](#_Toc185135616)

[Приложение Г 71](#_Toc185135617)

[Приложение Д 72](#_Toc185135618)

[Приложение Е 74](#_Toc185135619)

[Приложение Ж 75](#_Toc185135620)

# **Введение**

Целью курсового проекта по дисциплине «Конструирование программного обеспечения» является написание спецификации и разработка компилятора для собственного языка программирования LPA-2024, ориентированного на выполнение простейших арифметических операций в консольном режиме. Компиляция кода языка LPA-2024 будет производиться в язык ассемблер.

Этапы разработки компилятора для языка LPA-2024:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка выражений;

– генерация кода на язык Assembler;

– тестирование транслятора.

Информация о каждом этапе разработки компилятора приведена в соответствующих разделах пояснительной записки.

В первом разделе приведена спецификация языка программирования.

Во втором разделе описана структура транслятора.

В третьем разделе описаны принцип работ и этапы разработки лексического анализатора, определены разрешенные символы и ключевые слова языка программирования.

В четвертом разделе описан принцип работы синтаксического анализатора.

В пятом разделе описаны принцип работы и основные функции семантического анализатора.

В шестом разделе описаны выражения, допускаемые языком, форма, принципы построения и вычисления выражений.

В седьмом разделе описан процесс генерации кода.

В восьмом разделе приведены примеры тестирования транслятора.

Язык программирования LPA-2024 предназначен для работы с консолью, выполнения простейших арифметических операций.

# **Спецификация языка программирования**

## **Характеристика языка программирования**

LPA-2024 – это компилируемый, строго типизированный, универсальный язык программирования, ориентированный на выполнение простейших арифметических операций в консольном режиме.

## **Определение алфавита языка программирования**

Алфавит языка LPA-2024 основан на кодировке Windows-1251, изображённой на рисунке 1.1.



Рисунок 1.1 – Кодировка Windows-1251

## **Применяемые сепараторы**

Символы-сепараторы служат в качестве разделителей цепочек языка во время обработки исходного текста программы с целью разделения на токены.

Сепараторы, применяемые в языке LPA-2024, приведены в таблице 1.1.

Таблица 1.1 Символы-сепараторы

|  |  |
| --- | --- |
| Разделители | Назначение |
| «пробел», «табуляция», «переход на новую строку» | Разделяют входные лексемы |
| +, -, \*, /, % | Арифметические операторы. Используются в арифметических операторах |
| = | Оператор присваивания. Используется для присваивания значения переменной |
| <, >, <=, >=, !=, == | Условные операторы. Используются в условии цикла |
| ( ) | Блок параметров функции, так же указывает на приоритет в арифметических операциях |
| , | Разделяет параметры функции |
| { } | Ограничивает программные конструкции |
| ; | Признак конца инструкции языка |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования LPA-2024 используется кодировка Windows-1251.

## **1.5 Типы данных**

В языке LPA-2024 поддерживается 4 типа данных: целочисленный (2 байта), символьный, строковый и логический. Пользовательские типы данных не поддерживается. Описание типов данных представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка программирования LPA-2024

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Целочисленный  (int) | В памяти занимает 2 байта.  Максимальное значение: 32767.  Минимальное значение: -32768.  Значение по умолчанию: 0.  Операции: сложение (+), вычитание (-), умножение (\*), деление нацело (/), остаток от деления (%), присваивание (=), а также условия для операторов цикла: больше (>), меньше (<), проверка на неравенство (!). |
| Символьный (char) | В памяти занимает 1 байт.  Значение по умолчанию: нулевое значение. |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Строковый  (str) | В памяти занимает n + 1 байт, где n – количество символов + символ конца строки.  Максимальное количество символов в строке: 256.  Значение по умолчанию: строка нулевой длины «».  Операции: присваивание идентификатору значения другого строкового идентификатора, строкового литерала или значение строковой функции, использование библиотечных функций копирования строк и вычисления длины строки. |
| Логический (bool) | В памяти занимает 1 байт.  Принимает одно из двух значений: истинна (true) или ложь (false).  В зависимости от значения 1 бит числа установлен true: 1 или false: 0.  Логический тип необходим для работы условных операций в языке LPA-2024 |

## **Преобразование типов данных**

В языке LPA-2024 предусмотрено преобразование строкового (str) типа данных в целочисленный (int) при помощи библиотечной функции strtransint.

## **Идентификаторы**

Идентификаторы используются для именования переменных, функций. Идентификатор должен начинаться с строчной буквы латинского алфавита. После первого символа допускается использование строчных букв латинского алфавита и цифр. Идентификаторы не могут совпадать с ключевыми словами языка.

Пример корректных идентификаторов: yasha1, y002asha, yasha2.

Пример некорректных идентификаторов: 1yasha (начинается с цифры), Y2asha (содержит прописную букву), #yaSha2 (содержит недопустимый символ и прописную букву).

## **Литералы**

Литерал представляет собой запись в исходном коде программы, которая обозначает фиксированное значение. В языке программирования LPA-2024 поддерживается три типа литералов: целочисленные, строковые и символьные. Целочисленные литералы могут быть представлены в двух системах счисления: восьмеричной и десятеричной.

Описание литералов приведено в таблице 1.3.

Таблица 1.3 – Литералы языка LPA-2024

|  |  |
| --- | --- |
| Тип литерала | Характеристика |
| Целочисленный | Восьмеричный:  Последовательность восьмеричных цифр 0..7 с предшествующим знаком минус или без него, в конце которой стоит символ «O» (признак восьмеричного целочисленного литерала).  Десятичный: Последовательность десятичных цифр 0..9 с предшествующим знаком минус или без него.  Допустимый диапазон значений:  От -32768 до 32767 в десятичной системе исчисления. |
| Строковый | Набор, состоящий из символов русского и латинского алфавитов, десятичных цифр и специальных символов, заключённый в двойные кавычки. Допустимый диапазон значений:  От 0 до 255 символов. |
| Символьный | Один символ из списка русского и латинского алфавитов, специальных символов, цифр, заключённый в одинарные кавычки.  Может содержать только 1 символ. |

Примеры корректных литералов: “str154str”, ‘g’, -56O, 56, -56, 7O.

Примеры некорректных литералов: str154str (не заключён в кавычки), o56 (недопустимый символ), 9O и -8O (восьмеричный литерал может принимать только цифры от 0 до 7), s (не заключён в одинарные кавычки).

## **Объявление данных**

В языке LPA-2024 для объявления переменных используются ключевое слово new, после которого указывается тип данных переменной, а после имя идентификатора. Ключевое слово int обозначает целочисленные переменные, str – строковые, char – символьные и bool – логические. После ключевого слова указывается имя идентификатора. Инициализация переменной допустима прямо в объявлении.

new int <имя идентификатора>;

new int <имя идентификатора> = <литерал>;

new str <имя идентификатора>;

new str <имя идентификатора> = <литерал>;

new char <имя идентификатора>;

new char <имя идентификатора> = <литерал>;

new bool <имя идентификатора>;

new bool <имя идентификатора> = <литерал>;

Переменные могут быть объявлены в блоке main, в блоке функции или внутри цикла. Область видимости идентификаторов определяется границами блока кода, заключённого в фигурные скобки { }.

## **1.10 Инициализация данных**

В языке программирования LPA-2024 поддерживаются два способа инициализации переменных:

– Инициализация в месте объявления – задаётся непосредственно при объявлении переменной:

new int <имя идентификатора> = <литерал>;

new str <имя идентификатора> = <литерал>;

new char <имя идентификатора> = <литерал>;

new bool <имя идентификатора> = <литерал>;

– Инициализация после объявления – выполняется присваиванием значения уже объявленной переменной:

<имя идентификатора> = <литерал>;

При объявлении переменных без явного указания значения инициализация выполняется автоматически значениями по умолчанию:

* для типа int – 0;
* для типа str – пустая строка «»;
* для типа char – пустой символ «»;
* для типа bool – false;

Примеры инициализации в месте объявления:

* int count = 11;
* str name = “LPA-2024”;
* char simvol = ‘a’;
* bool logika = true;

Примеры инициализации после объявления:

* int total;

total = 33;

* str username;

username = “Kapetto”;

* char idclass;

idclass = ‘a’;

* bool pravda;

pravda = true;

## **1.11 Инструкции языка**

Инструкции языка LPA-2024 приведены в таблице 1.4.

Таблица 1.4 – Инструкции языка LPA-2024

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Объявление переменной | new <тип данных> <имя идентификатора>; |
| Объявление переменной с явной инициализацией | new <тип данных> <имя идентификатора> = <значение>; |
| Присваивание | <имя идентификатора> = <значение>; |
| Объявление функции | <тип данных> function <имя идентификатора> (<тип данных> <имя идентификатора>, …) {  /тело функции/  return <имя идентификатора/литерал>;  }; |
| Возврат из функции | Для функций, возвращающих значение:  return <имя идентификатора/литерал>; |
| Вызов функции | <имя идентификатора> (<имя идентификатора>, …); |
| Вывод данных | write <литерал/имя идентификатора>;  writeline <литерал/имя идентификатора>; |
| Оператор цикла | cycle (<имя идентификатора переменной/литерал> <условный оператор> <имя идентификатора переменной/литерал>)  {  /тело цикла/ } |

## **1.12 Операции языка**

Язык LPA-2024 поддерживает арифметические и логические операции. Приоритет: умножение, деление, деление с остатком, сложение и вычитание. Логические операции имеют равный приоритет.

Описание операций языка LPA-2024 приведено в таблице 1.5

Таблица 1.5 – Операции языка LPA-2024

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические операции | \* – умножение  / – деление  % – остаток от деления  + – сложение  - – вычитание  = – присваивание |
| Логические операции | > – больше  < – меньше  >= – больше или равно  <= – меньше или равно  == – проверка на равенство  != – проверка на неравенство |

## **1.13 Выражения и их вычисления**

Выражение языка программирования LPA-2024 представляет собой совокупность переменных, литералов, вызовов функций, знаков операций, скобок, которая может быть вычислена в соответствии с синтаксисом языка.

Правила составления выражений:

* выражение записывается в строку без переносов;
* в выражении могут присутствовать только операнды одинакового типа;
* в выражении могут использоваться функции. Как стандартные, так и пользовательские;
* в выражении не могут идти подряд два оператора;
* допускается использование круглых скобок для смены приоритета операций.

В арифметических выражениях допускаются только операнды целочисленного типа.

Перед генерацией кода выражения приводятся к ПОЛИЗ для более удобного вычисления на языке ассемблера.

## **Конструкции языка**

Программа на языке LPA-2024 состоит из функций пользователя и главной функции main. При составлении функций рекомендуется выделять блоки и фрагменты и применять отступы для улучшения читаемости кода.

Программные конструкции языка представлены в таблице 1.6.

Таблица 1.6 – Программные конструкции языка LPA-2024

|  |  |
| --- | --- |
| Конструкция | Описание |
| Главная функция | main  {  …  }; |
| Пользовательская функция | <тип данных возвращаемого значения> function<имя идентификатора> (<тип данных параметра> <имя идентификатора параметра>, …)  {  …  return <имя идентификатора переменной/литерал>;  };  Максимальное количество параметров: 8. |
| Цикл | cycle (<имя идентификатора переменной/литерал> <условный оператор> <имя идентификатора переменной/литерал>)  { …  } |

## **Область видимости идентификаторов**

Область видимости идентификаторов ограничивается блоками, в которых они объявлены. Переменные внутри функции недоступны вне её. Идентификаторы уникальны и локальны в пределах функции. Идентификатор нельзя использовать до его объявления.

## **Семантические проверки**

Семантическим анализатором языка LPA-2024 предусмотрены следующие проверки:

* наличие блока main – точки входа в программу;
* единственность точки входа;
* использование идентификаторов до их объявления;
* переопределение идентификаторов;

– соответствие параметров, передаваемых в функцию, с параметрами в объявлении функции;

* соответствие типа возвращаемого значения с типом функции;
* соответствие типов в выражениях;
* превышение размера целочисленных и строковых литералов;
* превышение длины лексемы;

– соответствие операторов типам данных, для работы с которыми они предназначены.

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

Память делится на сегмент констант для хранения литералов и сегмент данных для переменных и параметров. Локальность обеспечивается префиксами в именах идентификаторов. Префикс определяется именем функции, в которой находится переменная.

## **Стандартная библиотека и её состав**

В языке LPA-2024 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Содержание библиотеки и описание функций представлено в таблице 1.7.

Таблица 1.7 – Стандартная библиотека языка LPA-2024

|  |  |
| --- | --- |
| Функция | Описание |
| str strduplicate (str parm1, str parm2); | Строковая функция. Выполняет копирование строки parm2 в parm1. Возвращает parm1. |
| int strlength (str parm1); | Целочисленная функция. Вычисляет и возвращает длину строки parm1. |

Окончание таблицы 1.7

|  |  |
| --- | --- |
| int strtransint (str parm1); | Целочисленная функция. Преобразует строку parm1 в число. Строка не должна состоять из буквенных или специальных символов. Необходима для работы функции strlength. Возвращает число. |

Стандартная библиотека написана на языке C++. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций.

## **Ввод и вывод данных**

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

Функции управляющие выводом данных, реализованы на языке C++ и вызываются из транслированного кода, конечному пользователю недоступны. На этапе генерации кода операторы вывода языка LPA-2024 заменяются на встроенные функции, находящиеся в стандартной библиотеке.

## **Точка входа**

В языке LPA-2024 точкой входа в программу является главная функция main.

## **Препроцессор**

Препроцессор в языке программирования LPA-2024 не предусмотрен.

## **Соглашение о вызовах**

В языке LPA-2024 вызов функций происходит по стандартному соглашению о вызовах stdcall. Данное соглашение имеет особенности:

* все параметры функции передаются через стек;
* память высвобождает вызываемый код;
* занесение в стек параметров идёт справа налево.

## **1.23 Объектный код**

Язык LPA-2024 транслируется в язык ассемблера.

## **1.24 Классификация сообщений транслятора**

Описание и классификация сообщений компилятора об ошибках приведено в таблице 1.8.

Таблица 1.8 – Описание ошибок транслятора языка LPA-2024

|  |  |
| --- | --- |
| Интервал кодов | Описание |
| 0-9 | Системные ошибки. |
| 10-19 | Ошибки параметров. |
| 20-29 | Ошибки файлов. |
| 110-129 | Ошибки лексического анализатора. |
| 130-149 | Ошибки семантического анализатора. |
| 600-609 | Ошибки синтаксического анализатора. |
| 610-999 | Зарезервированные коды ошибок. |

## **1.25 Контрольный пример**

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

# **2.** **Спецификация языка программирования**

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

Транслятор – программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке.

Схема принципа работы транслятора изображена на рисунке 2.1.

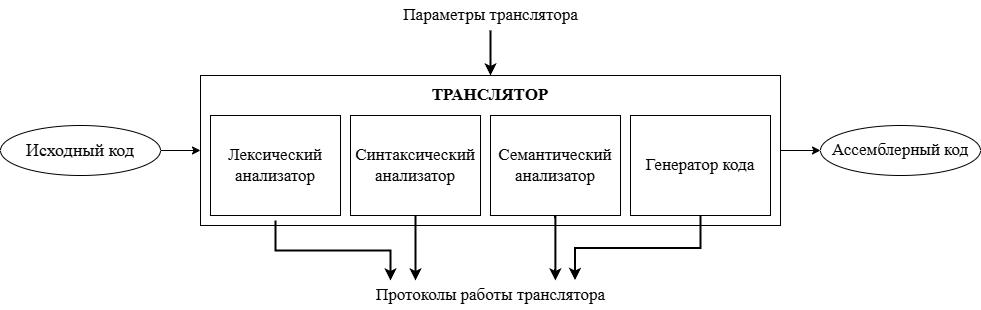


Рисунок 2.1 – принцип работы транслятора

В языке LPA-2024 исходный код транслируется в язык ассемблера. Транслятор разделён на 4 основные части: лексический анализатор, синтаксический анализатор, семантический анализатор и генератор кода. Каждый этап имеет входные и выходные данные, которые передаются следующему компоненту транслятора.

Первый этап работы компилятора – лексический анализатор. На вход подаётся исходный код программы. Лексический анализатор делит код на токены, которые затем идентифицируются и заменяются на лексемы. На выходе мы получаем две таблицы: таблицу лексем и таблицу идентификаторов.

Второй этап работы компилятора – синтаксический анализатор. На вход подаётся таблица лексем и таблица идентификаторов. На выходе мы получаем дерево разбора.

Третий этап работы компилятора – семантический анализатор. На этом этапе проверяется соблюдения в исходной программе семантических правил входного языка. На вход подаётся таблица идентификаторов, таблица лексем и дерево разбора.

Последний этап работы компилятора – генератор кода. На вход генератора подаются таблица лексем и таблица идентификаторов. На основе таблиц 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 – Стандартная библиотека языка LPA-2024

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к файлу> | Файл с исходным кодом на языке LPA-2024 в формате .txt. | Не предусмотрено |
| -out:<путь к файлу> | Выходной файл, содержащий код на языке ассемблера | Не предусмотрено |
| -log:<путь к файлу> | Файл с протоколом работы транслятора. | <имя файла in>.log |
| -an:<путь к файлу> | Файл, содержащий таблицы лексем, таблицу идентификаторов и дерево разбора. | <имя файла>.an.txt |

## **2.3 Протоколы, формируемые транслятором**

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

Таблица 2.2 – Стандартная библиотека языка LPA-2024

|  |  |
| --- | --- |
| Входной параметр | Описание |
| Файл, заданный параметром «-log:» | Содержит информацию о ходе выполнения трансляции: перечисление входных параметров, количество символов и строк, успех или ошибку по каждому этапу трансляции. Если обнаружена ошибка, в файл будет выведена вся информация об ошибке. |
| Файл, заданный параметром «-an:» | Содержит таблицы лексем и идентификаторов и дерево разбора. |

# **3. Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор – это программа, которая преобразует исходный текст программы, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Структура лексического анализатора изображена на рисунке 3.1.

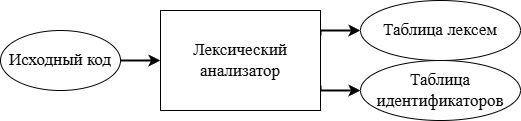


Рисунок 3.1 – принцип работы транслятора

Лексический анализатор работает в два этапа:

* Разбиение исходного кода программы на токены.
* Идентификация токенов и замена их на лексемы. Заполнение таблицы лексем и таблицы идентификаторов.

Входные данные: исходный код программы.

Результат работы: таблица лексем и таблица идентификаторов.

## **3.2 Контроль входных символов**

Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. Таблица контроля входных символов языка LPA-2024 представлена на рисунке 3.2.

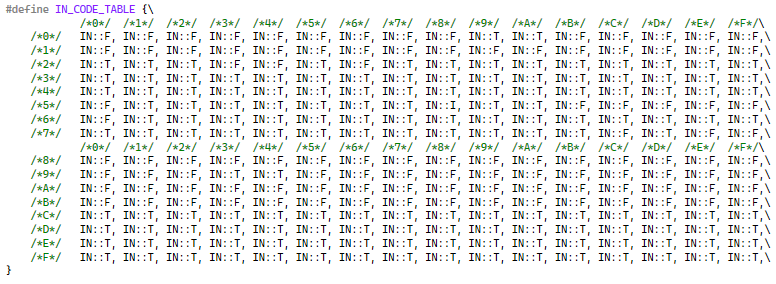


Рисунок 3.2 – Таблица контроля входных символов

Каждому элементу соответствует значение в шестнадцатеричной системе счисления из кодировки Windows-1251, изображённой на рисунке 1.1.

Каждый считанный символ из исходного кода проверяется на допустимость. Если символ в таблице обозначен как T, то данный символ считается разрешённым и записывается в строку, если символ обозначен как F, то его позиция в исходном коде записывается в структуру ошибок. В случае обнаружения недопустимого символа будет сгенерирована ошибка и она будет выведена в консоль вместе с позицией в исходном коде. Символ, обозначенный буквой I – игнорируемый.

## **3.3 Удаление избыточных символов**

Избыточный символ – это символ, отсутствие которого не влияет на исходный текст программы. В языке LPA-2024 символ пробела и табуляции являются избыточными символами. Их удаление предусмотрено на этапе разбиения исходного кода на токены.

Описание алгоритма удаления избыточных символов:

1. посимвольно считываем файл с исходным кодом программы;
2. пробел и табуляция являются символами-сепараторами, которые сигнализируют о начале или конце записи токена;
3. в отличии от других символов-сепараторов пробел и табуляцию не записываем в очередь лексем. Таким образом они будут проигнорированы.

## **3.4 Перечень ключевых слов**

Ключевые слова, сепараторы, символы операций, соответствующие им лексемы приведены в таблице 3.1.

Таблица 3.1 – Соответствие токенов и сепараторов с лексемами.

|  |  |  |
| --- | --- | --- |
| Слово (токен) | Лексема | Описание |
| int | d | Целочисленный тип данных. |
| char | c | Символьный тип данных. |
| str | s | Строковый тип данных. |
| bool | b | Логический тип данных. |
| идентификатор | i | Идентификатор типов данных. |
| литерал | l | Литерал любого типа данных. |
| main | m | Главная функция – точка входа в программу. |
| return | r | Выход из функции и возврат значения. |
| strduplicate | g | Стандартная строковая функция, выполняющая копирование строк. |
| strlength | h | Стандартная целочисленная функция. Вычисляет и возвращает длину строки. |
| strtransint | t | Стандартная целочисленная функция. Преобразует строку в число. |
|  |  |  |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Слово (токен) | Лексема | Описание |
| strtransint | t | Стандартная целочисленная функция. Преобразует строку в число. |
| function | f | Начало объявления функции. |
| new | n | Начало объявления переменной. |
| writeline | w | Оператор вывода в консоль. |
| write | V | Оператор вывода в консоль. |
| cycle | p | Оператор цикла. |
| { | { | Начало блока кода. |
| } | } | Конец блока кода. |
| ( | ( | Начало перечислений параметров у функции. Начало приоритета операций в выражениях. |
| ) | ) | Конец перечислений параметров у функции. Конец приоритета операций в выражениях. |
| + | + | Арифметический оператор (сложение). |
| - | - | Арифметический оператор (вычитание). |
| \* | \* | Арифметический оператор (умножение). |
| / | / | Арифметический оператор (деление). |
| % | % | Арифметический оператор (остаток от деления). |
| = | = | Арифметический оператор (присваивание). |
| > | > | Оператор сравнения (больше). |
| < | < | Оператор сравнения (меньше). |
| >= | ] | Оператор сравнения (больше либо равно). |
| <= | [ | Оператор сравнения (меньше либо равно). |
| == | # | Оператор сравнения (равенство). |
| != | ! | Оператор сравнения (неравенство). |
| , | , | Разделитель параметров функции. |
| ; | ; | Конец инструкции. |

Каждому токену соответствует детерминированный конечный автомат, по которому происходит разбор выражения. Графы переходов конечных автоматов для ключевых слов языка LPA-2024 представлены в приложении Б.

## **3.5 Основные структуры данных**

Основные структуры данных лексического анализатора: таблица токенов, таблица лексем и таблица идентификаторов.

В листинге 3.1 представлена реализация структуры токена и таблицы токенов в языке LPA-2024.

|  |
| --- |
| struct Token // Структура токена  {  char token[258];  int length;  int line;  int linePosition;  };  struct TokenTable // Структура таблицы токенов  {  int maxsize;  int size;  Token\* table;  }; |

Листинг 3.1 – Структура токена и таблицы токенов

Структура Token представляет собой строку таблицы TokenTable, где в массив token записывается слово, length – длина слова, line – номер строки в исходном тексте программы, linePosition – позиция в строке.

Структура TokenTable представляет таблицу токенов, где maxsize – максимальный размер таблицы, size – текущий размер таблицы, table – указатель на строку таблицы.

В листинге 3.2 представлена реализация структуры таблицы лексем.

|  |
| --- |
| struct Entry  {  char lexema; // символ лексемы  int sn; // номер строки в коде  int idxTI; // номер в IT  };  struct LexTable  {  int maxsize;  int size;  Entry\* table;  }; |

Листинг 3.2 – Структура таблицы лексем

Структура Entry представляет собой строку таблицы LexTable, где lexeme – лексема, sn – номер строки в исходном тексте программы, idxTI – номер в таблице идентификаторов.

Структура LexTable представляет таблицу лексем, где maxsize – число, равное максимальному размеру таблицы, size – текущий размер таблицы, table – указатель на строку таблицы.

В листинге 3.3 представлена реализация структуры таблицы идентификаторов.

|  |
| --- |
| struct Entry  {  int idxfirstLE; // идекс первого вхождения в таблицу лексем  std::string id; // идентификатор  IDDATATYPE idDataType; // тип данных  IDTYPE idType; // тип идентиикатора  struct  {  int vshort; // целочисленное значение  char vchar; // символ  struct  {  int len; // длин астроки  std::string str; // строка  } vstr;  } value;  };  struct IdTable  {  int maxsize;  int size;  Entry\* table;  }; |

Листинг 3.3 – Структура таблицы идентификаторов

Структура Entry представляет собой строку таблицы idTable, где переменная idxfirstLE – индекс первого вхождения в таблицу лексем, id – идентификатор, idDataType – тип данных, idType – тип идентификатора, vshort – целочисленное значение, len – длина строки, str – строка.

Структура IdTable представляет таблицу идентификаторов, где maxsize – максимальный размер таблицы, size – текущий размер таблицы, table – указатель на строку таблицы.

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

При возникновении ошибки в лексическом анализаторе формируется ошибка в следующем формате: Номер ошибки, пояснительный текст, строка в исходном тексте, позиция в строке. Перечень сообщений лексического анализатора приведен в таблице 3.2.

Таблица 3.2 – Перечень сообщений лексического анализатора языка LPA-2024

|  |  |
| --- | --- |
| Код ошибки | Пояснительный текст |
| 110 | Ошибка лексического анализатора: Недопустимый символ в исходном файле (-in:) |
| 111 | Ошибка лексического анализатора: Превышена емкость таблицы лексем |

Продолжение таблицы 3.2

|  |  |
| --- | --- |
| Код ошибки | Пояснительный текст |
| 112 | Ошибка лексического анализатора: Превышено количество строк в таблице лексем |
| 113 | Ошибка лексического анализатора: В таблице лексем отсутствует строка с заданным номером |
| 114 | Ошибка лексического анализатора: Превышена емкость таблицы идентификаторов |
| 115 | Ошибка лексического анализатора: Превышено количество строк в таблице идентификаторов |
| 116 | Ошибка лексического анализатора: В таблице идентификаторов отсутствует строка с заданным номером |
| 117 | Ошибка лексического анализатора: Не удалось определить тип лексемы |
| 118 | Ошибка лексического анализатора: Превышена емкость таблицы токенов |
| 119 | Ошибка лексического анализатора: Превышено количество токенов в таблице токенов |
| 120 | Ошибка лексического анализатора: Ошибка с разбиением исходного текста на токены |
| 121 | Ошибка лексического анализатора: Ошибка с разбором строкового литерала |
| 122 | Ошибка лексического анализатора: Ошибка с разбором символьного литерала |

* 1. **Принцип обработки ошибок**

При обнаружении ошибки в исходном коде программы лексический анализатор формирует сообщение об ошибке и выводит его в консоль и записывает в файл с протоколом работы, заданный параметром –log:. Максимальное количество кодов с ошибками – 1000. Максимальная длина поясняющего сообщения ошибки – 200 символов.

* 1. **Параметры лексического анализатора**

Входным параметром лексического анализатора является исходный текст программы после проверки на допустимость символов. Этот параметр используется для создания таблицы лексем и таблицы идентификаторов, необходимых для дальнейшей работы транслятора.

* 1. **Алгоритм лексического анализа**

Алгоритм лексического анализа языка LPA-2024 выполняется по следующему порядку:

1. считывается исходный текст программы и делится на токены, формируя структуру таблицы токенов;
2. слова из токенов проходят через графы конечного автомата, определяя тип лексем;
3. составляется таблица лексем и таблица идентификаторов

Исходный код программы, реализующий алгоритм лексического анализа представлен в листинге 3.4.

|  |
| --- |
| namespace LA  {  struct LEX  {  IT::IdTable idtable;  LT::LexTable lextable;  LEX();  LEX(int lexTableSize, int idTableSize);  };  char LexType(Tokens::Token token); // Определяем тип лексемы  LEX FillingInTables(Tokens::TokenTable tokenTable); // Заполнение таблиц  int SearchID(stack<int> areaOfVisibility, string id, IT::IdTable& idTable); // Поиск id в таблице  int SearchGlobalFunctionID(int globalAreaOfVisibility, string id, IT::IdTable& idTable); // Поиск id функции в таблице  } |

Листинг 3.4 – Реализация лексического анализатора

* 1. **Контрольный пример**

Результат работы лексического анализатора, полученный при выполнении контрольного примера представлен в приложении В.

# **4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализ языка LPA-2024 выполняется после завершения работы лексического анализатора. Синтаксический анализатор предназначен для сопоставления последовательности лексем языка LPA-2024 с его формальной грамматикой.

Входные данные: таблица лексем, таблица идентификаторов.

Результат работы: дерево разбора.

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

В синтаксическом анализаторе языка LPA-2024 используется грамматика типа 2 иерархии Хомского (контекстно-свободная грамматика) G = {N, T, P, S}, где N – конечный алфавит нетерминальных символов, приведённый в первом столбце таблицы 4.1; T – конечный алфавит терминальных символов; P – конечное множество правил прохождения; S – начальный нетерминал грамматики G.

Контекстно-свободная грамматика G имеет нормальную форму Грейбах, если она не является леворекурсивной и правила P имеют вид:

* A → aα, где a ∈ T, α ∈ N\*;
* S → λ, где S ∈ N – начальный символ, если есть такое правило, то S не должен встречаться в правой части правил.

Алгоритм преобразования грамматик в нормальную форму Грейбах:

* исключить недостижимые символы из грамматики;
* исключить лямбда-правила из грамматики;
* исключить цепные правила.

Перечень и описание терминальных, нетерминальных символов и правил языка приведен в таблице 4.1.

Таблица 4.1 – Описания правил, составляющих грамматику языка LPA-2024

|  |  |  |
| --- | --- | --- |
| Нетерминальный символ | Цепочки правил | Описание |
| S | m{N};  m{N};S  dfi(){N};S  dfi(){N};  bfi(){N};S  bfi(){N};  sfi(){N};S  sfi(){N};  dfi(F){N};  bfi(F){N};S  bfi(F){N}; | Правила, порождающие главную функцию main и глобальные функции. |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминальный символ | Цепочки правил | Описание |
|  | sfi(F){N};S sfi(F){N}; |  |
| N | nTi;N  nTi=E;N  nTi;  nTi=E;  i=E;  i=E;N  oL;N  oL;  p(W);N  p(W);  a(W);N  a(W);  i(W);N  i(W);  i();N  i();  cQ{N}N  cQ{N}  cQ{N}e{N}  cQ{N}e{N}N  rL;  N rL; | Правила порождающие конструкции в функциях. |
| F | di  si  bi  ci  di,F  si,F  bi,F  ci,F | Правила, порождающие параметры объявления функции. |
| E | i  l  (E)  i(W)  i()  p(W)  a(W)  iM  lM | Правила, порождающие выражения. |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминальный символ | Цепочки правил | Описание |
|  | (E)M  i(W)M  i()M  p(W)M  a(W)M |  |
| W | i  l  i,W  l,W  i()  i(),W  i(W)  i(W),W  p(W)  p(W),W  a(W)  a(W),W | Правила, порождающие параметры вызываемой функции. |
| M | +E  +EM  -E  -EM  /E  /EM  \*E  \*EM  %E  %EM  !L  &L  >L  <L  ]L  [L | Правила, порождающие операторы. |
| Q | (L<L)  (L>L)  (L!L)  (L&L)  (L]L)  (L[L) | Правила, порождающие условия в циклах |
| L | L  i | Правила, порождающие литерал и идентификатор |

Окончание таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминальный символ | Цепочки правил | Описание |
| T | d  s  b  c | Правила, порождающие типы данных |

## **4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку, описание которой приведено ниже.

M = {Q, V, Z, δ, q0, z0, F}, где

* Q – множество состояний;
* V – алфавит входных символов;
* Z – специальный алфавит магазинных символов;
* δ – функция переходов автомата;
* q0 ∈ Q – начальное состояние автомата;
* z0 ∈ Z – начальное состояние магазина (маркер дна);
* F ⊆ Q – множество конечных состояний

Схема конечного автомата с магазинной памятью изображена на рисунке 4.1.

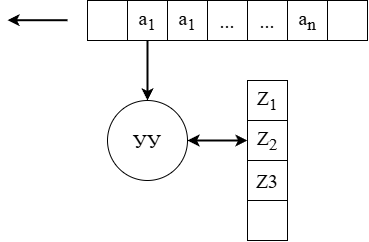


Рисунок 4.1 – принцип работы транслятора

Алгоритм работы конечного автомата с магазинной памятью:

* состояние автомата (q, aα, zβ);
* читает символ a, находящийся под головкой (сдвигает ленту);
* не читает ничего (читает λ, не сдвигает ленту);
* из δ определяет новое состояние q', если (q', γ) ∈ δ(q, a, z) или (q', γ) ∈ δ(q, λ, z);
* читает верхний в стеке символ z и записывает цепочку γ т.к. (q', γ) ∈ δ(q, a, z), при этом, если γ = λ, то верхний символ магазина просто удаляется;
* работа автомата заканчивается (q, λ, λ).

Результат, демонстрирующий успешный разбор цепочки из контрольного примера, приведён в приложении Г.

## **4.4 Основные структуры данных**

Программный код основных структур данных на языке C++, описывающих контекстно-свободную грамматику, представлен в приложении Д.

## **4.5 Описание алгоритма синтаксического разбора**

Алгоритм синтаксического разбора языка LPA-2024:

1. в магазин заносится стартовый символ;
2. формируется входная лента, полученная из таблицы лексем;
3. нетерминальный символ раскрывается, согласно правилам, и записывается в магазин;
4. если терминал на вершине стека и в начале ленты совпадают, то данный терминал удаляется из входной ленты. Иначе возвращается в предыдущее состояние и выбирает другую цепочку нетерминала;
5. если в магазине встречается нетерминал, переход к пункту 3;
6. если достигнуто дно стека и входная цепочка пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

Обобщённая блок-схема алгоритма синтаксического анализа изображена в приложении Е.

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

При возникновении ошибки в синтаксическом анализаторе формируется ошибка в следующем формате: Номер ошибки, пояснительный текст, строка в исходном тексте.

Перечень сообщений синтаксического анализатора представлен в таблице 4.2.

Таблица 4.2 – Сообщения синтаксического анализатора языка LPA-2024

|  |  |
| --- | --- |
| Номер ошибки | Пояснительный текст |
| 600 | Неверная структура программы |
| 601 | Ошибка в конструкции блока кода |
| 602 | Ошибка в выражении |
| 603 | Ошибка в параметрах функции |
| 604 | Ошибка в параметрах вызываемой функции |
| 605 | Ошибочный оператор |
| 606 | Ошибка в условной конструкции |
| 607 | Ошибка типа |

## **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.

Выходными параметрами являются правила разбора, которые выводятся на консоль.

## **4.8 Принцип обработки ошибок**

Обработка ошибок происходит по алгоритму:

* синтаксический анализатор перебирает все правила и цепочки правил грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
* при невозможности подбора подходящей цепочки, то генерируется соответствующая ошибка.
* ошибки записываются в общую структуру ошибок.
* в случае нахождения ошибки после всей процедуры трассировки в протокол и на консоль будет выведено диагностическое сообщение в файл протокола .log.

## **4.9 Контрольный пример**

Дерево разбора, полученное в результате работы синтаксического анализатора при выполнении контрольного примера представлено в приложении Г.

# **5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ языка LPA-2024 выполняется после выполнения лексического и синтаксического анализа. Несмотря на это, некоторые семантические проверки выполняются на этапе лексического анализа. На вход семантического анализатора подаются таблица лексем и таблица идентификаторов. Схема семантического анализатора представлена на рисунке 5.1

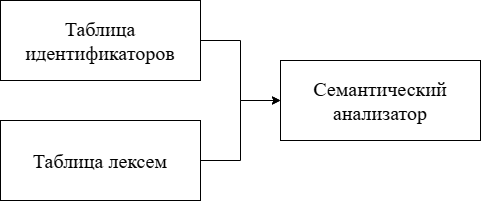


Рисунок 5.1 – Схема семантического анализатора

## **5.2 Функции семантического анализатора**

Семантические проверки языка LPA-2024 с указанием фаз их выполнения приведены в таблице 5.1.

Таблица 5.1 – Семантические проверки языка LPA-2024

|  |  |
| --- | --- |
| Семантическая проверка | Фаза выполнения |
| Превышение длины строки | Лексический анализ |
| Превышение целочисленного значения | Лексический анализ |
| Повторная реализация функции main | Лексический анализ |
| Превышение длины лексемы | Лексический анализ |
| Наличие точки входа в программу | Лексический анализ |
| Объявление идентификатора перед использованием | Семантический анализ |
| Повторное объявление идентификатора | Семантический анализ |
| Соответствие типов в выражении | Семантический анализ |
| Количество параметров функции | Семантический анализ |
| Соответствие параметров объявленной и вызываемой функции | Семантический анализ |
| Соответствие параметров встроенной функции | Семантический анализ |
| Проверка левосторонних выражений | Семантический анализ |
| Повторная реализация функции | Семантический анализ |

Окончание таблицы 5.1

|  |  |
| --- | --- |
| Семантическая проверка | Фаза выполнения |
| Соответствие типа возвращаемого значения типу функции | Семантический анализ |
| Перегрузка оператора для работы со строками | Семантический анализ |

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

Структура и текст сообщений семантического анализатора приведены на рисунке 5.2.

Таблица 5.2 – Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| Номер ошибки | Пояснительный текст |
| 130 | Превышена длина строки в 255 символов |
| 131 | Функция main уже имеет реализацию |
| 132 | Превышена длина лексемы |
| 133 | Превышено значение целочисленного литерала (2 byte) |
| 134 | Не найдена точка входа в программу (main) |
| 135 | Идентификатор с таким именем не найден |
| 136 | Повторное объявление идентификатора |
| 137 | Несоответствие типов в выражении |
| 138 | Слишком много параметров в функции |
| 139 | Превышено количество функций |
| 140 | Несоответствие параметров объявленной и вызываемой функций |
| 141 | Несоответствие параметров встроенной функции |
| 142 | Левостороннее выражение не является идентификатором и не должно являться функцией |
| 143 | Данная функция уже имеет реализацию |
| 144 | В вызове функции отсутствуют () |
| 145 | Тип возвращаемого значения не соответствует типу функции |
| 146 | Оператор не предназначен для работы со строками. |
| 147 | В функции отсутствует возвращаемое значение. |
| 148 | Деление на 0 недопустимо. |
| 149 | Недопустимый строковый литерал. |

## **5.4 Принцип обработки ошибок**

При обнаружении ошибки в исходном коде программы семантический анализатор формирует сообщение об ошибки и выводит его на консоль и в файл с протоколом работы, заданный параметром –log:.

## **5.5 Принцип обработки ошибок**

Контрольный пример для демонстрации ошибок, диагностируемых семантическим анализатором вместе с отчетом выданных сообщений представлен в приложении А.

В таблице 5.3 представлены примеры некоторых ошибок в исходном коде и соответствующие им диагностические сообщения.

Таблица 5.3 – Примеры диагностических ошибок

|  |  |
| --- | --- |
| Исходный код | Текст сообщения |
| int function summa(int a, int b)  {  new int res = a + b;  return res;  };  int function summa(int a, int b)  {  new int res = a - b;  return res;  }; | Ошибка 130: Ошибка семантического анализа: Функция с таким идентификатором уже существует, строка 6, позиция 13 |
| main {  new int a;  new int a = 10;  }; | Ошибка 131: Ошибка семантического анализа: Повторное объявление идентификатора, строка 3, позиция 10 |
| main {  new str perem = “strochka”;  new str test = perem + 10;  }; | Ошибка 132: Ошибка семантического анализа: Оператор не предназначен для работы со строками, строка 3, позиция 23 |

# **6. Вычисление выражений**

## **6.1 Выражения, допускаемые языком**

В языке LPA-2024 допускаются выражения, применимые к целочисленным типам данных. Допускается использование функций, возвращающих целочисленное значение, в выражениях. Операции и их приоритетность приведены в таблице 6.1.

Таблица 6.1 – Операции и их приоритетность в языке LPA-2024

|  |  |
| --- | --- |
| Приоритет | Операция |
| 0 | ( |
| 0 | ) |
| 1 | , |
| 2 | + |
| 2 | - |
| 3 | \* |
| 3 | / |
| 3 | % |

Пример выражения: 5\*(23O+strlength(“text”)-1

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

Язык LPA-2024 транслируется в язык ассемблера, в котором все вычисления производятся через стек. Преобразование исходных выражений в обратную польскую запись, упрощает генерацию кода вычисления выражений в язык ассемблера. Алгоритм построения польской записи приведен ниже:

* читаем очередной символ;
* если символ является числом, добавляем его к выходной строке;
* если символ является функцией, помещаем его в стек;
* если символ является открывающей скобкой, помещаем его в стек;
* если символ является закрывающей скобкой: до тех пор, пока верхним элементом стека не станет открывающая скобка, выталкиваем элементы из стека в выходную строку. При этом открывающая скобка удаляется из стека, но в выходную строку не добавляется.
* если символ является операцией:

1. пока операция на вершине стека приоритетнее или пока на вершине стека функция;
2. помещаем операцию в стек.

Когда входная строка закончилась, выталкиваем все символы из стека в выходную строку.

Пример построения обратной польской записи: lllpl-\*l+.

## **6.3 Программная реализация обработки выражений**

Фрагмент кода, реализующего преобразование выражений в обратный польский формат представлен в листинге 6.1.

|  |
| --- |
| case LEX\_IDENTIFIER:  {  if (idtable.table[lextable.table[lextable\_pos].idxTI].idType == IT::IDTYPE::F)  {  stack.push(lextable.table[lextable\_pos]);  continue;  }  queue.push(lextable.table[lextable\_pos]);  continue;  }  case LEX\_LITERAL:  {  queue.push(lextable.table[lextable\_pos]);  continue;  }  case LEX\_LEFTHESIS\_OPEN:  {  stack.push(lextable.table[lextable\_pos]);  continue;  }  case LEX\_RIGHTHESIS\_CLOSE:  {  if (!stack.empty())  {  while (stack.top().lexema != LEX\_LEFTHESIS\_OPEN)  {  queue.push(stack.top());  stack.pop();  }  stack.pop();  }  continue;  } |

Листинг 6.1 – Преобразование выражений в обратный польский формат

При встрече лексемы идентификатора, идет проверка на функцию. Если идентификатор является функцией, идентификатор помещается в стек. Иначе идентификатор помещается в выходную строку.

При встрече литерала, литерал помещается в выходную строку.

При встрече открывающей скобки, скобка кладется в стек.

При встрече закрывающей скобки, идет извлечение элементов из стека, пока не будет достигнута открывающая скобка.

## **6.4 Контрольный пример**

В приложении В представлена обновленная таблица лексем, с выражениями, приведенными к обратной польской записи.

# **7. Генерация кода**

## **7.1 Структура генератора кода**

Трансляция с языка LPA-2024 производиться в язык ассемблера. Структура генератора кода изображена на рисунке 7.1.

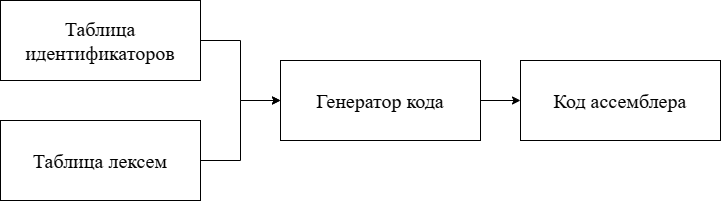


Рисунок 7.1 – Структура генератора кода

Генератор кода последовательно проходит таблицу лексем, при необходимости обращаясь к таблице идентификаторов. В зависимости от пройденных лексем выполняется генерация кода ассемблера.

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

Плоская модель памяти (flat): приложению для кода и данных предоставляется один непрерывный сегмент. Данный сегмент в свою очередь разбит на области:

* .STACK – стек;
* .CONST – константы;
* .DATA – переменные;
* .CODE – код.

Соответствие типов данных в исходном языке программирования LPA-2024 типам целевого языка приведены в таблице 7.1.

Таблица 7.1 – Соответствие типов идентификаторов языка LPA-2024 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора языка LPA-2024 | Тип идентификатора ассемблера | Пояснение |
| int | SDWORD | Хранит знаковый целочисленный тип. |
| str | DWORD | Хранит указатель на начало строки. |
| char | BYTE | Хранит один символ. |
| bool | DWORD | Хранит логическое значение. |

## **7.3 Статическая библиотека**

Функции, входящие в состав статической библиотеки языка LPA-2024, приведены в таблице 1.7.

Статическая библиотека написана на языке C++. Путь к статической библиотеке указан в Свойства проекта > Компоновщик > Командная строка. Библиотека подключается на этапе компоновки.

Таблица 1.7 – Стандартная библиотека языка LPA-2024

|  |  |
| --- | --- |
| Функция | Описание |
| str strduplicate (str parm1, str parm2); | Строковая функция. Выполняет копирование строки parm2 в parm1. Возвращает parm1. |
| int strlength (str parm1); | Целочисленная функция. Вычисляет и возвращает длину строки parm1. |
| int strtransint (str parm1); | Целочисленная функция. Преобразует строку parm1 в число. Строка не должна состоять из буквенных или специальных символов. Необходима для работы функции strlength. Возвращает число. |

## **7.4 Особенности алгоритма генерации кода**

Обобщенная блок-схема алгоритма генерации кода языка ассемблера изображена на рисунке 7.2.

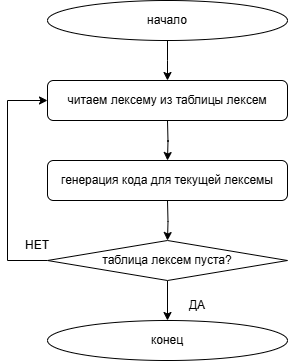


Рисунок 7.2 – Блок-схема алгоритма генерации кода в язык ассемблера

Пока таблица лексем не пуста, читаем лексему:

* если для лексемы есть код генерации, генерируем код;
* иначе читаем следующую лексему, пока таблица лексем не будет пуста.

Генерация кода строится с помощью блока макросов и функции void CodeGeneration(LA::LEX lex, wchar\_t outfile[]).

В листинге 7.1 представлен блок макросов, используемый при генерации кода.

|  |
| --- |
| #define START \  ".586\n"\  ".model flat, stdcall\n"\  "includelib libucrt.lib\n"\  "includelib kernel32.lib\n"\  #define PROTOTYPES \  "\nExitProcess PROTO:DWORD "\  "\nSYSPAUSE PROTO "\  "\nsoutl PROTO : BYTE "\  "\nnoutl PROTO : SDWORD "\  "\n\_pow PROTO : SDWORD, : SDWORD "\  "\n\_abs PROTO : SDWORD "\  "\n\n.STACK 4096\n\n"  #define FINISH \  "\tcall SYSPAUSE"\  "\n\tpush 0"\  "\n\tcall ExitProcess"\  "\nSOMETHINGWRONG::"\  "\n\tpush offset null\_division"\  "\n\tcall soutl"\  "\njmp THEEND"\  "\noverflow::"\  "\n\tpush offset OVER\_FLOW"\  "\n\tcall soutl"\  "\nTHEEND:"\  "\n\tcall SYSPAUSE"\  "\n\tpush -1"\  "\n\tcall ExitProcess"\  "\nmain ENDP\nend main" |

Листинг 7.1 – Использование макросов при генерации

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

На вход генератору кода поступают таблицы лексем и идентификаторов.

Результат работы генератора кода выводится в файл с расширением .asm.

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

Результат генерации кода на основе контрольного примера из приложения А представлен в приложении Ж.

# **8. Тестирование транслятора**

## **8.1 Общие положения**

Тесты предназначены для выявления ошибок в работе компилятора и последующего их устранения. Ошибки были выявлены как на раннем этапе разработки компилятора, так и на позднем.

После возникновения ошибки работа транслятора прекращается, поскольку ошибка на одном этапе трансляции может вызвать ошибки на последующих этапах (за исключением синтаксического анализатора). Текст с номером и сообщением об ошибки будет выведен в файл протокола и консоль.

## **8.2 Результаты тестирования**

Описание тестовых наборов, демонстрирующих проверки на разных этапах трансляции, приведено в таблице 8.1.

Таблица 8.1 – Описание тестовых наборов языка LPA-2024

|  |  |
| --- | --- |
| Фрагмент исходного кода | Диагностическое сообщение |
| Допустимость символов | |
| new int gf#; | Ошибка 110: Ошибка лексического анализатора: Недопустимый символ в исходном файле (-in:), строка 3, позиция 12. |
| Лексический анализ | |
| 24rgt str stroka; | Ошибка 117: Ошибка лексического анализатора: Не удалось определить тип лексемы, строка 3, позиция 1 |
| Лексический анализ | |
| main{  new str str2 = “qwerty;  }; | Ошибка 121: Ошибка лексического анализатора: Ошибка с разбором строкового литерала, строка 2, позиция 1 |
| Синтаксический анализ | |
| new main {} | Ошибка 600: Ошибка синтаксического анализа: Неверная структура программы, строка 1, позиция 1 |
| Cycle (i>5) { | Ошибка 601: Ошибка синтаксического анализа: Ошибка в конструкции блока кода, строка 3, позиция 1 |
| new int x = 5 ++ 4; | Ошибка 602: Ошибка синтаксического анализа: Ошибка в выражении, строка 2, позиция 1 |
| str function str1(str x,) {}; | Ошибка 603: Ошибка синтаксического анализа:  Ошибка в параметрах функции, строка 1, позиция 1 |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Фрагмент исходного кода | Диагностическое сообщение |
| Синтаксический анализ | |
| new int x = (54 – 5; | Ошибка 605: Ошибка синтаксического анализа:  Ошибочный оператор, строка 2, позиция 1 |
| main{ new ind; }; | Ошибка 607: Ошибка синтаксического анализа: Ошибка типа, строка 2, позиция 1 |
| Семантический анализ | |
| main{new str s;};  main{new int i;}; | Ошибка 131: Ошибка семантического анализа: Функция main уже имеет реализацию, строка 2, позиция 1 |
| new int k = 32800; | Ошибка 133: Ошибка семантического анализа: Превышено значение целочисленного литерала (2 byte), строка 2, позиция 14 |
| int function f1(){return 0;}; | Ошибка 134: Ошибка семантического анализа: Не найдена точка входа в программу (main), строка -1, позиция -1 |
| main{  x = 5;  }; | Ошибка 135: Ошибка семантического анализа: Идентификатор с таким именем не найден, строка 2, позиция 2 |
| main{  new int r;  new int r;  }; | Ошибка 136: Ошибка семантического анализа: Повторное объявление идентификатора, строка 3, позиция 13 |
| main{  new int a = 5;  new str b = “stroka”;  a = a + b;  }; | Ошибка 137: Ошибка семантического анализа: Несоответствие типов в выражении, строка 4, позиция 1 |
| int function f1(int x){  return x;  }; entry{ f1("str"); }; | Ошибка 140: Ошибка семантического анализа: Несоответствие параметров объявленной и вызываемой функций, строка 6, позиция 1 |

Окончание таблицы 8.1.

|  |  |
| --- | --- |
| Фрагмент исходного кода | Диагностическое сообщение |
| Семантический анализ | |
| int function f2(str x){…};  main{  f2 = 5;  }; | Ошибка 142: Ошибка семантического анализа: Левостороннее выражение не является идентификатором и не должно являться функцией, строка 7, позиция 1 |
| int function f3(int a){  …};  str function f3(str b){  … }; | Ошибка 143: Ошибка семантического анализа: Данная функция уже имеет реализацию, строка 5, позиция 14 |
| int function f4(int v){  …};  main{  new int c = f4;}; | Ошибка 144: Ошибка семантического анализа: В вызове функции отсутствуют (), строка 15, позиция 1 |
| int function f5(int v){  new str res = "qwerty";  return res; }; | Ошибка 145: Ошибка семантического анализа: Тип возвращаемого значения не соответствует типу функции, строка 4, позиция 1 |
| main{ new str s = "qwe" + "asd"; }; | Ошибка 146: Ошибка семантического анализа: Оператор не предназначен для работы со строками, строка 3, позиция 1 |
| int function f6(){  new int s;  }; | Ошибка 147: Ошибка семантического анализа: В функции отсутствует возвращаемое значение, строка 1, позиция 1 |
| main{ new int a = 3/0; }; | Ошибка 148: Ошибка семантического анализа: Деление на 0 недопустимо, строка 3, позиция 1 |
| main{ new str s = ""; }; | Ошибка 149: Ошибка семантического анализа: Недопустимый строковый литерал, строка 3, позиция 13 |

# **Заключение**

В ходе выполнения курсовой работы был разработан компилятор для языка LPA-2024. Выполнены минимальные требования к курсовому проекту, а также ряд дополнений.

Язык LPA-2024 включает в себя:

* 4 типа данных;
* 2 оператора вывода данных в консоль;
* Вызов функций из стандартной библиотеки;
* 5 арифметических операторов для вычисления выражений;
* 6 операторов сравнения;
* Оператор цикла.

Работа по разработке компилятора позволила получить необходимое представление о структурах и процессах, использующихся при построении компиляторов, также были изучены основы теории формальных грамматик и основы общей теории компиляторов

**Список использованных источников**

1 Курс лекций по конструированию программного обеспечения / Наркевич А. С.

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

3 Калашников, О. А. Ассемблер – это просто. Учимся программировать. – / О. А. Калашников. – Петербург: БХВ-Петербург, 2011. – 336с.

# **Приложение А**

Листинг контрольного примера на языке LPA-2024

|  |
| --- |
| int function testfunc1(int a, int b)  {  new int res = a + b;  return res;  };  str function testfunc2(str parm1)  {  new str parm2 = "TestText20";  strduplicate(parm1, parm2); // Копирование строки parm2 в parm1  return parm1;  };  char function testfunc3(char simbol)  {  new int i = 0;  cycle(i < 10) {  write simbol;  i = i + 1;  }  }  bool function testfunc4(int chetnoe)  {  return chetnoe % 2 == 0;  }  main  {  writeline "Типы данных:";  new int it = 1;  new char ct = 'a';  new string st = "teststring";  new bool bt = true;  writeline "Арифметические операции:";  new int a;  new int b = 15;  a = 10 \* 5; // Умножение  write a;  a = 10 / 5; // Деление  write a;  a = 15 % 10; // Деление с остатком  write a;  a = 10 + 5; // Сумма  write a;  a = 10 - 5; // Разница  write a;  writeline "Вызов функций:";  // Пользовательские функции:  new int number = testfunc1(0, 5);  write number;  new str parm1 = "TestText1";  new str string = testfunc2(parm1);  writeline string;  new char alpha = 'a';  testfunc3(alpha);  new int myint = 120;  new bool pravda = testfunc4(myint);  writeline pravda;  // Библиотечные функции:  new str stroka = "Тестовая строчка";  new str strokachislo = "125";  new copystr;  strduplicate(copystr, stroka); // Копирование строки stroka в copystr  writeline copystr;  new int dlina = strlength(stroka); // Вычисление длины строки stroka  writeline dlina;  new int testint = strtransint(strokachislo);  new int testint2 = 5 + testint;  writeline testint2;  writeline "Цикл:";  new int i = 0;  cycle(i < 10)  {  write i;  i = i + 1;  }  } |

Листинг А.1 – Контрольный пример на языке LPA-2024

|  |
| --- |
| int function testfunc1(int a, int b)  {  new int res = a + b;  return res;  };  str function testfunc1(str parm1) // Ошибка 130 (существующий идентификатор функции)  {  new str parm2 = "TestText20";  strduplicate(parm1, parm2); // Копирование строки parm2 в parm1  return parm1;  };  char function testfunc3(char simbol)  {  new int i = 0;  cycle(i < 10) {  write simbol;  i = i + 1;  }  }  bool function testfunc4(int chetnoe)  {  return chetnoe % 2 == 0;  }  main  {  writeline "Арифметические операции:";  new int a;  new int b = 15;  new str c = "text";  new int a = 10 \* 5; // Умножение. Ошибка 131 (повторное объявление идентификатора)  write a;  a = 10 / 5; // Деление  write a;  a = 15 % 10; // Деление с остатком  write a;  a = 10 + 5; // Сумма  write a;  a = 10 - 5; // Разница  write a;  new str d = c + 10; // Ошибка 132 (оператор не предназначен для работы со строками  writeline "Вызов функций:";  // Пользовательские функции:  new int number = testfunc1(0, 5);  write number;  new str parm1 = "TestText1";  new str string = testfunc1(parm1);  writeline string;  new char alpha = 'a';  new testfunc3(alpha);  new int myint = 120;  new bool pravda = testfunc4(myint);  writeline pravda;  // Библиотечные функции:  new str stroka = "Тестовая строчка";  new str strokachislo = "125";  new str copystr;  strduplicate(copystr, stroka); // Копирование строки stroka в copystr  writeline copystr;  new int dlina = strlength(stroka); // Вычисление длины строки stroka  writeline dlina;  new int testint = strtransint(strokachislo);  new int testint2 = 5 + testint;  writeline testint2;  writeline "Цикл:";  new int i = 0;  cycle(i < 10)  {  write i;  i = i + 1;  }  } |

Листинг А.2 – Контрольный пример, содержащий 3 семантические ошибки

Допущенные семантические ошибки:

* Ошибка 130: Ошибка семантического анализа: Функция с таким идентификатором уже существует, строка 7, позиция 13
* Ошибка 131: Ошибка семантического анализа: Повторное объявление идентификатора, строка 35, позиция 6
* Ошибка 132: Ошибка семантического анализа: Оператор не предназначен для работы со строками, строка 46, позиция 16

# **Приложение Б**

Листинг графов переходов конечных автоматов

|  |
| --- |
| struct GRAPH  {  char lex;  FST::FST graph; // Недетерминированный конечный автомат  };  // Типы данных  #define GRAPH\_SHORT \  4, \  FST::NODE(1, FST::RELATION('i', 1)), \  FST::NODE(1, FST::RELATION('n', 2)), \  FST::NODE(1, FST::RELATION('t', 3)), \  FST::NODE()  #define GRAPH\_CHAR \  5, \  FST::NODE(1, FST::RELATION('c', 1)), \  FST::NODE(1, FST::RELATION('h', 2)), \  FST::NODE(1, FST::RELATION('a', 3)), \  FST::NODE(1, FST::RELATION('r', 4)), \  FST::NODE()  #define GRAPH\_STR \  4, \  FST::NODE(1, FST::RELATION('s', 1)), \  FST::NODE(1, FST::RELATION('t', 2)), \  FST::NODE(1, FST::RELATION('r', 3)), \  FST::NODE()  #define GRAPH\_BOOL \  5, \  FST::NODE(1, FST::RELATION('b', 1)), \  FST::NODE(1, FST::RELATION('o', 2)), \  FST::NODE(1, FST::RELATION('o', 3)), \  FST::NODE(1, FST::RELATION('l', 4)), \  FST::NODE()  // Идентификаторы  #define GRAPH\_IDENTIFIER \  2, \  FST::NODE(62, \  FST::RELATION('a', 0), FST::RELATION('b', 0), FST::RELATION('c', 0), FST::RELATION('d', 0), \  FST::RELATION('e', 0), FST::RELATION('f', 0), FST::RELATION('g', 0), FST::RELATION('h', 0), \  FST::RELATION('i', 0), FST::RELATION('j', 0), FST::RELATION('k', 0), FST::RELATION('l', 0), \  FST::RELATION('m', 0), FST::RELATION('n', 0), FST::RELATION('o', 0), FST::RELATION('p', 0), \  FST::RELATION('q', 0), FST::RELATION('r', 0), FST::RELATION('s', 0), FST::RELATION('t', 0), \  FST::RELATION('u', 0), FST::RELATION('v', 0), FST::RELATION('w', 0), FST::RELATION('x', 0), \  FST::RELATION('y', 0), FST::RELATION('z', 0), \  FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), FST::RELATION('d', 1), \  FST::RELATION('e', 1), FST::RELATION('f', 1), FST::RELATION('g', 1), FST::RELATION('h', 1), \  FST::RELATION('i', 1), FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1), \  FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), FST::RELATION('p', 1), \  FST::RELATION('q', 1), FST::RELATION('r', 1), FST::RELATION('s', 1), FST::RELATION('t', 1), \  FST::RELATION('u', 1), FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1), \  FST::RELATION('y', 1), FST::RELATION('z', 1), \  FST::RELATION('0', 1), FST::RELATION('1', 1), FST::RELATION('2', 1), FST::RELATION('3', 1), \  FST::RELATION('4', 1), FST::RELATION('5', 1), FST::RELATION('6', 1), FST::RELATION('7', 1), \  FST::RELATION('8', 1), FST::RELATION('9', 1)), \  FST::NODE()  // Литералы  #define GRAPH\_BOOL\_TRUE \  5, \  FST::NODE(1, FST::RELATION('t', 1)), \  FST::NODE(1, FST::RELATION('r', 2)), \  FST::NODE(1, FST::RELATION('u', 3)), \  FST::NODE(1, FST::RELATION('e', 4)), \  FST::NODE()  #define GRAPH\_BOOL\_FALSE \  6, \  FST::NODE(1, FST::RELATION('f', 1)), \  FST::NODE(1, FST::RELATION('a', 2)), \  FST::NODE(1, FST::RELATION('l', 3)), \  FST::NODE(1, FST::RELATION('s', 4)), \  FST::NODE(1, FST::RELATION('e', 5)), \  FST::NODE()  #define GRAPH\_SHORT\_LITERAL\_8 \  3, \  FST::NODE(17, \  FST::RELATION('0', 0), FST::RELATION('1', 0), FST::RELATION('2', 0), \  FST::RELATION('3', 0), FST::RELATION('4', 0), FST::RELATION('5', 0), \  FST::RELATION('6', 0), FST::RELATION('7', 0), FST::RELATION('-', 0), \  FST::RELATION('0', 1), FST::RELATION('1', 1), FST::RELATION('2', 1), \  FST::RELATION('3', 1), FST::RELATION('4', 1), FST::RELATION('5', 1), \  FST::RELATION('6', 1), FST::RELATION('7', 1)), \  FST::NODE(1, FST::RELATION('O', 2)), \  FST::NODE()  #define GRAPH\_SHORT\_LITERAL\_10 \  2, \  FST::NODE(21, \  FST::RELATION('0', 0), FST::RELATION('1', 0), FST::RELATION('2', 0), \  FST::RELATION('3', 0), FST::RELATION('4', 0), FST::RELATION('5', 0), \  FST::RELATION('6', 0), FST::RELATION('7', 0), FST::RELATION('8', 0), \  FST::RELATION('9', 0), FST::RELATION('-', 0), \  FST::RELATION('0', 1), FST::RELATION('1', 1), FST::RELATION('2', 1), \  FST::RELATION('3', 1), FST::RELATION('4', 1), FST::RELATION('5', 1), \  FST::RELATION('6', 1), FST::RELATION('7', 1), FST::RELATION('8', 1), \  FST::RELATION('9', 1)), \  FST::NODE()  #define GRAPH\_STRING\_LITERAL \  4, \  FST::NODE(2, FST::RELATION('\"', 1), FST::RELATION('\"', 2)), \  FST::NODE(290, \  FST::RELATION('A', 1), FST::RELATION('B', 1), FST::RELATION('C', 1), \  FST::RELATION('D', 1), FST::RELATION('E', 1), FST::RELATION('F', 1), \  FST::RELATION('G', 1), FST::RELATION('H', 1), FST::RELATION('I', 1), \  FST::RELATION('J', 1), FST::RELATION('K', 1), FST::RELATION('L', 1), \  FST::RELATION('M', 1), FST::RELATION('N', 1), FST::RELATION('O', 1), \  FST::RELATION('P', 1), FST::RELATION('Q', 1), FST::RELATION('R', 1), \  FST::RELATION('S', 1), FST::RELATION('T', 1), FST::RELATION('U', 1), \  FST::RELATION('V', 1), FST::RELATION('W', 1), FST::RELATION('X', 1), \  FST::RELATION('Y', 1), FST::RELATION('Z', 1), \  FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), \  FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1), \  FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), \  FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1), \  FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), \  FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1), \  FST::RELATION('s', 1), FST::RELATION('t', 1), FST::RELATION('u', 1), \  FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1), \  FST::RELATION('y', 1), FST::RELATION('z', 1), \  \  FST::RELATION('0', 1), FST::RELATION('1', 1), FST::RELATION('2', 1), \  FST::RELATION('3', 1), FST::RELATION('4', 1), FST::RELATION('5', 1), \  FST::RELATION('6', 1), FST::RELATION('7', 1), FST::RELATION('8', 1), \  FST::RELATION('9', 1), \  \  FST::RELATION('А', 1), FST::RELATION('Б', 1), FST::RELATION('В', 1), \  FST::RELATION('Г', 1), FST::RELATION('Д', 1), FST::RELATION('Е', 1), \  FST::RELATION('Ё', 1), FST::RELATION('Ж', 1), FST::RELATION('З', 1), \  FST::RELATION('И', 1), FST::RELATION('Й', 1), FST::RELATION('К', 1), \  FST::RELATION('Л', 1), FST::RELATION('М', 1), FST::RELATION('Н', 1), \  FST::RELATION('О', 1), FST::RELATION('П', 1), FST::RELATION('Р', 1), \  FST::RELATION('С', 1), FST::RELATION('Т', 1), FST::RELATION('У', 1), \  FST::RELATION('Ф', 1), FST::RELATION('Х', 1), FST::RELATION('Ц', 1), \  FST::RELATION('Ч', 1), FST::RELATION('Ш', 1), FST::RELATION('Щ', 1), \  FST::RELATION('Ъ', 1), FST::RELATION('Ы', 1), FST::RELATION('Ь', 1), \  FST::RELATION('Э', 1), FST::RELATION('Ю', 1), FST::RELATION('Я', 1), \  FST::RELATION('а', 1), FST::RELATION('б', 1), FST::RELATION('в', 1), \  FST::RELATION('г', 1), FST::RELATION('д', 1), FST::RELATION('е', 1), \  FST::RELATION('ё', 1), FST::RELATION('ж', 1), FST::RELATION('з', 1), \  FST::RELATION('и', 1), FST::RELATION('й', 1), FST::RELATION('к', 1), \  FST::RELATION('л', 1), FST::RELATION('м', 1), FST::RELATION('н', 1), \  FST::RELATION('о', 1), FST::RELATION('п', 1), FST::RELATION('р', 1), \  FST::RELATION('с', 1), FST::RELATION('т', 1), FST::RELATION('у', 1), \  FST::RELATION('ф', 1), FST::RELATION('х', 1), FST::RELATION('ц', 1), \  FST::RELATION('ч', 1), FST::RELATION('ш', 1), FST::RELATION('щ', 1), \  FST::RELATION('ъ', 1), FST::RELATION('ы', 1), FST::RELATION('ь', 1), \  FST::RELATION('э', 1), FST::RELATION('ю', 1), FST::RELATION('я', 1), \  \  FST::RELATION(' ', 1), FST::RELATION('.', 1), FST::RELATION(',', 1), \  FST::RELATION('?', 1), FST::RELATION('!', 1), FST::RELATION(';', 1), \  FST::RELATION(':', 1), FST::RELATION('-', 1), FST::RELATION(')', 1), \  FST::RELATION('(', 1), FST::RELATION('<', 1), FST::RELATION('>', 1), \  FST::RELATION('/', 1), FST::RELATION('%', 1), FST::RELATION('=', 1), \  FST::RELATION('+', 1), FST::RELATION('\*', 1), \  \  FST::RELATION('A', 2), FST::RELATION('B', 2), FST::RELATION('C', 2), \  FST::RELATION('D', 2), FST::RELATION('E', 2), FST::RELATION('F', 2), \  FST::RELATION('G', 2), FST::RELATION('H', 2), FST::RELATION('I', 2), \  FST::RELATION('J', 2), FST::RELATION('K', 2), FST::RELATION('L', 2), \  FST::RELATION('M', 2), FST::RELATION('N', 2), FST::RELATION('O', 2), \  FST::RELATION('P', 2), FST::RELATION('Q', 2), FST::RELATION('R', 2), \  FST::RELATION('S', 2), FST::RELATION('T', 2), FST::RELATION('U', 2), \  FST::RELATION('V', 2), FST::RELATION('W', 2), FST::RELATION('X', 2), \  FST::RELATION('Y', 2), FST::RELATION('Z', 2), \  FST::RELATION('a', 2), FST::RELATION('b', 2), FST::RELATION('c', 2), \  FST::RELATION('d', 2), FST::RELATION('e', 2), FST::RELATION('f', 2), \  FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('i', 2), \  FST::RELATION('j', 2), FST::RELATION('k', 2), FST::RELATION('l', 2), \  FST::RELATION('m', 2), FST::RELATION('n', 2), FST::RELATION('o', 2), \  FST::RELATION('p', 2), FST::RELATION('q', 2), FST::RELATION('r', 2), \  FST::RELATION('s', 2), FST::RELATION('t', 2), FST::RELATION('u', 2), \  FST::RELATION('v', 2), FST::RELATION('w', 2), FST::RELATION('x', 2), \  FST::RELATION('y', 2), FST::RELATION('z', 2), \  \  FST::RELATION('0', 2), FST::RELATION('1', 2), FST::RELATION('2', 2), \  FST::RELATION('3', 2), FST::RELATION('4', 2), FST::RELATION('5', 2), \  FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2), \  FST::RELATION('9', 2), \  \  FST::RELATION('А', 2), FST::RELATION('Б', 2), FST::RELATION('В', 2), \  FST::RELATION('Г', 2), FST::RELATION('Д', 2), FST::RELATION('Е', 2), \  FST::RELATION('Ё', 2), FST::RELATION('Ж', 2), FST::RELATION('З', 2), \  FST::RELATION('И', 2), FST::RELATION('Й', 2), FST::RELATION('К', 2), \  FST::RELATION('Л', 2), FST::RELATION('М', 2), FST::RELATION('Н', 2), \  FST::RELATION('О', 2), FST::RELATION('П', 2), FST::RELATION('Р', 2), \  FST::RELATION('С', 2), FST::RELATION('Т', 2), FST::RELATION('У', 2), \  FST::RELATION('Ф', 2), FST::RELATION('Х', 2), FST::RELATION('Ц', 2), \  FST::RELATION('Ч', 2), FST::RELATION('Ш', 2), FST::RELATION('Щ', 2), \  FST::RELATION('Ъ', 2), FST::RELATION('Ы', 2), FST::RELATION('Ь', 2), \  FST::RELATION('Э', 2), FST::RELATION('Ю', 2), FST::RELATION('Я', 2), \  FST::RELATION('а', 2), FST::RELATION('б', 2), FST::RELATION('в', 2), \  FST::RELATION('г', 2), FST::RELATION('д', 2), FST::RELATION('е', 2), \  FST::RELATION('ё', 2), FST::RELATION('ж', 2), FST::RELATION('з', 2), \  FST::RELATION('и', 2), FST::RELATION('й', 2), FST::RELATION('к', 2), \  FST::RELATION('л', 2), FST::RELATION('м', 2), FST::RELATION('н', 2), \  FST::RELATION('о', 2), FST::RELATION('п', 2), FST::RELATION('р', 2), \  FST::RELATION('с', 2), FST::RELATION('т', 2), FST::RELATION('у', 2), \  FST::RELATION('ф', 2), FST::RELATION('х', 2), FST::RELATION('ц', 2), \  FST::RELATION('ч', 2), FST::RELATION('ш', 2), FST::RELATION('щ', 2), \  FST::RELATION('ъ', 2), FST::RELATION('ы', 2), FST::RELATION('ь', 2), \  FST::RELATION('э', 2), FST::RELATION('ю', 2), FST::RELATION('я', 2), \  \  FST::RELATION(' ', 2), FST::RELATION('.', 2), FST::RELATION(',', 2), \  FST::RELATION('?', 2), FST::RELATION('!', 2), FST::RELATION(';', 2), \  FST::RELATION(':', 2), FST::RELATION('-', 2), FST::RELATION(')', 2), \  FST::RELATION('(', 2), FST::RELATION('<', 2), FST::RELATION('>', 2), \  FST::RELATION('/', 2), FST::RELATION('%', 2), FST::RELATION('=', 2), \  FST::RELATION('+', 2), FST::RELATION('\*', 2)), \  FST::NODE(1, FST::RELATION('\"', 3)), \  FST::NODE()  #define GRAPH\_CHAR\_LITERAL \  4, \  FST::NODE(2, FST::RELATION('\'', 1), FST::RELATION('\'', 2)), \  FST::NODE(145, \  FST::RELATION('A', 2), FST::RELATION('B', 2), FST::RELATION('C', 2), \  FST::RELATION('D', 2), FST::RELATION('E', 2), FST::RELATION('F', 2), \  FST::RELATION('G', 2), FST::RELATION('H', 2), FST::RELATION('I', 2), \  FST::RELATION('J', 2), FST::RELATION('K', 2), FST::RELATION('L', 2), \  FST::RELATION('M', 2), FST::RELATION('N', 2), FST::RELATION('O', 2), \  FST::RELATION('P', 2), FST::RELATION('Q', 2), FST::RELATION('R', 2), \  FST::RELATION('S', 2), FST::RELATION('T', 2), FST::RELATION('U', 2), \  FST::RELATION('V', 2), FST::RELATION('W', 2), FST::RELATION('X', 2), \  FST::RELATION('Y', 2), FST::RELATION('Z', 2), \  FST::RELATION('a', 2), FST::RELATION('b', 2), FST::RELATION('c', 2), \  FST::RELATION('d', 2), FST::RELATION('e', 2), FST::RELATION('f', 2), \  FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('i', 2), \  FST::RELATION('j', 2), FST::RELATION('k', 2), FST::RELATION('l', 2), \  FST::RELATION('m', 2), FST::RELATION('n', 2), FST::RELATION('o', 2), \  FST::RELATION('p', 2), FST::RELATION('q', 2), FST::RELATION('r', 2), \  FST::RELATION('s', 2), FST::RELATION('t', 2), FST::RELATION('u', 2), \  FST::RELATION('v', 2), FST::RELATION('w', 2), FST::RELATION('x', 2), \  FST::RELATION('y', 2), FST::RELATION('z', 2), \  \  FST::RELATION('0', 2), FST::RELATION('1', 2), FST::RELATION('2', 2), \  FST::RELATION('3', 2), FST::RELATION('4', 2), FST::RELATION('5', 2), \  FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2), \  FST::RELATION('9', 2), \  \  FST::RELATION('А', 2), FST::RELATION('Б', 2), FST::RELATION('В', 2), \  FST::RELATION('Г', 2), FST::RELATION('Д', 2), FST::RELATION('Е', 2), \  FST::RELATION('Ё', 2), FST::RELATION('Ж', 2), FST::RELATION('З', 2), \  FST::RELATION('И', 2), FST::RELATION('Й', 2), FST::RELATION('К', 2), \  FST::RELATION('Л', 2), FST::RELATION('М', 2), FST::RELATION('Н', 2), \  FST::RELATION('О', 2), FST::RELATION('П', 2), FST::RELATION('Р', 2), \  FST::RELATION('С', 2), FST::RELATION('Т', 2), FST::RELATION('У', 2), \  FST::RELATION('Ф', 2), FST::RELATION('Х', 2), FST::RELATION('Ц', 2), \  FST::RELATION('Ч', 2), FST::RELATION('Ш', 2), FST::RELATION('Щ', 2), \  FST::RELATION('Ъ', 2), FST::RELATION('Ы', 2), FST::RELATION('Ь', 2), \  FST::RELATION('Э', 2), FST::RELATION('Ю', 2), FST::RELATION('Я', 2), \  FST::RELATION('а', 2), FST::RELATION('б', 2), FST::RELATION('в', 2), \  FST::RELATION('г', 2), FST::RELATION('д', 2), FST::RELATION('е', 2), \  FST::RELATION('ё', 2), FST::RELATION('ж', 2), FST::RELATION('з', 2), \  FST::RELATION('и', 2), FST::RELATION('й', 2), FST::RELATION('к', 2), \  FST::RELATION('л', 2), FST::RELATION('м', 2), FST::RELATION('н', 2), \  FST::RELATION('о', 2), FST::RELATION('п', 2), FST::RELATION('р', 2), \  FST::RELATION('с', 2), FST::RELATION('т', 2), FST::RELATION('у', 2), \  FST::RELATION('ф', 2), FST::RELATION('х', 2), FST::RELATION('ц', 2), \  FST::RELATION('ч', 2), FST::RELATION('ш', 2), FST::RELATION('щ', 2), \  FST::RELATION('ъ', 2), FST::RELATION('ы', 2), FST::RELATION('ь', 2), \  FST::RELATION('э', 2), FST::RELATION('ю', 2), FST::RELATION('я', 2), \  \  FST::RELATION(' ', 2), FST::RELATION('.', 2), FST::RELATION(',', 2), \  FST::RELATION('?', 2), FST::RELATION('!', 2), FST::RELATION(';', 2), \  FST::RELATION(':', 2), FST::RELATION('-', 2), FST::RELATION(')', 2), \  FST::RELATION('(', 2), FST::RELATION('<', 2), FST::RELATION('>', 2), \  FST::RELATION('/', 2), FST::RELATION('%', 2), FST::RELATION('=', 2), \  FST::RELATION('+', 2), FST::RELATION('\*', 2)), \  FST::NODE(1, FST::RELATION('\'', 3)), \  FST::NODE()  // Функции  #define GRAPH\_MAIN \  5, \  FST::NODE(1,FST::RELATION('m', 1)), \  FST::NODE(1,FST::RELATION('a', 2)), \  FST::NODE(1,FST::RELATION('i', 3)), \  FST::NODE(1,FST::RELATION('n', 4)), \  FST::NODE()  #define GRAPH\_STRDUPLICATE \  13, \  FST::NODE(1,FST::RELATION('s', 1)), \  FST::NODE(1,FST::RELATION('t', 2)), \  FST::NODE(1,FST::RELATION('r', 3)), \  FST::NODE(1,FST::RELATION('d', 4)), \  FST::NODE(1,FST::RELATION('u', 5)), \  FST::NODE(1,FST::RELATION('p', 6)), \  FST::NODE(1,FST::RELATION('l', 7)), \  FST::NODE(1,FST::RELATION('i', 8)), \  FST::NODE(1,FST::RELATION('c', 9)), \  FST::NODE(1,FST::RELATION('a', 10)), \  FST::NODE(1,FST::RELATION('t', 11)), \  FST::NODE(1,FST::RELATION('e', 12)), \  FST::NODE()  #define GRAPH\_STRLENGTH \  10, \  FST::NODE(1,FST::RELATION('s', 1)), \  FST::NODE(1,FST::RELATION('t', 2)), \  FST::NODE(1,FST::RELATION('r', 3)), \  FST::NODE(1,FST::RELATION('l', 4)), \  FST::NODE(1,FST::RELATION('e', 5)), \  FST::NODE(1,FST::RELATION('n', 6)), \  FST::NODE(1,FST::RELATION('g', 7)), \  FST::NODE(1,FST::RELATION('t', 8)), \  FST::NODE(1,FST::RELATION('h', 9)), \  FST::NODE()  #define GRAPH\_STRTRANSINT \  12, \  FST::NODE(1,FST::RELATION('s', 1)), \  FST::NODE(1,FST::RELATION('t', 2)), \  FST::NODE(1,FST::RELATION('r', 3)), \  FST::NODE(1,FST::RELATION('t', 4)), \  FST::NODE(1,FST::RELATION('r', 5)), \  FST::NODE(1,FST::RELATION('a', 6)), \  FST::NODE(1,FST::RELATION('n', 7)), \  FST::NODE(1,FST::RELATION('s', 8)), \  FST::NODE(1,FST::RELATION('i', 9)), \  FST::NODE(1,FST::RELATION('n', 10)), \  FST::NODE(1,FST::RELATION('t', 11)), \  FST::NODE()  // Ключевые слова  #define GRAPH\_RETURN \  7, \  FST::NODE(1,FST::RELATION('r', 1)), \  FST::NODE(1,FST::RELATION('e', 2)), \  FST::NODE(1,FST::RELATION('t', 3)), \  FST::NODE(1,FST::RELATION('u', 4)), \  FST::NODE(1,FST::RELATION('r', 5)), \  FST::NODE(1,FST::RELATION('n', 6)), \  FST::NODE()  #define GRAPH\_NEW \  4, \  FST::NODE(1,FST::RELATION('n', 1)), \  FST::NODE(1,FST::RELATION('e', 2)), \  FST::NODE(1,FST::RELATION('w', 3)), \  FST::NODE()  #define GRAPH\_FUNCTION \  9, \  FST::NODE(1,FST::RELATION('f', 1)), \  FST::NODE(1,FST::RELATION('u', 2)), \  FST::NODE(1,FST::RELATION('n', 3)), \  FST::NODE(1,FST::RELATION('c', 4)), \  FST::NODE(1,FST::RELATION('t', 5)), \  FST::NODE(1,FST::RELATION('i', 6)), \  FST::NODE(1,FST::RELATION('o', 7)), \  FST::NODE(1,FST::RELATION('n', 8)), \  FST::NODE()  // Операторы вывода и цикла  #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()  #define GRAPH\_WRITE \  6, \  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()  #define GRAPH\_CYCLE \  6, \  FST::NODE(1,FST::RELATION('c', 1)), \  FST::NODE(1,FST::RELATION('y', 2)), \  FST::NODE(1,FST::RELATION('c', 3)), \  FST::NODE(1,FST::RELATION('l', 4)), \  FST::NODE(1,FST::RELATION('e', 5)), \  FST::NODE()  // Скобки  #define GRAPH\_LEFTBRACE\_OPEN \  2, \  FST::NODE(1,FST::RELATION('{', 1)), \  FST::NODE()  #define GRAPH\_RIGHTBRACE\_CLOSE \  2, \  FST::NODE(1,FST::RELATION('}', 1)), \  FST::NODE()  #define GRAPH\_LEFTHESIS\_OPEN \  2, \  FST::NODE(1,FST::RELATION('(', 1)), \  FST::NODE()  #define GRAPH\_RIGHTHESIS\_CLOSE \  2, \  FST::NODE(1,FST::RELATION(')', 1)), \  FST::NODE()  // Арифметические операторы  #define GRAPH\_ADDITION \  2, \  FST::NODE(1,FST::RELATION('+', 1)), \  FST::NODE()  #define GRAPH\_SUBSTRACTION \  2, \  FST::NODE(1,FST::RELATION('-', 1)), \  FST::NODE()  #define GRAPH\_MULTIPLICATION \  2, \  FST::NODE(1,FST::RELATION('\*', 1)), \  FST::NODE()  #define GRAPH\_DIVISION \  2, \  FST::NODE(1,FST::RELATION('/', 1)), \  FST::NODE()  #define GRAPH\_REMAINDERDIVISION \  2, \  FST::NODE(1,FST::RELATION('%', 1)), \  FST::NODE()  #define GRAPH\_EQUALS \  2, \  FST::NODE(1,FST::RELATION('=', 1)), \  FST::NODE()  // Операторы сравнения  #define GRAPH\_MORE \  2, \  FST::NODE(1,FST::RELATION('>', 1)), \  FST::NODE()  #define GRAPH\_LESS \  2, \  FST::NODE(1,FST::RELATION('<', 1)), \  FST::NODE()  #define GRAPH\_MOREEQUAL \  3, \  FST::NODE(1,FST::RELATION('>', 1)), \  FST::NODE(1,FST::RELATION('=', 2)), \  FST::NODE()  #define GRAPH\_LESSEQUAL \  3, \  FST::NODE(1,FST::RELATION('<', 1)), \  FST::NODE(1,FST::RELATION('=', 2)), \  FST::NODE()  #define GRAPH\_INEQUALITY \  3, \  FST::NODE(1,FST::RELATION('=', 1)), \  FST::NODE(1,FST::RELATION('=', 2)), \  FST::NODE()  #define GRAPH\_NOTEQUALS \  3, \  FST::NODE(1,FST::RELATION('!', 1)), \  FST::NODE(1,FST::RELATION('=', 2)), \  FST::NODE()  // Разное  #define GRAPH\_COMMA \  2, \  FST::NODE(1,FST::RELATION(',', 1)), \  FST::NODE()  #define GRAPH\_SEMICOLON \  2, \  FST::NODE(1,FST::RELATION(';', 1)), \  FST::NODE() |

Листинг Б – Графы переходов конечных автоматов

# **Приложение В**

Таблица идентификаторов

=========================ID TABLE=========================

+-----+------------+---------------+---------------+---------------+--------------------+

|№ |Name |Type |Data type |First in LT |Value |

|0 |testfunc11 |Function |Int |2 |- |

|1 |a1 |Parameter |Int |5 |- |

|2 |b1 |Parameter |Int |8 |- |

|3 |res10 |Variable |Int |13 |- |

|4 |testfunc22 |Function |Str |26 |- |

|5 |parm12 |Parameter |Str |29 |- |

|6 |parm231 |Variable |Str |34 |- |

|7 |str0 |Literal |Str |36 |"TestText20" |

|8 |testfunc33 |Function |Char |52 |- |

|9 |simbol3 |Parameter |Char |55 |- |

|10 |i57 |Variable |Int |60 |- |

|11 |short1 |Literal |Int |62 |0 |

|12 |short2 |Literal |Int |68 |10 |

|13 |short3 |Literal |Int |78 |1 |

|14 |testfunc44 |Function |Bool |84 |- |

|15 |chetnoe4 |Parameter |Int |87 |- |

|16 |short4 |Literal |Int |93 |2 |

|17 |short5 |Literal |Int |95 |0 |

|18 |str6 |Literal |Str |101 |"Арифметические операции:"|

|19 |a99 |Variable |Int |105 |- |

|20 |b99 |Variable |Int |109 |- |

|21 |short7 |Literal |Int |111 |15 |

|22 |short8 |Literal |Int |115 |10 |

|23 |short9 |Literal |Int |117 |5 |

|24 |short10 |Literal |Int |124 |10 |

|25 |short11 |Literal |Int |126 |5 |

|26 |short12 |Literal |Int |133 |15 |

|27 |short13 |Literal |Int |135 |10 |

|28 |short14 |Literal |Int |142 |10 |

|29 |short15 |Literal |Int |144 |5 |

|30 |short16 |Literal |Int |151 |10 |

|31 |short17 |Literal |Int |153 |5 |

|32 |str18 |Literal |Str |159 |"Вызов функций:" |

|33 |number99 |Variable |Int |163 |- |

|34 |short19 |Literal |Int |167 |0 |

|35 |short20 |Literal |Int |169 |5 |

|36 |parm199 |Variable |Str |177 |- |

|37 |str21 |Literal |Str |179 |"TestText1" |

|38 |string99 |Variable |Str |183 |- |

|39 |alpha99 |Variable |Char |195 |- |

|40 |char22 |Literal |Char |197 |

|41 |myint99 |Variable |Int |206 |- |

|42 |short23 |Literal |Int |208 |120 |

|43 |pravda99 |Variable |Bool |212 |- |

|44 |stroka99 |Variable |Str |224 |- |

|45 |str24 |Literal |Str |226 |"Тестовая строчка" |

|46 |strokachislo99|Variable |Str |230 |- |

|47 |str25 |Literal |Str |232 |"125" |

|48 |copystr99 |Variable |Undefined |235 |- |

|49 |dlina99 |Variable |Int |249 |- |

|50 |testint99 |Variable |Int |261 |- |

|51 |testint299 |Variable |Int |270 |- |

|52 |short26 |Literal |Int |272 |5 |

|53 |str27 |Literal |Str |280 |"Цикл:" |

|54 |i99 |Variable |Int |284 |- |

|55 |short28 |Literal |Int |286 |0 |

|56 |short29 |Literal |Int |292 |10 |

|57 |short30 |Literal |Int |302 |1 |

Таблица лексем

|  |  |
| --- | --- |
| ========LEX TABLE========  +-----+---------+---------+----------+  |# |Line |Lexema |ID from IT|  |0 |0 |d |- |  |1 |0 |f |- |  |2 |0 |i |0 |  |3 |0 |( |- |  |4 |0 |d |- |  |5 |0 |i |1 |  |6 |0 |, |- |  |7 |0 |d |- |  |8 |0 |i |2 |  |9 |0 |) |- |  |10 |1 |{ |- |  |11 |2 |n |- |  |12 |2 |d |- |  |13 |2 |i |3 |  |14 |2 |= |- |  |15 |2 |i |1 |  |16 |2 |+ |- |  |17 |2 |i |2 |  |18 |2 |; |- |  |19 |3 |r |- |  |20 |3 |i |3 |  |21 |3 |; |- |  |22 |4 |} |- |  |23 |4 |; |- |  |24 |6 |s |- |  |25 |6 |f |- |  |26 |6 |i |4 |  |27 |6 |( |- |  |28 |6 |s |- |  |29 |6 |i |5 |  |30 |6 |) |- |  |31 |7 |{ |- |  |32 |8 |n |- |  |33 |8 |s |- |  |34 |8 |i |6 |  |35 |8 |= |- |  |36 |8 |l |7 |  |37 |8 |; |- |  |38 |10 |g |- |  |39 |10 |( |- |  |40 |10 |i |5 |  |41 |10 |, |- |  |42 |10 |i |6 |  |43 |10 |) |- |  |44 |10 |; |- |  |45 |12 |r |- |  |46 |12 |i |5 |  |47 |12 |; |- |  |48 |13 |} |- |  |49 |13 |; |- |  |50 |14 |c |- |  |51 |14 |f |- |  |52 |14 |i |8 |  |53 |14 |( |- |  |54 |14 |c |- |  |55 |14 |i |9 |  |56 |14 |) |- |  |57 |15 |{ |- |  |58 |16 |n |- |  |59 |16 |d |- |  |60 |16 |i |10 |  |61 |16 |= |- |  |62 |16 |l |11 |  |63 |16 |; |- |  |64 |17 |p |- |  |65 |17 |( |- |  |66 |17 |i |10 |  |67 |17 |< |- |  |68 |17 |l |12 |  |69 |17 |) |- |  |70 |17 |{ |- |  |71 |18 |v |- |  |72 |18 |i |9 |  |73 |18 |; |- |  |74 |19 |i |10 |  |75 |19 |= |- |  |76 |19 |i |10 |  |77 |19 |+ |- |  |78 |19 |l |13 |  |79 |19 |; |- |  |80 |20 |} |- |  |81 |21 |} |- |  |82 |22 |b |- |  |83 |22 |f |- |  |84 |22 |i |14 |  |85 |22 |( |- |  |86 |22 |d |- |  |87 |22 |i |15 |  |88 |22 |) |- |  |89 |23 |{ |- |  |90 |24 |r |- |  |91 |24 |i |15 |  |92 |24 |% |- |  |93 |24 |l |16 |  |94 |24 |# |- |  |95 |24 |l |17 |  |96 |24 |; |- |  |97 |25 |} |- |  |98 |26 |m |- |  |99 |27 |{ |- |  |100 |28 |w |- |  |101 |28 |l |18 |  |102 |28 |; |- |  |103 |30 |n |- |  |104 |30 |d |- |  |105 |30 |i |19 |  |106 |30 |; |- |  |107 |31 |n |- |  |108 |31 |d |- |  |109 |31 |i |20 |  |110 |31 |= |- |  |111 |31 |l |21 |  |112 |31 |; |- |  |113 |33 |i |19 |  |114 |33 |= |- |  |115 |33 |l |22 |  |116 |33 |\* |- |  |117 |33 |l |23 |  |118 |33 |; |- |  |119 |34 |v |- |  |120 |34 |i |19 |  |121 |34 |; |- |  |122 |35 |i |19 |  |123 |35 |= |- |  |124 |35 |l |24 |  |125 |35 |/ |- |  |126 |35 |l |25 |  |127 |35 |; |- |  |128 |36 |v |- |  |129 |36 |i |19 |  |130 |36 |; |- |  |131 |37 |i |19 |  |132 |37 |= |- |  |133 |37 |l |26 |  |134 |37 |% |- |  |135 |37 |l |27 |  |136 |37 |; |- |  |137 |38 |v |- |  |138 |38 |i |19 |  |139 |38 |; |- |  |140 |39 |i |19 |  |141 |39 |= |- |  |142 |39 |l |28 |  |143 |39 |+ |- |  |144 |39 |l |29 |  |145 |39 |; |- |  |146 |40 |v |- |  |147 |40 |i |19 |  |148 |40 |; |- |  |149 |41 |i |19 |  |150 |41 |= |- | | |151 |41 |l |30 |  |152 |41 |- |- |  |153 |41 |l |31 |  |154 |41 |; |- |  |155 |42 |v |- |  |156 |42 |i |19 |  |157 |42 |; |- |  |158 |44 |w |- |  |159 |44 |l |32 |  |160 |44 |; |- |  |161 |48 |n |- |  |162 |48 |d |- |  |163 |48 |i |33 |  |164 |48 |= |- |  |165 |48 |i |0 |  |166 |48 |( |- |  |167 |48 |l |34 |  |168 |48 |, |- |  |169 |48 |l |35 |  |170 |48 |) |- |  |171 |48 |; |- |  |172 |49 |v |- |  |173 |49 |i |33 |  |174 |49 |; |- |  |175 |51 |n |- |  |176 |51 |s |- |  |177 |51 |i |36 |  |178 |51 |= |- |  |179 |51 |l |37 |  |180 |51 |; |- |  |181 |52 |n |- |  |182 |52 |s |- |  |183 |52 |i |38 |  |184 |52 |= |- |  |185 |52 |i |4 |  |186 |52 |( |- |  |187 |52 |i |36 |  |188 |52 |) |- |  |189 |52 |; |- |  |190 |53 |w |- |  |191 |53 |i |38 |  |192 |53 |; |- |  |193 |55 |n |- |  |194 |55 |c |- |  |195 |55 |i |39 |  |196 |55 |= |- |  |197 |55 |l |40 |  |198 |55 |; |- |  |199 |56 |i |8 |  |200 |56 |( |- |  |201 |56 |i |39 |  |202 |56 |) |- |  |203 |56 |; |- |  |204 |58 |n |- |  |205 |58 |d |- |  |206 |58 |i |41 |  |207 |58 |= |- |  |208 |58 |l |42 |  |209 |58 |; |- |  |210 |59 |n |- |  |211 |59 |b |- |  |212 |59 |i |43 |  |213 |59 |= |- |  |214 |59 |i |14 |  |215 |59 |( |- |  |216 |59 |i |41 |  |217 |59 |) |- |  |218 |59 |; |- |  |219 |60 |w |- |  |220 |60 |i |43 |  |221 |60 |; |- |  |222 |64 |n |- |  |223 |64 |s |- |  |224 |64 |i |44 |  |225 |64 |= |- |  |226 |64 |l |45 |  |227 |64 |; |- |  |228 |65 |n |- |  |229 |65 |s |- |  |230 |65 |i |46 |  |231 |65 |= |- |  |232 |65 |l |47 |  |233 |65 |; |- |  |234 |66 |n |- |  |235 |66 |i |48 |  |236 |66 |; |- |  |237 |68 |g |- |  |238 |68 |( |- |  |239 |68 |i |48 |  |240 |68 |, |- |  |241 |68 |i |44 |  |242 |68 |) |- |  |243 |68 |; |- |  |244 |69 |w |- |  |245 |69 |i |48 |  |246 |69 |; |- |  |247 |71 |n |- |  |248 |71 |d |- |  |249 |71 |i |49 |  |250 |71 |= |- |  |251 |71 |h |- |  |252 |71 |( |- |  |253 |71 |i |44 |  |254 |71 |) |- |  |255 |71 |; |- |  |256 |72 |w |- |  |257 |72 |i |49 |  |258 |72 |; |- |  |259 |74 |n |- |  |260 |74 |d |- |  |261 |74 |i |50 |  |262 |74 |= |- |  |263 |74 |t |- |  |264 |74 |( |- |  |265 |74 |i |46 |  |266 |74 |) |- |  |267 |74 |; |- |  |268 |75 |n |- |  |269 |75 |d |- |  |270 |75 |i |51 |  |271 |75 |= |- |  |272 |75 |l |52 |  |273 |75 |+ |- |  |274 |75 |i |50 |  |275 |75 |; |- |  |276 |76 |w |- |  |277 |76 |i |51 |  |278 |76 |; |- |  |279 |78 |w |- |  |280 |78 |l |53 |  |281 |78 |; |- |  |282 |80 |n |- |  |283 |80 |d |- |  |284 |80 |i |54 |  |285 |80 |= |- |  |286 |80 |l |55 |  |287 |80 |; |- |  |288 |82 |p |- |  |289 |82 |( |- |  |290 |82 |i |54 |  |291 |82 |< |- |  |292 |82 |l |56 |  |293 |82 |) |- |  |294 |83 |{ |- |  |295 |84 |v |- |  |296 |84 |i |54 |  |297 |84 |; |- |  |298 |85 |i |54 |  |299 |85 |= |- |  |300 |85 |i |54 |  |301 |85 |+ |- |  |302 |85 |l |57 |  |303 |85 |; |- |  |304 |86 |} |- |  |305 |87 |} |- | |

Таблица лексем после построения ПОЛИЗ

|  |  |
| --- | --- |
| =======LEX TABLE========  +-----+---------+---------+----------+  |# |Line |Lexema |ID from IT|  |0 |0 |d |- |  |1 |0 |f |- |  |2 |0 |i |0 |  |3 |0 |( |- |  |4 |0 |d |- |  |5 |0 |i |1 |  |6 |0 |, |- |  |7 |0 |d |- |  |8 |0 |i |2 |  |9 |0 |) |- |  |10 |1 |{ |- |  |11 |2 |n |- |  |12 |2 |d |- |  |13 |2 |i |3 |  |14 |2 |= |- |  |15 |2 |i |1 |  |16 |2 |i |2 |  |17 |2 |+ |- |  |18 |2 |; |- |  |19 |3 |r |- |  |20 |3 |i |3 |  |21 |3 |; |- |  |22 |4 |} |- |  |23 |4 |; |- |  |24 |6 |s |- |  |25 |6 |f |- |  |26 |6 |i |4 |  |27 |6 |( |- |  |28 |6 |s |- |  |29 |6 |i |5 |  |30 |6 |) |- |  |31 |7 |{ |- |  |32 |8 |n |- |  |33 |8 |s |- |  |34 |8 |i |6 |  |35 |8 |= |- |  |36 |8 |l |7 |  |37 |8 |; |- |  |38 |10 |g |- |  |39 |10 |( |- |  |40 |10 |i |5 |  |41 |10 |, |- |  |42 |10 |i |6 |  |43 |10 |) |- |  |44 |10 |; |- |  |45 |12 |r |- |  |46 |12 |i |5 |  |47 |12 |; |- |  |48 |13 |} |- |  |49 |13 |; |- |  |50 |14 |c |- |  |51 |14 |f |- |  |52 |14 |i |8 |  |53 |14 |( |- |  |54 |14 |c |- |  |55 |14 |i |9 |  |56 |14 |) |- |  |57 |15 |{ |- |  |58 |16 |n |- |  |59 |16 |d |- |  |60 |16 |i |10 |  |61 |16 |= |- |  |62 |16 |l |11 |  |63 |16 |; |- |  |64 |17 |p |- |  |65 |17 |( |- |  |66 |17 |i |10 |  |67 |17 |< |- |  |68 |17 |l |12 |  |69 |17 |) |- |  |70 |17 |{ |- |  |71 |18 |v |- |  |72 |18 |i |9 |  |73 |18 |; |- |  |74 |19 |i |10 |  |75 |19 |= |- |  |76 |19 |i |10 |  |77 |19 |l |13 |  |78 |19 |+ |- |  |79 |19 |; |- |  |80 |20 |} |- |  |81 |21 |} |- |  |82 |22 |b |- |  |83 |22 |f |- |  |84 |22 |i |14 |  |85 |22 |( |- |  |86 |22 |d |- |  |87 |22 |i |15 |  |88 |22 |) |- |  |89 |23 |{ |- |  |90 |24 |r |- |  |91 |24 |i |15 |  |92 |24 |% |- |  |93 |24 |l |16 |  |94 |24 |# |- |  |95 |24 |l |17 |  |96 |24 |; |- |  |97 |25 |} |- |  |98 |26 |m |- |  |99 |27 |{ |- |  |100 |28 |w |- |  |101 |28 |l |18 |  |102 |28 |; |- |  |103 |30 |n |- |  |104 |30 |d |- |  |105 |30 |i |19 |  |106 |30 |; |- |  |107 |31 |n |- |  |108 |31 |d |- |  |109 |31 |i |20 |  |110 |31 |= |- |  |111 |31 |l |21 |  |112 |31 |; |- |  |113 |33 |i |19 |  |114 |33 |= |- |  |115 |33 |l |22 |  |116 |33 |l |23 |  |117 |33 |\* |- |  |118 |33 |; |- |  |119 |34 |v |- |  |120 |34 |i |19 |  |121 |34 |; |- |  |122 |35 |i |19 |  |123 |35 |= |- |  |124 |35 |l |24 |  |125 |35 |l |25 |  |126 |35 |/ |- |  |127 |35 |; |- |  |128 |36 |v |- |  |129 |36 |i |19 |  |130 |36 |; |- |  |131 |37 |i |19 |  |132 |37 |= |- |  |133 |37 |l |26 |  |134 |37 |l |27 |  |135 |37 |% |- |  |136 |37 |; |- |  |137 |38 |v |- |  |138 |38 |i |19 |  |139 |38 |; |- |  |140 |39 |i |19 |  |141 |39 |= |- |  |142 |39 |l |28 |  |143 |39 |l |29 |  |144 |39 |+ |- |  |145 |39 |; |- |  |146 |40 |v |- |  |147 |40 |i |19 |  |148 |40 |; |- |  |149 |41 |i |19 |  |150 |41 |= |- |  |151 |41 |l |30 | | |152 |41 |l |31 |  |153 |41 |- |- |  |154 |41 |; |- |  |155 |42 |v |- |  |156 |42 |i |19 |  |157 |42 |; |- |  |158 |44 |w |- |  |159 |44 |l |32 |  |160 |44 |; |- |  |161 |48 |n |- |  |162 |48 |d |- |  |163 |48 |i |33 |  |164 |48 |= |- |  |165 |48 |l |34 |  |166 |48 |l |35 |  |167 |48 |i |0 |  |168 |-1 |# |- |  |169 |-1 |# |- |  |170 |-1 |# |- |  |171 |48 |; |- |  |172 |49 |v |- |  |173 |49 |i |33 |  |174 |49 |; |- |  |175 |51 |n |- |  |176 |51 |s |- |  |177 |51 |i |36 |  |178 |51 |= |- |  |179 |51 |l |37 |  |180 |51 |; |- |  |181 |52 |n |- |  |182 |52 |s |- |  |183 |52 |i |38 |  |184 |52 |= |- |  |185 |52 |i |36 |  |186 |52 |i |4 |  |187 |-1 |# |- |  |188 |-1 |# |- |  |189 |52 |; |- |  |190 |53 |w |- |  |191 |53 |i |38 |  |192 |53 |; |- |  |193 |55 |n |- |  |194 |55 |c |- |  |195 |55 |i |39 |  |196 |55 |= |- |  |197 |55 |l |40 |  |198 |55 |; |- |  |199 |56 |i |8 |  |200 |56 |( |- |  |201 |56 |i |39 |  |202 |56 |) |- |  |203 |56 |; |- |  |204 |58 |n |- |  |205 |58 |d |- |  |206 |58 |i |41 |  |207 |58 |= |- |  |208 |58 |l |42 |  |209 |58 |; |- |  |210 |59 |n |- |  |211 |59 |b |- |  |212 |59 |i |43 |  |213 |59 |= |- |  |214 |59 |i |41 |  |215 |59 |i |14 |  |216 |-1 |# |- |  |217 |-1 |# |- |  |218 |59 |; |- |  |219 |60 |w |- |  |220 |60 |i |43 |  |221 |60 |; |- |  |222 |64 |n |- |  |223 |64 |s |- |  |224 |64 |i |44 |  |225 |64 |= |- |  |226 |64 |l |45 |  |227 |64 |; |- |  |228 |65 |n |- |  |229 |65 |s |- |  |230 |65 |i |46 |  |231 |65 |= |- |  |232 |65 |l |47 |  |233 |65 |; |- |  |234 |66 |n |- |  |235 |66 |i |48 |  |236 |66 |; |- |  |237 |68 |g |- |  |238 |68 |( |- |  |239 |68 |i |48 |  |240 |68 |, |- |  |241 |68 |i |44 |  |242 |68 |) |- |  |243 |68 |; |- |  |244 |69 |w |- |  |245 |69 |i |48 |  |246 |69 |; |- |  |247 |71 |n |- |  |248 |71 |d |- |  |249 |71 |i |49 |  |250 |71 |= |- |  |251 |71 |i |44 |  |252 |71 |h |- |  |253 |-1 |# |- |  |254 |-1 |# |- |  |255 |71 |; |- |  |256 |72 |w |- |  |257 |72 |i |49 |  |258 |72 |; |- |  |259 |74 |n |- |  |260 |74 |d |- |  |261 |74 |i |50 |  |262 |74 |= |- |  |263 |74 |i |46 |  |264 |74 |t |- |  |265 |-1 |# |- |  |266 |-1 |# |- |  |267 |74 |; |- |  |268 |75 |n |- |  |269 |75 |d |- |  |270 |75 |i |51 |  |271 |75 |= |- |  |272 |75 |l |52 |  |273 |75 |i |50 |  |274 |75 |+ |- |  |275 |75 |; |- |  |276 |76 |w |- |  |277 |76 |i |51 |  |278 |76 |; |- |  |279 |78 |w |- |  |280 |78 |l |53 |  |281 |78 |; |- |  |282 |80 |n |- |  |283 |80 |d |- |  |284 |80 |i |54 |  |285 |80 |= |- |  |286 |80 |l |55 |  |287 |80 |; |- |  |288 |82 |p |- |  |289 |82 |( |- |  |290 |82 |i |54 |  |291 |82 |< |- |  |292 |82 |l |56 |  |293 |82 |) |- |  |294 |83 |{ |- |  |295 |84 |v |- |  |296 |84 |i |54 |  |297 |84 |; |- |  |298 |85 |i |54 |  |299 |85 |= |- |  |300 |85 |i |54 |  |301 |85 |l |57 |  |302 |85 |+ |- |  |303 |85 |; |- |  |304 |86 |} |- |  |305 |87 |} |- | |

# **Приложение Г**

Последовательность правил грамматики

|  |  |  |
| --- | --- | --- |
| 0 : S->dfi(F){N};S  4 : F->di,F  7 : F->di  11 : N->nTi=E;N  12 : T->d  15 : E->l  17 : N->cQ{N}e{N}N  18 : Q->(L<L)  19 : L->i  21 : L->l  24 : N->i=E;  26 : E->a(W)  28 : W->i  34 : N->i=E;  36 : E->p(W)  38 : W->i,W  40 : W->i  44 : N->rL;  45 : L->i  49 : S->sfi(F){N};S  53 : F->si  57 : N->rL;  58 : L->i  62 : S->m{N};  64 : N->nTi;N  65 : T->s  68 : N->nTi=E;N  69 : T->s  72 : E->i(W)  74 : W->l  77 : N->oL;N  78 : L->i  80 : N->oL;N  81 : L->l  83 : N->nTi=E;N  84 : T->d  87 : E->Lm  88 : M->/E | 89 : E->l  91 : N->oL;N  92 : L->i  94 : N->oL;N  95 : L->l  97 : N->i=E;N  99 : E->lM  100 : M->\*E  101 : E->(E)M  102 : E->p(W)M  104 : W->l,W  106 : W->l  108 : M->-E  109 : E->l  111 : M->+E  112 : E->l  114 : N->oL;N  115 : L->i  117 : N->i=E;N  119 : E->lM  120 : M->+E  121 : E->l  123 : N->oL;N  124 : L->l  126 : N->oL;N  127 : L->i  129 : N->oL;N  130 : L->l  132 : N->nTi=E;N  133 : T->d  136 : E->i(W)  138 : W->l,W  140 : W->l  143 : N->oL;N  144 : L->i  146 : N->oL;N  147 : L->l  149 : N->nTi;N | 150 : T->d  153 : N->cQ{N}e{N}N  154 : Q->(L]L)  155 : L->i  157 : L->l  160 : N->oL;N  161 : L->l  163 : N->i=E;  165 : E->l  170 : N->oL;N  171 : L->l  173 : N->i=E;  175 : E->l  178 : N->oL;N  179 : L->i  181 : N->nTi;N  182 : T->b  185 : N->cQ{N}e{N}  186 : Q->(L&L)  187 : L->i  189 : L->l  192 : N->oL;  193 : L->l  198 : N->oL;  199 : L->l |

# **Приложение Д**

Листинг структуры данных контекстно-свободной грамматики

|  |
| --- |
| struct Rule // правило в грамматике Грейбах  {  GRBALPHABET nn; // нетерминал (левый символ правила) < 0  int iderror; // идентификатор диагностического сообщения  short size; // количество цепочек - правых частей правила  struct Chain // цепочка (правая часть правила)  {  short size; // длина цепочки  GRBALPHABET\* nt; // цепочка терминалов (> 0) и нетерминалов (< 0)  Chain() { size = 0; nt = 0; };  Chain(  short psize, // количество символов в цепочке  GRBALPHABET s, ... // символы (терминал или нетерминал)  );  char\* getCChain(char\* b); // получить правую сторону правила  static GRBALPHABET T(char t) { return GRBALPHABET(t); }; // терминал  static GRBALPHABET N(char n) { return -GRBALPHABET(n); }; // нетерминал  static bool isT(GRBALPHABET s) { return s > 0; }; // терминал?  static bool isN(GRBALPHABET s) { return !isT(s); }; // нетерминал?  static char alphabet\_to\_char(GRBALPHABET s) { return isT(s) ? char(s) : char(-s); }; // GRBALPHABET->char  }\*chains; // массив цепочек - правых частей правила  Rule() { nn = 0x00; size = 0; };  Rule(  GRBALPHABET pnn, // нетерминал (< 0)  int iderror, // идентификатор диагностического сообщения (Error)  short psize, // количество цепочек - правых частей правила  Chain c, ... // множество цепочек - правых частей правила  );  char\* getCRule( // получить правило в виде N-цепочки (для распечатки)  char\* b, // буфер  short nchain // номер цепочки (правой части) в правиле  );  short getNextChain( // получить следующую за j подходящую цепочку, вернуть её номер или -1  GRBALPHABET t, // первый символ цепочки  Rule::Chain& pchain, // возвращаемая цепочка  short j // номер цепочки  );  };  struct Greibach // грамматика Грейбах  {  short size; // количество правил  GRBALPHABET startN; // стартовый символ  GRBALPHABET stbottomT; // дно стека  Rule\* rules; // множество правил  Greibach() { short 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, возвращающая номер правила или -1  GRBALPHABET pnn, // левый символ  Rule& prule // возвращаемое правило грамматики  );  Rule getRule(short n); // получить правило по номеру  };  Greibach getGreibach(); // получить грамматику  } |

Листинг Д – Структуры данных контекстно-свободной грамматики

# **Приложение Е**

Блок-схема алгоритма синтаксического анализа

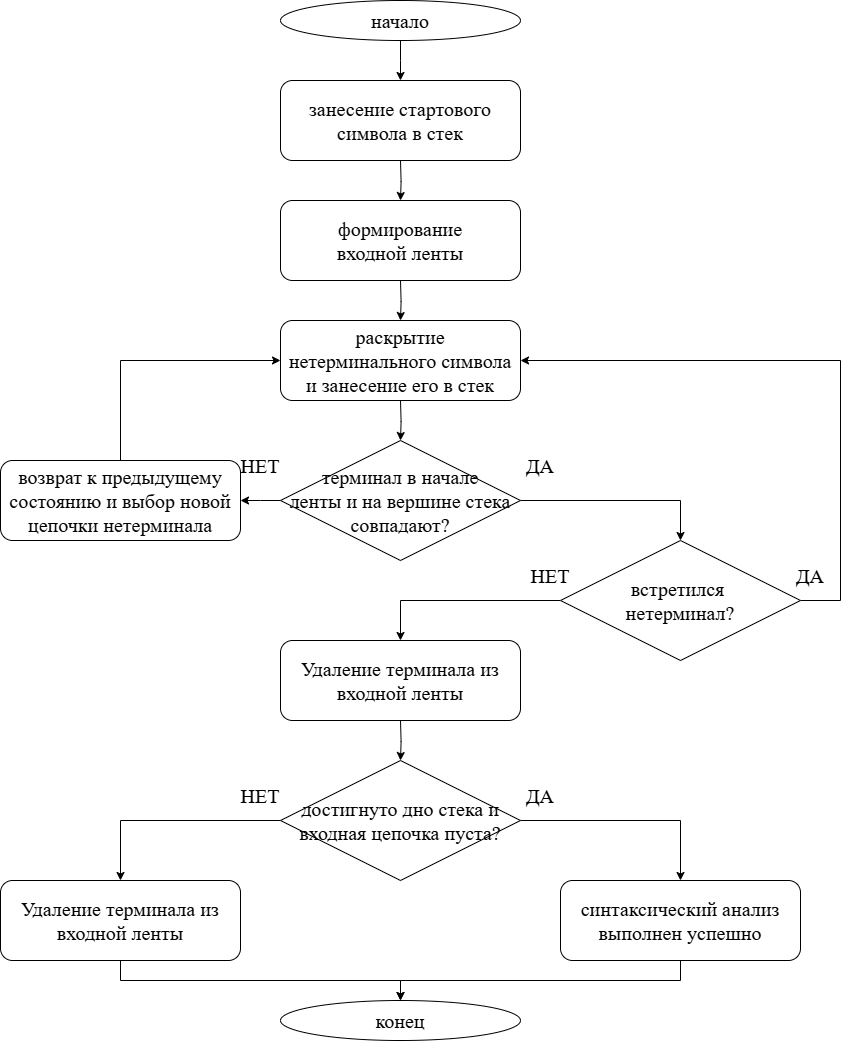


Рисунок Е – Алгоритм синтаксического анализа

# **Приложение Ж**

Листинг результата работы генератора кода

|  |
| --- |
| .386  .model flat, stdcall  .stack 4096  includelib kernel32.lib  includelib user32.lib ; Для использования MessageBoxA  ExitProcess PROTO :DWORD  MessageBoxA PROTO :DWORD, :DWORD, :DWORD, :DWORD  WriteConsoleA PROTO :DWORD, :DWORD, :DWORD, :DWORD, :DWORD  GetStdHandle PROTO :DWORD  .data  message1 db "Типы данных:", 0  message2 db "Арифметические операции:", 0  message3 db "Вызов функций:", 0  message4 db "Цикл:", 0  msgBoxText db "Hello, world!", 0  msgBoxTitle db "Message Box", 0  parm1 db "TestText1", 0  parm2 db "TestText20", 0  stroka db "Тестовая строчка", 0  strokachislo db "125", 0  copystr db 256 dup(0)  hConsoleOutput dd 0 ; Хэндл консоли  .code  ; Функция testfunc1: int testfunc1(int a, int b) {  ; return a + b;  ; }  \_testfunc1 proc  push ebp  mov ebp, esp  mov eax, [ebp+8] ; a  add eax, [ebp+12] ; b  pop ebp  ret  \_testfunc1 endp  ; Функция testfunc2: str testfunc2(str parm1) {  ; strduplicate(parm1, parm2);  ; return parm1;  ; }  \_testfunc2 proc  push ebp  mov ebp, esp  lea eax, parm1  lea edx, parm2  ; Простое копирование через строку в память  mov ecx, 0  \_copy\_string:  mov al, [edx+ecx]  mov [eax+ecx], al  inc ecx  test al, al  jnz \_copy\_string  mov eax, offset parm1  pop ebp  ret  \_testfunc2 endp  ; Функция testfunc3: char testfunc3(char simbol) {  ; cycle (i < 10) { write simbol; }  ; }  \_testfunc3 proc  push ebp  mov ebp, esp  sub esp, 4 ; Выделяем место для i  mov dword ptr [ebp-4], 0 ; i = 0  \_cycle\_start:  cmp dword ptr [ebp-4], 10 ; i < 10 ?  jge \_cycle\_end  ; Отправляем символ в консоль  mov eax, 1 ; Количество символов  lea edx, [ebp+8] ; Адрес символа  invoke WriteConsoleA, hConsoleOutput, edx, eax, 0, 0  inc dword ptr [ebp-4] ; i++  jmp \_cycle\_start  \_cycle\_end:  add esp, 4 ; Убираем i  pop ebp  ret  \_testfunc3 endp  ; Основная функция main  \_main proc  ; Получаем хэндл консоли  invoke GetStdHandle, -11  mov [hConsoleOutput], eax  ; Типы данных  invoke MessageBoxA, 0, offset message1, offset msgBoxTitle, 0  ; Арифметические операции  invoke MessageBoxA, 0, offset message2, offset msgBoxTitle, 0  ; Вызов функций  invoke MessageBoxA, 0, offset message3, offset msgBoxTitle, 0  ; Цикл  invoke MessageBoxA, 0, offset message4, offset msgBoxTitle, 0  ; Завершение программы  invoke ExitProcess, 0  \_main endp  end \_main |

Листинг Ж – Результат генерации кода с LPA-2024 на ассемблер