МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора KAD-2022»

Выполнил студент Кравченко Алексей Дмитриевич

(Ф.И.О.)

Руководитель проекта пр.-ст. Карпович Максим Николаевич

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

Заведующий кафедрой к.т.н., доц. Пацей Наталья Владимировна

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

Консультант пр.-ст. Карпович Максим Николаевич

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

Нормоконтролер пр.-ст. Карпович Максим Николаевич

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

Курсовой проект защищен с оценкой

Минск 2022

**Содержание**

[Введение 3](#_Toc122274077)

[1. Спецификация языка программирования 4](#_Toc122274078)

[1.1 Характеристика языка программирования 4](#_Toc122274079)

[1.2 Алфавит языка 4](#_Toc122274080)

[1.3 Применяемые сепараторы 4](#_Toc122274081)

[1.4 Применяемые кодировки 5](#_Toc122274082)

[1.5 Типы данных 5](#_Toc122274083)

[1.6 Преобразование типов данных 6](#_Toc122274084)

[1.7 Идентификаторы 6](#_Toc122274085)

[1.8 Литералы 6](#_Toc122274086)

[1.9 Объявления данных и область видимости 7](#_Toc122274087)

[1.10 Инициализация данных 7](#_Toc122274088)

[1.11 Инструкции языка 7](#_Toc122274089)

[1.12 Операции языка 8](#_Toc122274090)

[1.13 Выражения и их вычисления 8](#_Toc122274091)

[1.14 Программные конструкции языка 9](#_Toc122274092)

[1.15 Область видимости идентификаторов 10](#_Toc122274093)

[1.16 Семантические проверки 10](#_Toc122274094)

[1.17 Распределение оперативной памяти на этапе выполнения 10](#_Toc122274095)

[1.18 Стандартные библиотеки и их состав 11](#_Toc122274096)

[1.19 Ввод и вывод данных 12](#_Toc122274097)

[1.20 Точка входа 12](#_Toc122274098)

[1.21 Препроцессор 12](#_Toc122274099)

[1.22 Соглашения о вызовах 12](#_Toc122274100)

[1.23 Объектный код 12](#_Toc122274101)

[1.24 Классификация сообщений транслятора 13](#_Toc122274102)

[1.25 Контрольный пример 13](#_Toc122274103)

[2. Структура транслятора 14](#_Toc122274104)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 14](#_Toc122274105)

[2.2 Перечень входных параметров транслятора 15](#_Toc122274106)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 16](#_Toc122274107)

[3. Разработка лексического анализатора 17](#_Toc122274108)

[3.1 Структура лексического анализатора 17](#_Toc122274109)

[3.2. Контроль входных символов 18](#_Toc122274110)

[3.3 Удаление избыточных символов 18](#_Toc122274111)

[3.4 Перечень ключевых слов 19](#_Toc122274112)

[3.5 Основные структуры данных 21](#_Toc122274113)

[3.6 Принцип обработки ошибок 22](#_Toc122274114)

[3.7 Структура и перечень сообщений лексического анализатора 22](#_Toc122274115)

[3.8 Параметры лексического анализатора 22](#_Toc122274116)

[3.9 Алгоритм лексического анализа 22](#_Toc122274117)

[3.10 Контрольный пример 23](#_Toc122274118)

[4. Разработка синтаксического анализатора](#_Toc122274128) 24

[4.1 Структура синтаксического анализатора 24](#_Toc122274119)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 24](#_Toc122274120)

[4.3 Построение конечного магазинного автомата 26](#_Toc122274121)

[4.4 Основные структуры данных 26](#_Toc122274122)

[4.5 Описание алгоритма синтаксического разбора 27](#_Toc122274123)

[4.6 Структура и перечень сообщений синтаксического анализатора 27](#_Toc122274124)

[4.7. Параметры синтаксического анализатора и режимы его работы 27](#_Toc122274125)

[4.8. Принцип обработки ошибок 28](#_Toc122274126)

[4.9. Контрольный пример 28](#_Toc122274127)

[5. Разработка семантического анализатора 29](#_Toc122274128)

[5.1 Структура семантического анализатора 29](#_Toc122274129)

[5.2 Функции семантического анализатора 29](#_Toc122274130)

[5.3 Структура и перечень сообщений семантического анализатора 29](#_Toc122274131)

[5.4 Принцип обработки ошибок 30](#_Toc122274132)

[5.5 Контрольный пример 30](#_Toc122274133)

[6. Вычисление выражений 31](#_Toc122274128)

[6.1 Выражения, допускаемые языком 31](#_Toc122274134)

[6.2 Польская запись и принцип её построения 31](#_Toc122274135)

[6.3 Программная реализация обработки выражений 31](#_Toc122274136)

[6.4 Контрольный пример 32](#_Toc122274137)

[7. Генерация кода 33](#_Toc122274138)

[7.1 Структура генератора кода 33](#_Toc122274139)

[7.2 Представление типов данных в оперативной памяти 33](#_Toc122274140)

[7.3 Статическая библиотека 34](#_Toc122274141)

[7.4 Особенности алгоритма генерации кода 34](#_Toc122274142)

[7.5 Входные параметры генератора кода 35](#_Toc122274143)

[7.6 Контрольный пример 35](#_Toc122274144)

[8. Тестирование транслятора 36](#_Toc122274145)

[8.1 Тестирование проверки на допустимость символов 36](#_Toc122274146)

[8.2 Тестирование лексического анализатора 36](#_Toc122274147)

[8.3 Тестирование синтаксического анализатора 37](#_Toc122274148)

[8.4 Тестирование семантического анализатора 38](#_Toc122274149)

[Заключение](#_Toc122274145) 40

[Список использованных источников 41](#_Toc122274150)

[Приложение А 42](#_Toc122274151)

[Приложение Б 43](#_Toc122274152)

[Приложение В 44](#_Toc122274153)

[Приложение Г 63](#_Toc122274154)

[Приложение Д 66](#_Toc122274155)

[Приложение E 69](#_Toc122274156)

[Приложение Ж 70](#_Toc122274157)

**Введение**

В данном курсовом проекте поставлена задача разработки собственного языка программирования и транслятора для него. Название языка – KAD-2022. Написание транслятора будет осуществляться на языке C++, при этом код на языке KAD-2022 будет транслироваться в язык ассемблера.

Задание на курсовой проект можно разделить на следующие задачи:

1. Разработка спецификации языка KAD-2022;
2. Разработка лексического анализатора;
3. Разработка синтаксического анализатора;
4. Разработка семантического анализатора;
5. Разбор арифметических выражений;
6. Разработка генератора кода;
7. Тестирование транслятора.
8. Разработка графического интерфейса

1. Спецификация языка программирования

* 1. Характеристика языка программирования

Язык программирования KAD-2022 является процедурным, строго типизируемым, транслируемым на язык assembler.

* 1. Алфавит языка

Алфавит KAD-2022 состоит из символов, приведённых на Рисунке 1.1. Допустимо использование символов латинского алфавита, кириллицы, символов.

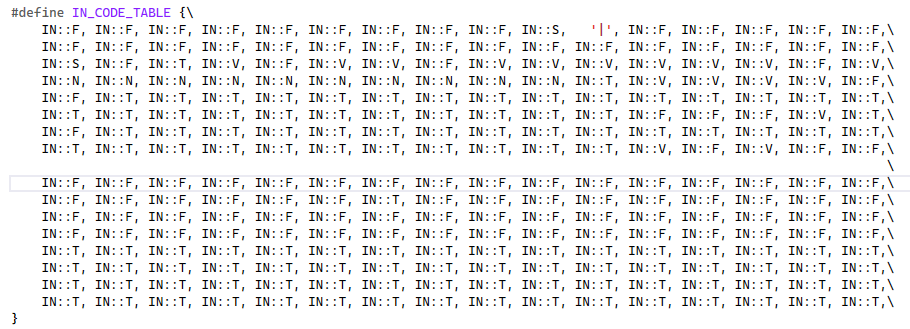


Рисунок 1.1 Символы, разрешенные к использованию

* 1. Применяемые сепараторы

Сепараторы служат для разделения операций языка. Сепараторы, применяемые в языке программирования KAD-2022, приведены в таблице 1.1.

Таблица 1.1 - Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение сепаратора |
| ; | Разделитель инструкций |
| { } | Программный блок |
| ( ) | Параметры, приоритетность операций в выражениях |
| пробел | Допускается везде, кроме идентификаторов и ключевых слов. Служит для их разделения |
| , | Разделитель параметров в функции |

Сепараторы – программные конструкции, позволяющие разделять программные блоки кода.

* 1. **Применяемые кодировки**

Кодировка, используемая для написания программ на языке KAD-2022 - стандартная кодировка Windows-1251, представленная на Рисунке 1.2.

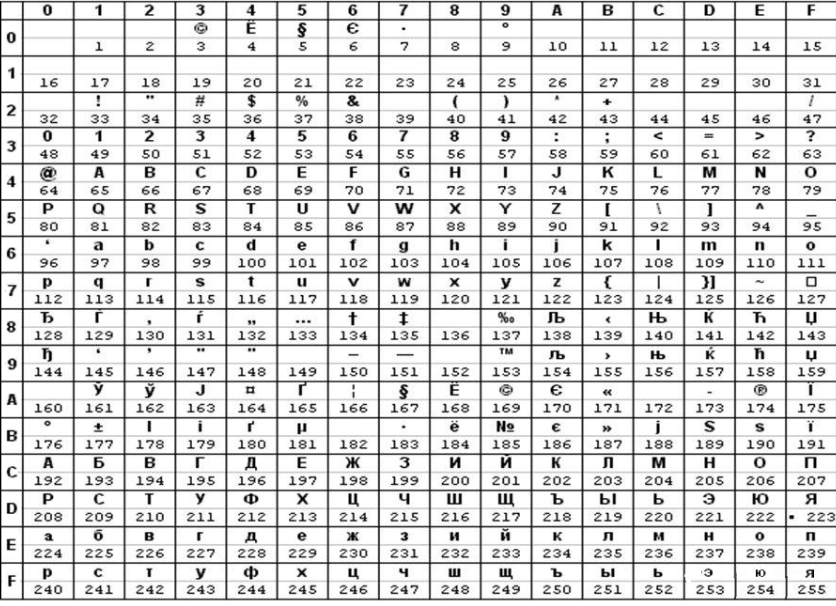


Рисунок 1.2 Таблица кодировки Windows 1251

Windows 1251 – это 8-битная кодировка символов, предназначенная для покрытия языков, использующих кириллицу.

* 1. Типы данных

Допускается использование фундаментальных типов данных, определенных в таблице 1.2.

Под типом данных понимают описание данных определенного вида, для которых известен способ представления в памяти (формат), следующие из него размерность и диапазон значений, а также определенный спектр операций.

Таблица 1.2 – Фундаментальные типы данных

|  |  |
| --- | --- |
| Тип данных | Описание |
| **int** | Фундаментальный тип данных. Предусмотрен для объявления целочисленных данных (4 байта). Максимальное значение: 2147483647, минимальное значение: -2147483648.  Автоматически инициализируется нулевым значением. |
| **str** | Фундаментальный тип данных. Предусмотрен для объявления строк. (1 символ – 1 байт).  Автоматическая инициализация строкой нулевой длины. Максимальное количество символов в строке – 255.  Первый байт – длина строки. |

Пользовательские типы данных не поддерживаются.

* 1. **Преобразование типов данных**

Преобразование типов данных в языке KAD-2022 не предусмотрено. Попытка преобразования типов данных приведет к ошибке.

**1.7 Идентификаторы**

Идентификаторы могут выступать в качестве имен функций, параметров, переменных. Зарезервированные идентификаторы не предусмотрены. Идентификаторы не должны совпадать с ключевыми словами.

Имя идентификатора составляется по следующим правилам:

* состоит из символов [a..z] и [A..Z] и ‘ ’;
* длина идентификатора не должна превышать 20 символов. При превышении максимально допустимой длины применяется усечение.

**1.8 Литералы**

В языке программирования KAD-2022 предусмотрены целочисленные и строковые литералы. Все литералы являются rvalue. Их правила записи приведены в таблице 1.3.

Литералы представляют неизменяемые значения (иногда их еще называют константами). Литералы можно передавать переменным в качестве значения. Литералы бывают различных типов.

Rvalue — это выражение, которое не представляет собой объект, который занимает идентифицируемое место в памяти.

Таблица 1.3 – Правила записи литералов

|  |  |
| --- | --- |
| Тип литерала | Описание литерала |
| Целочисленный | KAD-2022 поддерживает представление литералов в следующих системах счисления:  двоичная: [0, 1], начинается с префикса ‘b’;  восьмеричная: [0..7], начинается с префикса ‘o’;  десятичная: [0..9], без префикса;  шестнадцатеричная: [0..9], [A..F], начинается с префикса ‘h’.  При выходе за пределы будет произведен сдвиг. |
| Строковый | Символы, заключенные в “…” (двойные кавычки), число которых не превышает 255. |

Значения целочисленных литералов, выходящих за пределы допустимых значений, автоматически сдвигаются. Значения строковых литералов, у которых количество элементов строки превышает 255 символов – автоматически урезается.

**1.9** **Объявления данных и область видимости**

Для объявления переменной указывается тип данных, далее используется ключевое слово **var**, после чего имя идентификатора.

Для объявления функций используется ключевое слово **function**, перед которым указывается тип функции, а после – идентификатор.

Область видимости реализована Си-подобным образом: сверху-вниз. Все функции и процедуры имеют глобальную область видимости и могут быть объявлены только в глобальной области видимости. Переменные не могут быть глобальными. Любые идентификаторы должны быть объявлены до их использования.

**1.10 Инициализация данных**

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

**1.11 Инструкции языка**

В языке программирования KAD-2022 применяются инструкции, представленные в Таблице 1.4.

Таблица 1.4 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция языка | Синтаксис |
| Объявление переменной | <тип данных> var <идентификатор>;  <тип данных>var<идентификатор> = <литерал>|<выражение>|<идентификатор> |
| Присваивание | <идентификатор> = <литерал>|<выражение>|<идентификатор>; |
| Вызов функций | <идентификатор функции> ([параметр 1] [, параметр 2]); |
| Вывод данных в консоль | write <литерал>|<идентификатор>; |
| Возврат из функции | return <литерал>|<идентификатор>; |

**1.12 Операции языка**

Язык программирования KAD-2022 может выполнять арифметические операции и операции сравнения, представленные в таблице 1.5.

Таблица 1.5 - Приоритетности операций языка программирования KAD-2022

|  |  |  |
| --- | --- | --- |
| Операция | Арифметическое назначение | Приоритетность операции |
| () | Приоритетность операций | 0 |
| + | Сложение | 1 |
| - | Вычитание | 1 |
| \* | Умножение | 2 |
| / | Деление | 2 |
| % | Взятие остатка от деления | 2 |
| > | Сравнение чисел | 3 |
| < | Сравнение чисел | 3 |
| ^ | Сравнение чисел | 3 |
| & | Сравнение чисел | 3 |

Каждая операция имеет различную приоритетность.

**1.13 Выражения и их вычисления**

Выражения относятся только к целому типу.

Предусмотрены следующие правила составления выражений:

* выражения записываются до ввода сепаратора ‘;’;
* выражение может содержать вызов функций;
* рассматриваются слева направо;
* для изменения приоритета операция используются круглые скобки ();
* реализация выражений происходит с помощью обратной польской записи.

**1.14 Программные конструкции языка**

Программные конструкции представлены в Таблице 1.7.

Таблица 1.6 – Программные конструкции языка

|  |  |
| --- | --- |
| Конструкция | Представление в языке |
| Главная функция | main  {  <инструкции языка>  } |
| Функция | <тип данных> function <идентификатор>([<тип данных> param <идентификатор>][, <тип данных> param <идентификатор>])  {  <инструкции языка>  return <идентификатор>|<литерал>  } |
| Цикл | repeat(<идентификатор1>|<литерал>  <логический оператор><идентификатор2>|<литерал>)  {  <инструкции языка>  } |
| Условный оператор | if (<идентификатор1>|<литерал>  <логический оператор><идентификатор2>|<литерал>)  then{<инструкции языка>}  else{<инструкции языка>} |

Программные конструкции языка KAD-2022 представляют собой базовый функционал для выполнения различных операций, что делает возможным решать задачи различного уровня.

**1.15 Область видимости идентификаторов**

Все идентификаторы, объявленные внутри функции, являются локальными. Параметры видны только внутри функции, в которой объявлены.

**1.16 Семантические проверки**

Перечень семантических проверок, предусмотренных языком, приведен в Таблице 1.7.

Таблица 1.7 - Перечень семантических проверок

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Деление на ноль |
| 2 | Совпадение типов данных |
| 3 | Корректность строковых выражений справа от знака равно |
| 4 | Проверка на возвращающее значение из функции |
| 5 | Передаваемые параметру в функцию при вызове |
| 6 | Корректность условного выражения |
| 7 | Переопределение типов данных |
| 8 | Выход за пределы области видимости |
| 9 | Усечение слишком длинного значения str-литерала |
| 10 | Округление слишком большого значения int-литерала |
| 11 | Точка входа main |
| 12 | Проверка типа идентификаторов |
| 13 | Проверка лексем |
| 14 | Если ошибка возникает на этапе лексического анализа, синтаксический  анализ не выполняется |
| 15 | Если 3 подряд фразы не разобраны, то работа транслятора останавливается |
| 16 | При возникновении ошибки в процессе синтаксического анализа,  ошибочная фраза игнорируется (предполагается, что ее нет) и  осуществляется попытка разбора следующей фразы. Граница фразы –  точка с запятой. |

Назначение семантического анализа – проверка смысловой правильности конструкций языка программирования.

**1.17 Распределение оперативной памяти на этапе выполнения**

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

**1.18 Стандартные библиотеки и их состав**

В языке KAD-2022 присутствует стандартная статическая библиотека:

StandLib, подключается автоматически при трансляции кода в ассемблер. Функции стандартной библиотеки и их описание представлено в таблице 1.8.

Таблица 1.8 - Библиотека StandartLib языка KAD-2022

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| length | integer | str parm1 - строка | Функция вычисляет длину строки parm1 |
| copy | string | str parm1 – строка  str parm2 – строка  int count - цифра | Функция выполняет копирование определенного количества символов из parm2 в строку parm1 |
| write | отсутствует | integer <идентификатор> | <целочисленный литерал> | Функция выводит на консоль <выражение> |
| write | отсутствует | str <идентификатор> | <строковый литерал> | Функция выводит на консоль <выражение> |
| getLocalTimeAndDate | string | Без параметров | Функция возвращает текущее локальное время и дату |
| powNumber | integer | int num1 – цифра  int num2 - цифра | Функция выполняет возведение значения num1 в степень num2 |

Таблица 1.8 (продолжение) - Библиотека StandartLib языка KAD-2022

|  |  |  |  |
| --- | --- | --- | --- |
| random | integer | int start – цифра  int end – цифра | Функция генерирует случайное числовое значение в диапазоне от start до end |
| factorialOfNumber | integer | int num - цифра | Функция возвращает факториал заданного числа num |
| squareOfNumber | integer | int num - цифра | Функция извлекает квадратный корень из num и возвращает значение |

**1.19 Ввод и вывод данных**

Ввод данных в языке KAD-2022 не предусмотрен. Вывод данных осуществляется с помощью ключевого слова “write”. Есть возможность выводить значение переменных и строк, которые будут заключены в двойные кавычки.

## 1.20 Точка входа

Точкой входа в программе является ключевое слово “main”. Точка входа не может отсутствовать или быть переопределена.

**1.21 Препроцессор**

Препроцессор в языке программирования KAD-2022 не предусмотрен.

**1.22 Соглашения о вызовах**

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

**1.23 Объектный код**

Язык KAD-2022 транслируется в исходный код для языка ассемблера.

**1.24 Классификация сообщений транслятора**

Классификация ошибок представлена в таблице 1.9.

Таблица 1.9 - Классификация ошибок

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

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

**1.25 Контрольный пример**

Контрольный пример демонстрирует некоторые особенности языка KAD-2022: его фундаментальные типы, основные структуры, использование пользовательских функций, использование функций статической библиотеки, программные конструкции языка. Исходный код контрольного примера представлен в приложении **А.**

# **2. Структура транслятора**

## 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

В языке KAD-2022 исходный код транслируется в язык Assembler. Транслятор языка разделён на отдельные части, которые взаимодействуют между собой и выполняют отведённые им функции, которые представлены в пункте 2.1. Для того чтобы получить ассемблерный код, используется выходные данные работы лексического анализатора, а именно таблица лексем и таблица идентификаторов. Для указания выходных файлов используются входные параметры транслятора, которые описаны в таблице 2.1. Структура транслятора языка KAD-2022 приведена на рисунке 1.

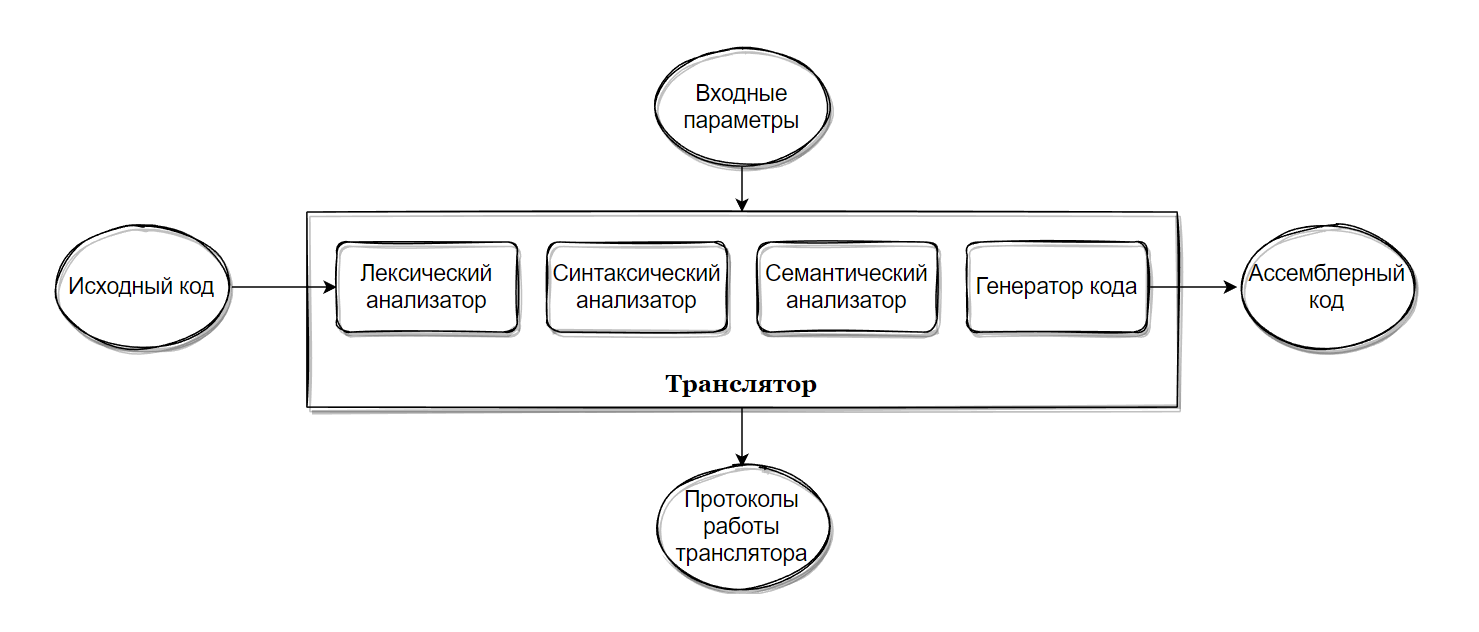


Рисунок 2.1 Структура транслятора языка KAD-2022

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

Цели лексического анализатора:

− убрать все лишние пробелы;

− выполнить распознавание лексем;

− построить таблицу лексем и таблицу идентификаторов;

− при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

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

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

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

## 2.2 Перечень входных параметров транслятора

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

Таблица 2.1 - Входные параметры транслятора языка KAD-2022

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке KAD-2022 , имеющий расширение .txt | Не предусмотрено |
| -log:<путь к log-файлу> | Файл журнала для вывода протоколов работы программы. | Значение по умолчанию:  <имя in-файла>.log |
| -out:<путь к out-файлу> | Выходной файл – результат работы транслятора. Содержит исходный код на языке асемблера. | Значение по умолчанию:  <имя in-файла>.asm |

Входные параметры указываются через командную строку вручную, в графическом интерфейсе – автоматически.

## 2.3 Перечень протоколов, формируемых транслятором и их содержимое

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

Таблица 2.2 - Протоколы, формируемые транслятором языка KAD-2022

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования KAD-2022. Содержит информацию про входные параметры, общем количестве символов и строк(исходные данные), протокол работы синтаксического анализатора, полученный на этапе синтаксического анализа. |
| Выходной файл с названием "Table.id.txt" | Файл содержит таблицу идентификаторов, сформированную во время лексического анализа. |
| Выходной файл с названием "Table.lex.txt" | Файл содержит таблицу лексем, сформированную во время лексического анализа. |
| Выходной файл, заданный параметром "-out:" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

Все файлы создаются в корневом каталоге.

# **3. Разработка лексического анализатора**

## 3.1 Структура лексического анализатора

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

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

Функции лексического анализатора:

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

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

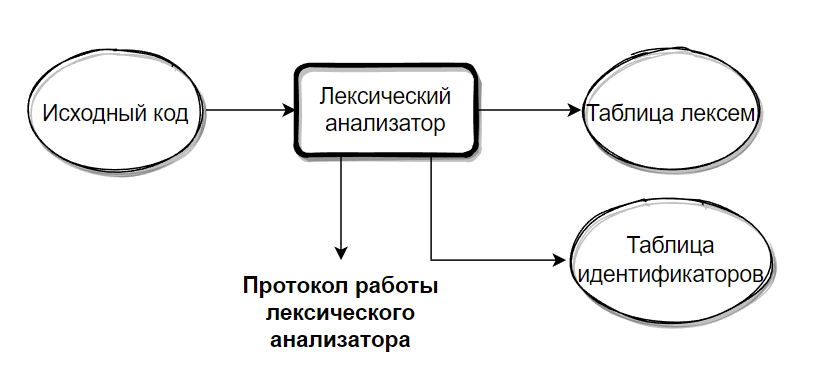


Рисунок 3.1 Структура лексического анализатора

## 3.2. Контроль входных символов

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

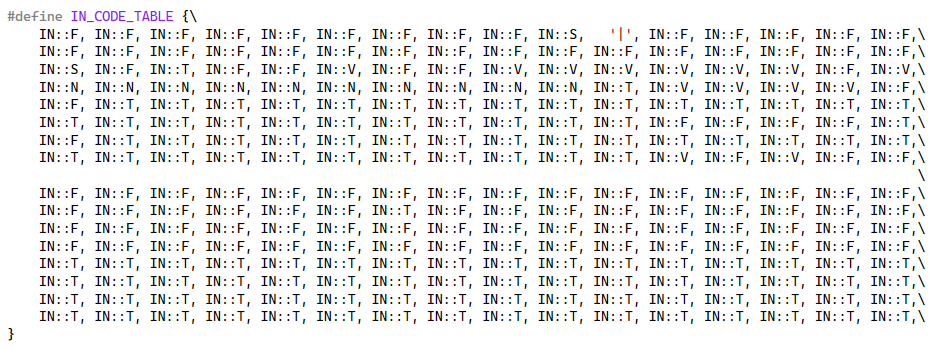


Рисунок 3.2 Таблица контроля входных символов

Таблица 3.1 - Соответствие символов и их значений в таблице

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

## 3.3 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на токены.

Описание алгоритма удаления избыточных символов:

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

## 3.4 Перечень ключевых слов

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

Таблица 3.2 - Соответствие токенов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| int, string | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 20 символов. |
| Литерал | l | Литерал любого доступного типа. |
| function | f | Объявление функции. |
| var | v | Объявление переменной |
| param | p | Объявление параметра функции |
| return | r | Выход из функции/процедуры. |
| main | m | Главная функция. |
| write | w | Вывод данных |
| if | ? | Уловный оператор. |
| then | : | Истинная ветвь условного оператора. |
| else | ! | Ложная ветвь условного оператора. |
| repeat | ~ | Указывает на начало тела цикла. |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| { | { | Начало блока/тела функции. |
| } | } | Закрытие блока/тела функции. |
| ( | ( | Передача параметров в функцию, приоритет операций. |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | = | Знак присваивания. |
| +  -  \*  /  % | # | Знаки операций. |
| >  <  ^  & | >  <  ^  & | Знаки логических операторов. |
| # | нет | Комментарии. |

Пример реализации таблицы лексем представлен в приложении Б.

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

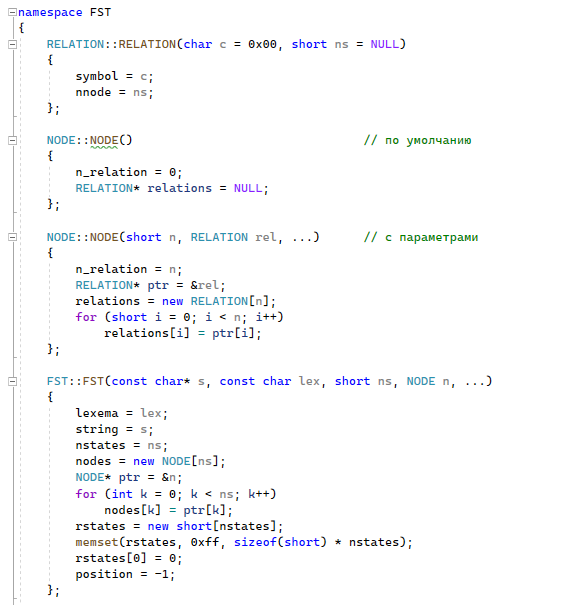


Рисунок 3.3 Структура конечного автомата

Пример графа перехода конечного автомата изображен на рисунке 3.4.

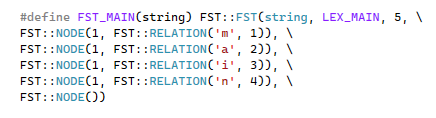


Рисунок 3.4 Пример реализации графа КА для токена main

## 3.5 Основные структуры данных

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

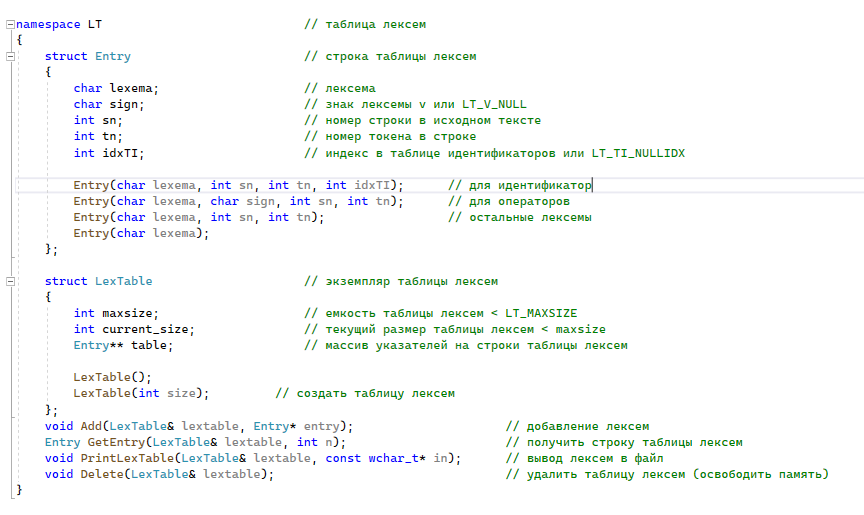


Рисунок 3.5 Структура таблицы лексем

Таблица идентификаторов содержит имя идентификатора (id), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype) и значение (или параметры функций) (value). Код C++ со структурой таблицы идентификаторов представлен на рисунке 3.6.

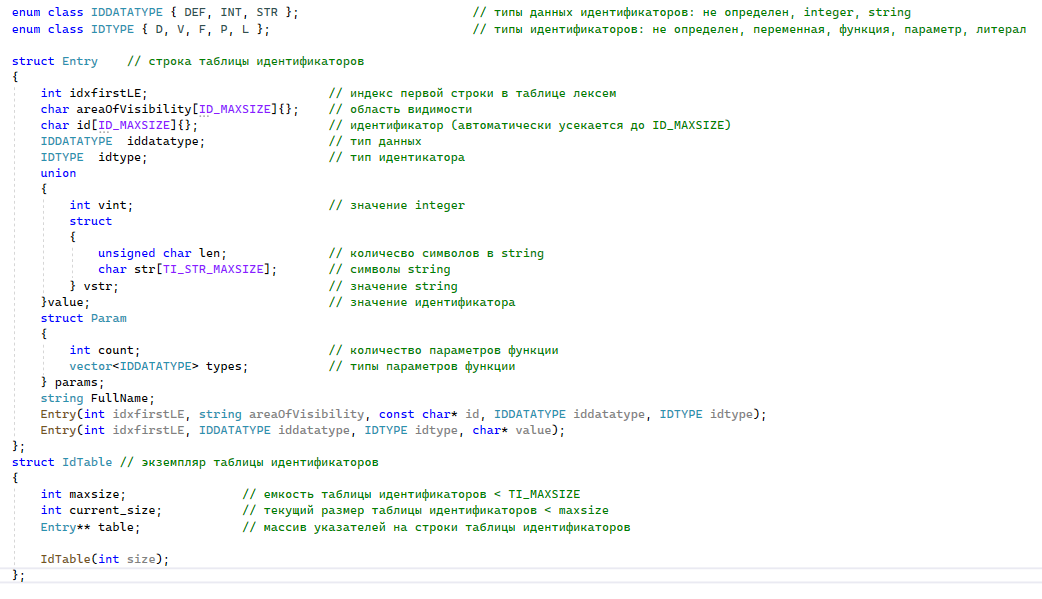


Рисунок 3.6 Структура таблицы идентификаторов

## 3.6 Принцип обработки ошибок

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. Перечень сообщений представлен на рисунке 3.7.

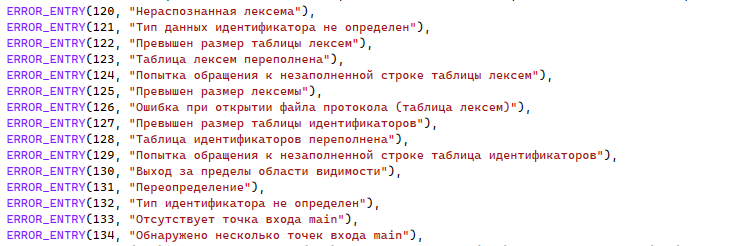


Рисунок 3.7 Сообщения лексического анализатора

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

## 3.7 Структура и перечень сообщений лексического анализатора

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

## 3.8 Параметры лексического анализатора

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся в два отдельных файла – lextable.txt и idtable.txt .

## 3.9 Алгоритм лексического анализа

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

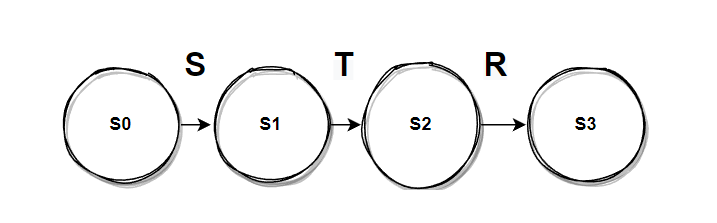


Рисунок 3.2 Пример графа переходов для цепочки **str**

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

## 3.10 Контрольный пример

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

4. Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора

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

Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

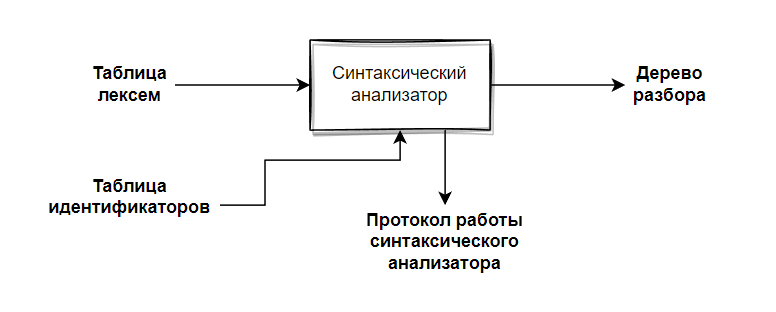


Рисунок 4.1 Структура синтаксического анализатора

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

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

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

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

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

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

S – начальный символ грамматики, являющийся нетерминалом.

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

Описание нетерминальных символов содержится в таблице 4.1.

Таблица 4.1 - Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| S | S-> m{N}S  S-> tfi(F){NrU;}S  S-> λ | Стартовые правила, описывающее общую структуру программы |
| N | N-> tY;N  N-> i=E;N  N-> wE;N  N-> ~KZN  N-> ?KJN  N-> λ | Правила для операторов |
| U | U-> l  U-> i  U->lU  U->iU  U->i,U  U->l,U | Только литерал или идентификатор (один или несколько) |
| J | J-> :{N}  J-> :{N}!{N} | Правила условного перехода |
| Y | Y-> vi  Y->vi=L | Правило определения переменной |
| K | K-> (U<U)  K-> (U>U) | Правила определения условия перехода |
| E | E-> iM  E-> lM  E-> (E)M  E-> i(W)M | Правила выражений |
| M | M-> #E  M-> λ | Правила арифметический операторов |
| F | F-> tP  F-> tP,F | Правила определения параметров функции |
| P | P-> pi | Правило определения параметров функции |
| W | W-> l  W-> i  W-> l,W  W-> i,W | Правила вызова функции с параметрами |
| L | L->i  L->l  L->i(U)  L->i() | Начальная инициализация |
| Z | Z->{N} | Тело цикла |

## 4.3 Построение конечного магазинного автомата

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

Таблица 4.2 – Описание компонентов магазинного автомата

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

## 4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка KAD-2022. Данные структуры в приложении В.

## 4.5 Описание алгоритма синтаксического разбора

Принцип работы автомата, следующий:

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

## 4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен на рисунке 4.3.

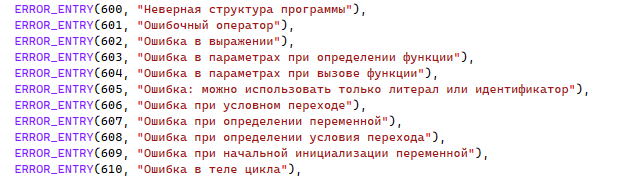


Рисунок 4.3 Сообщения синтаксического анализатора

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

## 4.7. Параметры синтаксического анализатора и режимы его работы

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

## 4.8. Принцип обработки ошибок

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

## 4.9. Контрольный пример

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

# **5. Разработка семантического анализатора**

## 5.1 Структура семантического анализатора

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

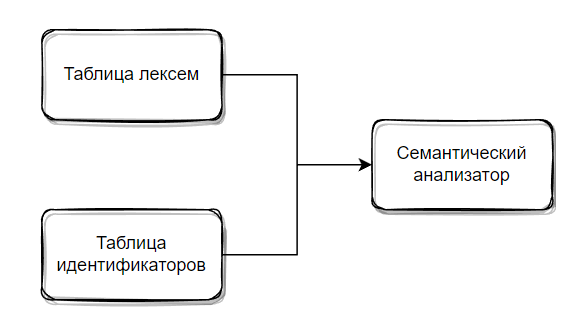


Рисунок 5.1 Структура семантического анализатора

## 5.2 Функции семантического анализатора

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

## 5.3 Структура и перечень сообщений семантического анализатора

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.1.

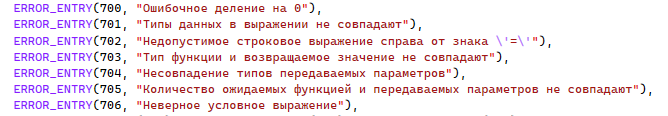


Рисунок 5.2 Перечень сообщений семантического анализатора.

На этапе семантического анализа предусмотрено возникновение 6 возможных ошибок.

## 5.4 Принцип обработки ошибок

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

## 5.5 Контрольный пример

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

Таблица 5.1 - Примеры диагностики ошибок

|  |  |
| --- | --- |
| Исходный код | Текст сообщения |
| main{  int var x = 15;  x = x + “string”;  } | Ошибка на этапе Семантического анализатора  Ошибка 701: Типы данных в выражении не совпадают, строка 3 |
| main{  int var x;  x = 5 / 0;  } | Ошибка на этапе Семантического анализатора  Ошибка 700: Деление на 0, строка 3, лексема 5 |
| main{  str var y;  y = "test2" + “test3”;  } | Ошибка на этапе Семантического анализатора  Ошибка 702: Недопустимое строковое выражение справа от знака '=', строка 3 |

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

6. Вычисление выражений

## 6.1 Выражения, допускаемые языком

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

Таблица 6.1 - Приоритеты операций

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

## 6.2 Польская запись и принцип её построения

Все выражения языка KAD-2022 преобразовываются к обратной польской записи.

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

исходная строка: выражение;

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

## 6.3 Программная реализация обработки выражений

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

## 6.4 Контрольный пример

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

Таблица 6.2 - Преобразование выражений к ПОЛИЗ

|  |  |
| --- | --- |
| Выражение | Обратная польская запись для выражения |
| l | l |
| i#l | il# |
| i#l | il# |

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

# **7. Генерация кода**

## 7.1 Структура генератора кода

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

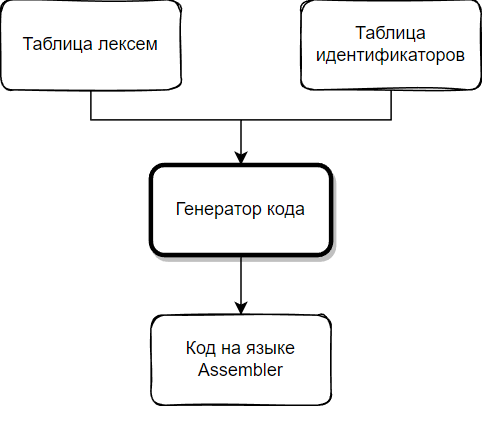


Рисунок 7.1 Структура генератора кода

## 7.2 Представление типов данных в оперативной памяти

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

Таблица 7.1 – Соответствия типов идентификаторов языка KAD-2022 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке KAD-2022 | Тип идентификатора на языке ассемблера | Пояснение |
| int | sdword | Хранит целочисленный тип данных. |
| str | dword | Хранит указатель на начало строки. Строка должна завершаться символом нуль-терминатор. |

## 7.3 Статическая библиотека

В языке KAD-2022 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера.

Таблица 7.3 – Функции статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| void write\_str(char\* str) | Вывод на консоль строки str |
| void write\_int(int num) | Вывод на консоль целочисленной переменной num |
| int length (char\* str) | Вычисление длины строки |
| char\* copy (char\* str1, char\* str2, int count) | Копирование определенного количества символов из str2 в строку str1 |
| int squareOfNumber(int num) | Извлекает квадратный корень из числа num |
| int factorialOfNumber(int num) | Вычисляет факториал заданного числа |
| int random(int start, int end) | Генерирует случайное значение в диапазоне от strat до end |
| int powNumber(int num, int num2) | Возводит числовое значение num в степень num2 |
| char\* getLocalTimeAndDate() | Возвращает текущее локальное время и дату |

Объявление функций статической библиотеки генерируется автоматически.

## 7.4 Особенности алгоритма генерации кода

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

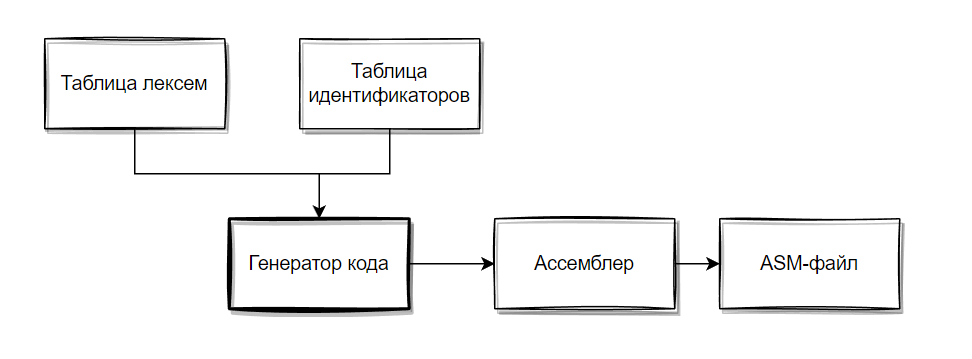


Рисунок 7.2 Структура генератора кода

## 7.5 Входные параметры генератора кода

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

## 7.6 Контрольный пример

Результат работы контрольного примера приведён на рисунке 7.2.

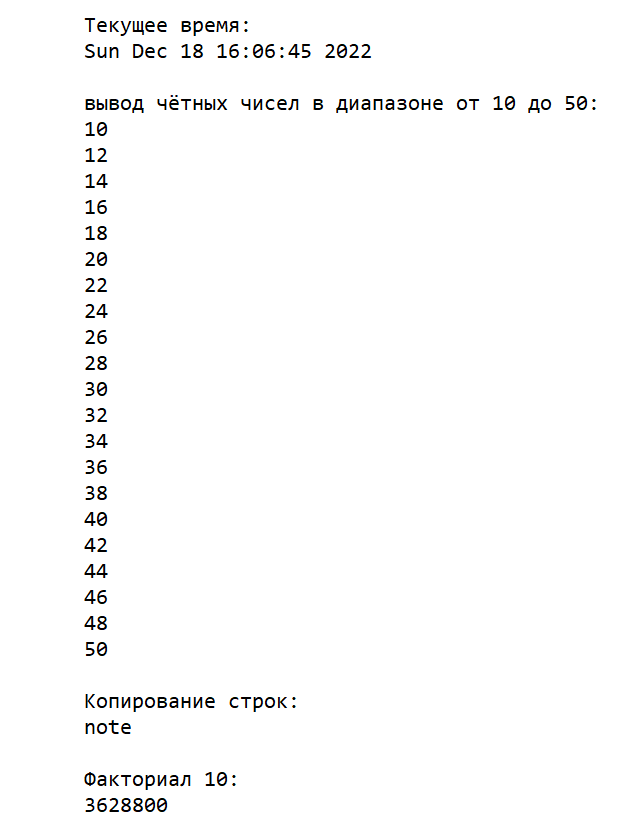


Рисунок 7.2 Результат работы программы на языке KAD-2022

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

# **8. Тестирование транслятора**

## 8.1 Тестирование проверки на допустимость символов

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

Таблица 8.1 - Тестирование проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main { ~ } | Ошибка при чтении из входного файла  Ошибка 110: Недопустимый символ в исходном файле (-in), строка 1, столбец 8 |
| main  {  write "test  } | Ошибка при чтении из входного файла  Ошибка 111: Нет закрывающей кавычки (-in), строка 3, столбец 16 |

## 8.2 Тестирование лексического анализатора

На этапе лексического анализа в языке KAD-2022 могут возникнуть ошибки, описанные в пункте 3.7. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 - Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  test123  } | Ошибка на этапе лексического анализатора  Ошибка 120: Нераспознанная лексема, строка 3, лексема 1 |
| main  {  var test;  } | Ошибка на этапе лексического анализатора  Ошибка 121: Тип данных идентификатора не определен, строка 3, лексема 2 |
| main  {  int var test;  int var test;  } | Ошибка на этапе лексического анализатора  Ошибка 131: Переопределение, строка 4, лексема 3 |
| main  {  int test;  } | Ошибка на этапе лексического анализатора  Ошибка 132: Тип идентификатора не определен, строка 3, лексема 2 |

Продолжение таблицы 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| int function FindMaxLen(str param x, str param y){} | Ошибка 133: Отсутствует точка входа main |
| main {}  main  {  int var test;  } | Ошибка 134: Обнаружено несколько точек входа main |

## 8.3 Тестирование синтаксического анализатора

На этапе синтаксического анализа в языке KAD-2022 могут возникнуть ошибки, описанные в пункте 4.6. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 - Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| function main  {  int var test;  } | 600: строка 1, Неверная структура программы |
| main  {  return 1;  } | 601: строка 4, Ошибочный оператор |
| main  {  int var test;  test = 1 + - 2;  } | 602: строка 4, Ошибка в выражении |
| int function FindMaxLen(str param x, )  {  }  main {} | 603: строка 1, Ошибка в параметрах при определении функции |
| int function FindMax(int param x)  { return 1;}  main  {  int var res;  res = FindMax(1, )  } | 604: строка 6, Ошибка в параметрах при вызове функции |

Продолжение таблицы 8.3 - Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| main  {  if (1 > 0)  then  } | 606: строка 4, Ошибка при условном переходе |
| main  {  int str var test;  } | 607: строка 3, Ошибка при определении переменной |
| main  {  if {1 > 0)  then {}  } | 608: строка 3, Ошибка при определении условия перехода |

## 8.4 Тестирование семантического анализатора

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

Таблица 8.4 - Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Текст сообщения |
| main{  int var x = 125;  x = x + “KAD2022”;  } | Ошибка на этапе Семантического анализатора  Ошибка 701: Типы данных в выражении не совпадают, строка 3 |
| main{  int var x;  x = 5 / 0;  } | Ошибка на этапе Семантического анализатора  Ошибка 700: Деление на 0, строка 3, лексема 5 |
| main{  str var y;  y = "test2" + “test3”;  } | Ошибка на этапе Семантического анализатора  Ошибка 702: Недопустимое строковое выражение справа от знака '=', строка 3 |
| str function Test(int param x)  {  return 0;  }  main {} | Ошибка на этапе Семантического анализатора  Ошибка 703: Тип функции и возвращаемое значение не совпадают, строка 3 |

Продолжение таблицы 8.4 - Тестирование семантического анализатора

|  |  |
| --- | --- |
| int function Test(int param x)  {return 0;}  main  {  int var temp;  temp = Test("test");  } | Ошибка на этапе Семантического анализатора  Ошибка 704: Несовпадение типов передаваемых параметров, строка 6, лексема 5 |
| int function Test(int param x)  {return 0;}  main  {  int var temp;  temp = Test(1, 2);  } | Ошибка на этапе Семантического анализатора  Ошибка 705: Кол-во ожидаемых функцией и передаваемых параметров не совпадают, строка 6 |
| main  {  if ("test1" > "test2")  then{}  } | Ошибка на этапе Семантического анализатора  Ошибка 706: Неверное условное выражение, строка 3 |

Семантический анализ в языке KAD-2022 содержит множество проверок по семантическим правилам, описанным в пункте 1.16.

**Заключение**

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

Сформулирована спецификация языка KAD-2022;

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

Окончательная версия языка KAD-2022 включает:

1. 2 типа данных;
2. Поддержка оператора вывода;
3. Возможность вызова 8 функций стандартной библиотеки;
4. Наличие 5 арифметических операторов для вычисления выражений;
5. Поддержка функций, операторов цикла и условия;
6. Структурированная и классифицированная система для обработки ошибок пользователя.

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

# **Список использованных источников**

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

2. Ахо, А. Теория синтаксического анализа, перевода и компиляции /А. Ахо, Дж. Ульман. – Москва : Мир, 1998. – Т. 2 : Компиляция. - 487 с.

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

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

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

# **Приложение А**

Листинг 1 – исходный код программы на языке KAD-2022

main

{

#получение текущего времени и даты

write "Текущее время:";

str var timeNow = getLocalTimeAndDate();

write timeNow;

#вывод чисел используя цикл

write "вывод чётных чисел в диапазоне от 10 до 50:";

int var i = b1010; #1010 в 2ой сс - 10

repeat(i < h32) #h32 в 8ой сс - 50

{

int var result;

result = i % 2;

if(result & 0)

then

{

write i;

}

i = i + 1;

}

#работа со строками

str var s = "notebook";

str var res = copy(res, s, 4);

write " ";

write "Копирование строк:";

write res;

#работа с числами

int var num= 10;

int var r = factorialOfNumber(num);

write " ";

write "Факториал 10:";

write r;

}

# **Приложение Б**

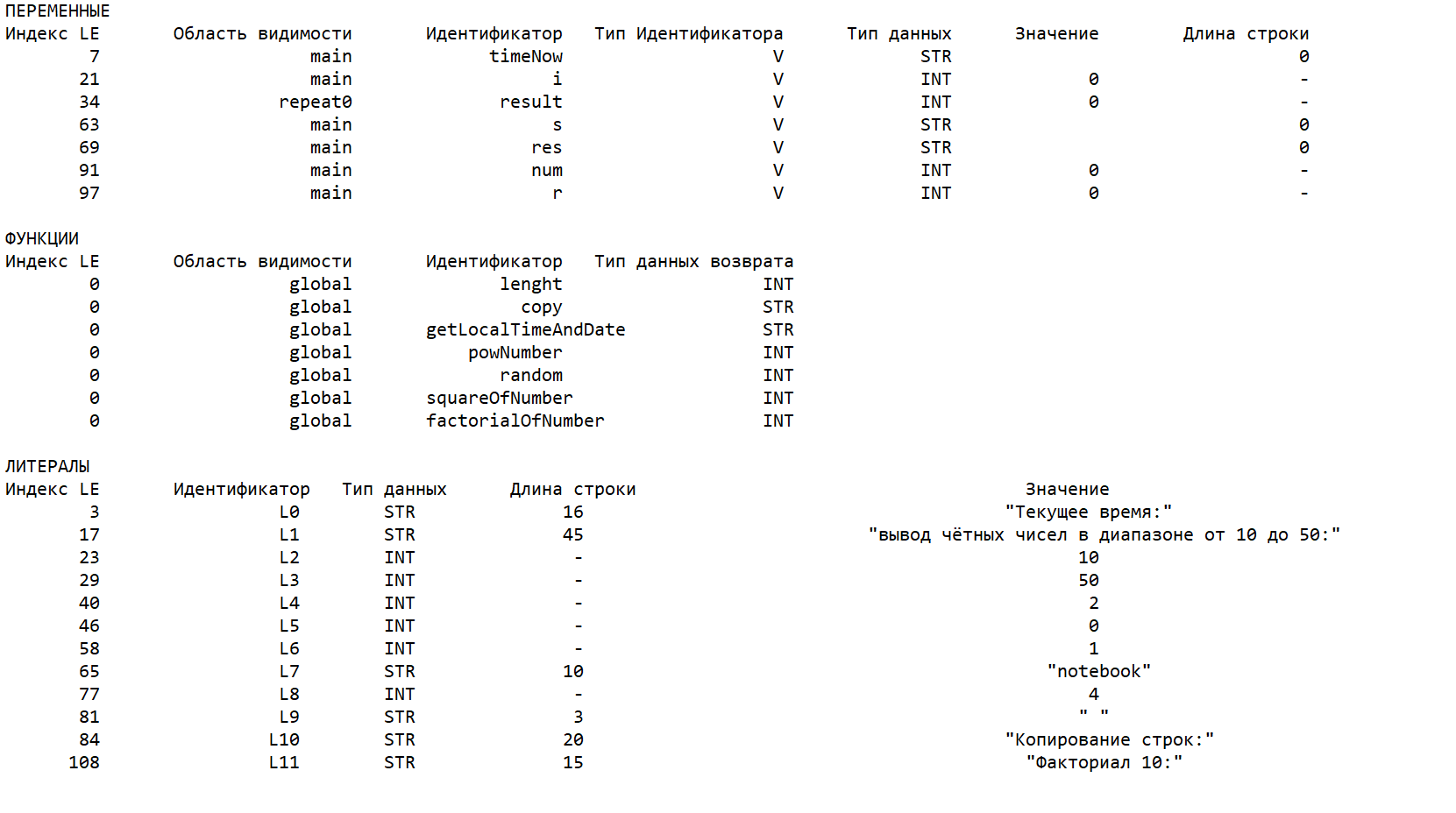


Рисунок 1. Таблица идентификаторов после контрольного примера

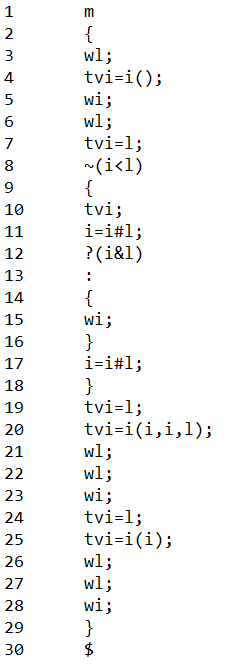


Рисунок 2. Таблица лексем после контрольного примера

# **Приложение В**

Листинг 1. Грамматика языка KAD-2022

Greibach greibach(NS('S'), TS('$'), 14,

Rule(NS('S'), GRB\_ERROR\_SERIES + 0, //Неверная структура программы

3, // m{N}S | tfi(F){NrU;}S | пустой переход

Rule::Chain(5, TS('m'), TS('{'), NS('N'), TS('}'), NS('S')),

Rule::Chain(13, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('U'), TS(';'), TS('}'), NS('S')),

Rule::Chain()

),

Rule(NS('N'), GRB\_ERROR\_SERIES + 1, //ошибочный оператор

6, // tY;N | i=E;N | wU;N | ~K{N}N | ?KJN | пустой переход

Rule::Chain(4, TS('t'), NS('Y'), TS(';'), NS('N')),

Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),

Rule::Chain(4, TS('w'), NS('E'), TS(';'), NS('N')),

Rule::Chain(4, TS('~'), NS('K'), NS('Z'), NS('N')),

Rule::Chain(4, TS('?'), NS('K'), NS('J'), NS('N')),

Rule::Chain()

),

Rule(NS('E'), GRB\_ERROR\_SERIES + 2, // Ошибка в выражении

4, // iM | lM | (E)M | i(W)M

Rule::Chain(2, TS('i'), NS('M')),

Rule::Chain(2, TS('l'), NS('M')),

Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),

Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M'))

),

Rule(NS('M'), GRB\_ERROR\_SERIES + 2, // Ошибка в выражении

2, // #E | пустой переход

Rule::Chain(2, TS('#'), NS('E')),

Rule::Chain()

),

Rule(NS('F'), GRB\_ERROR\_SERIES + 3, // Ошибка в параметрах при определении функции

2, // tP | tP,F

Листинг 1 (продолжение). Грамматика языка KAD-2022

Rule::Chain(2, TS('t'), NS('P')),

Rule::Chain(4, TS('t'), NS('P'), TS(','), NS('F'))

),

Rule(NS('P'), GRB\_ERROR\_SERIES + 3, // Ошибка в параметрах при определении функции

1, // pi

Rule::Chain(2, TS('p'), TS('i'))

),

Rule(NS('W'), GRB\_ERROR\_SERIES + 4, // Ошибка в параметрах при вызове функции

4, // i | l | i,W | l,W

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('l')),

Rule::Chain(3, TS('i'), TS(','), NS('W')),

Rule::Chain(3, TS('l'), TS(','), NS('W'))

),

Rule(NS('U'), GRB\_ERROR\_SERIES + 5, // Только литерал или идентификатор

6, // l | i

Rule::Chain(1, TS('l')),

Rule::Chain(1, TS('i')),

Rule::Chain(2, TS('l'), NS('U')),

Rule::Chain(2, TS('i'), NS('U')),

Rule::Chain(3, TS('i'), TS(','), NS('U')),

Rule::Chain(3, TS('l'), TS(','), NS('U'))

),

Rule(NS('J'), GRB\_ERROR\_SERIES + 6, // Ошибка при условном переходе

2, // :{N} | :{N}!{N}

Rule::Chain(4, TS(':'), TS('{'), NS('N'), TS('}')),

Rule::Chain(8, TS(':'), TS('{'), NS('N'), TS('}'), TS('!'), TS('{'), NS('N'), TS('}'))

),

Rule(NS('Y'), GRB\_ERROR\_SERIES + 7, // Ошибка при определении переменной

2, // vi || vi = L

Rule::Chain(2, TS('v'), TS('i')),

Rule::Chain(4, TS('v'), TS('i'), TS('='), NS('L'))

),

Rule(NS('K'), GRB\_ERROR\_SERIES + 8, // Ошибка при определении условия перехода

2, // (U<U) | (U>U)

Rule::Chain(5, TS('('), NS('U'), TS('<'), NS('U'), TS(')')),

Rule::Chain(5, TS('('), NS('U'), TS('>'), NS('U'), TS(')'))

),

Листинг 1 (продолжение). Грамматика языка KAD-2022

Rule(NS('L'), GRB\_ERROR\_SERIES + 9, // Ошибка при начальной инициализации

4, //vi = l || vi = i || vi = i(i)

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('l')),

Rule::Chain(4, TS('i'), TS('('), NS('U'), TS(')')),

Rule::Chain(3, TS('i'), TS('('), TS(')'))

),

Rule(NS('Z'), GRB\_ERROR\_SERIES + 10, //ошибка в теле цикла

1,

Rule::Chain(3, TS('{'), NS('N'), TS('}'))

)

);

Листинг 2. Структура магазинного автомата

struct MFST // магазинный автомат

{

enum RC\_STEP //код возврата функции step

{

NS\_OK, // найдено правило и цепочка, цепочка записана в стек

NS\_NORULE, // не найдено правило грамматики (ошибка в грамматике)

NS\_NORULECHAIN, // не найдена походящая цепочка правила (ошибка в исходном коде)

NS\_ERROR, // неизвесный нетерминальный символ грамматики

TS\_OK, // тек. символ ленты == вершине стека, продвинулась лента, pop стека

TS\_NOK, // тек. символ ленты != вершине стека, восстановленно состояние

LENTA\_END, // теущая позиция ленты >= lenta\_size

SURPRISE // неожиданный код возврата (ошибка в step)

};

struct MFST\_Diagnosis // диагностика

{

short posInLent; // позиция на ленте

RC\_STEP rc\_step; // код завершения шага

short ruleNum; // номер правила

Листинг 2 (продолжение). Структура магазинного автомата

short nrule\_chain; // номер цепочки правила

MFST\_Diagnosis();

MFST\_Diagnosis(short posInLent, RC\_STEP rc\_step, short ruleNum, short ruleChainNum);

} diagnosis[MFST\_DIAGN\_NUMBER]; // последние самые глубокие сообщения

GRBALPHABET\* lenta; // перекодированная (TS/NS) лента (из LEX)

short currentPosInLent; // текущая позиция на ленте

short currentRule; // номер текущего правила

short currentRuleChain; // номер текущей цепочки, текущего правила

short lenta\_size; // размер ленты

GRB::Greibach grebach; // грамматика Грейбах

LT::LexTable lexTable;

MFSTSTSTACK st; // стек автомата

std::stack<MFSTState> storestate; // стек для сохранения состояний

MFST();

MFST(const LT::LexTable& lexTable,GRB::Greibach grebach);

char\* getCSt(char\* buf); //получить содержиое стека

char\* getCLenta(char\* buf, short pos, short n = 25); //лента: n символов, начиная с pos

char\* getDiagnosis(short n, char\* buf); //получить n-ую строку диагностики или '\0'

bool savestate(std::ostream\* stream); //сохранить состояние автомата

bool resetstate(std::ostream\* stream); //восстановить состояние автомата

bool push\_chain(GRB::Rule::Chain chain);

RC\_STEP step(std::ostream\* stream); //выполнить шаг автомата

bool start(std::ostream\* stream); //запустить автомат

bool savedDiagnosis(RC\_STEP prc\_step);

void printRules(std::ostream\* stream); //вывести последовательность правил

Листинг 2 (продолжение). Структура магазинного автомата

struct Deducation // вывод

{

short stepsCount; // количество шагов в выводе

short\* nRules; // номер правила грамматики

short\* nChainsOfRules; // номер цепочек правил грамматики

Deducation()

{

this->stepsCount = 0;

this->nRules = 0;

this->nChainsOfRules = 0;

}

}deducation;

bool saveoutputTree(); // сохранить дерево вывода

};

Листинг 3. Структура грамматики Грейбаха

struct Greibach // грамматика Грейбаха

{

short size; // количество правил

GRBALPHABET startN; // стартовый символ

GRBALPHABET stbottomT; // дно стека

Rule\* rules; // множество правил

Greibach(){

size = 0;

startN = 0;

stbottomT = 0;

rules = 0;};

Greibach(GRBALPHABET pstartN, // стартовый символ

GRBALPHABET pstbottomT, // дно стека

short psize, // количество правил

Rule r, ...); // правила

short getRule( // получить правило

GRBALPHABET pnn, // левый символ правила

Rule& prule); // возвращаемое правило

Rule getRule(short n); // получить правило по номеру

};

Листинг 4. Разбор исходного кода синтаксическим анализатором

Шаг :Правило Входная лента Стек

0 :S->m{N}S m{wl;tvi=i();wi;wl;tvi=l; S$

1 : SAVESTATE: 1

1 : m{wl;tvi=i();wi;wl;tvi=l; m{N}S$

2 : {wl;tvi=i();wi;wl;tvi=l;~ {N}S$

3 : wl;tvi=i();wi;wl;tvi=l;~( N}S$

4 :N->wE;N wl;tvi=i();wi;wl;tvi=l;~( N}S$

5 : SAVESTATE: 2

5 : wl;tvi=i();wi;wl;tvi=l;~( wE;N}S$

6 : l;tvi=i();wi;wl;tvi=l;~(i E;N}S$

7 :E->lM l;tvi=i();wi;wl;tvi=l;~(i E;N}S$

8 : SAVESTATE: 3

8 : l;tvi=i();wi;wl;tvi=l;~(i lM;N}S$

9 : ;tvi=i();wi;wl;tvi=l;~(i< M;N}S$

10 :M-> ;tvi=i();wi;wl;tvi=l;~(i< M;N}S$

11 : SAVESTATE: 4

11 : ;tvi=i();wi;wl;tvi=l;~(i< ;N}S$

12 : tvi=i();wi;wl;tvi=l;~(i<l N}S$

13 :N->tY;N tvi=i();wi;wl;tvi=l;~(i<l N}S$

14 : SAVESTATE: 5

14 : tvi=i();wi;wl;tvi=l;~(i<l tY;N}S$

15 : vi=i();wi;wl;tvi=l;~(i<l) Y;N}S$

16 :Y->vi vi=i();wi;wl;tvi=l;~(i<l) Y;N}S$

17 : SAVESTATE: 6

17 : vi=i();wi;wl;tvi=l;~(i<l) vi;N}S$

18 : i=i();wi;wl;tvi=l;~(i<l){ i;N}S$

19 : =i();wi;wl;tvi=l;~(i<l){t ;N}S$

20 : TNS\_NS\_NORULECHAIN/NS\_NORULE

20 : RESTATE

20 : vi=i();wi;wl;tvi=l;~(i<l) Y;N}S$

21 :Y->vi=L vi=i();wi;wl;tvi=l;~(i<l) Y;N}S$

22 : SAVESTATE: 6

22 : vi=i();wi;wl;tvi=l;~(i<l) vi=L;N}S$

23 : i=i();wi;wl;tvi=l;~(i<l){ i=L;N}S$

24 : =i();wi;wl;tvi=l;~(i<l){t =L;N}S$

25 : i();wi;wl;tvi=l;~(i<l){tv L;N}S$

26 :L->i i();wi;wl;tvi=l;~(i<l){tv L;N}S$

27 : SAVESTATE: 7

27 : i();wi;wl;tvi=l;~(i<l){tv i;N}S$

28 : ();wi;wl;tvi=l;~(i<l){tvi ;N}S$

29 : TNS\_NS\_NORULECHAIN/NS\_NORULE

29 : RESTATE

29 : i();wi;wl;tvi=l;~(i<l){tv L;N}S$

Листинг 4 (продолжение). Разбор исходного кода синтаксическим анализатором

30 :L->i(U) i();wi;wl;tvi=l;~(i<l){tv L;N}S$

31 : SAVESTATE: 7

31 : i();wi;wl;tvi=l;~(i<l){tv i(U);N}S$

32 : ();wi;wl;tvi=l;~(i<l){tvi (U);N}S$

33 : );wi;wl;tvi=l;~(i<l){tvi; U);N}S$

34 : TNS\_NS\_NORULECHAIN/NS\_NORULE

34 : RESTATE

34 : i();wi;wl;tvi=l;~(i<l){tv L;N}S$

35 :L->i() i();wi;wl;tvi=l;~(i<l){tv L;N}S$

36 : SAVESTATE: 7

36 : i();wi;wl;tvi=l;~(i<l){tv i();N}S$

37 : ();wi;wl;tvi=l;~(i<l){tvi ();N}S$

38 : );wi;wl;tvi=l;~(i<l){tvi; );N}S$

39 : ;wi;wl;tvi=l;~(i<l){tvi;i ;N}S$

40 : wi;wl;tvi=l;~(i<l){tvi;i= N}S$

41 :N->wE;N wi;wl;tvi=l;~(i<l){tvi;i= N}S$

42 : SAVESTATE: 8

42 : wi;wl;tvi=l;~(i<l){tvi;i= wE;N}S$

43 : i;wl;tvi=l;~(i<l){tvi;i=i E;N}S$

44 :E->iM i;wl;tvi=l;~(i<l){tvi;i=i E;N}S$

45 : SAVESTATE: 9

45 : i;wl;tvi=l;~(i<l){tvi;i=i iM;N}S$

46 : ;wl;tvi=l;~(i<l){tvi;i=i# M;N}S$

47 :M-> ;wl;tvi=l;~(i<l){tvi;i=i# M;N}S$

48 : SAVESTATE: 10

48 : ;wl;tvi=l;~(i<l){tvi;i=i# ;N}S$

49 : wl;tvi=l;~(i<l){tvi;i=i#l N}S$

50 :N->wE;N wl;tvi=l;~(i<l){tvi;i=i#l N}S$

51 : SAVESTATE: 11

51 : wl;tvi=l;~(i<l){tvi;i=i#l wE;N}S$

52 : l;tvi=l;~(i<l){tvi;i=i#l; E;N}S$

53 :E->lM l;tvi=l;~(i<l){tvi;i=i#l; E;N}S$

54 : SAVESTATE: 12

54 : l;tvi=l;~(i<l){tvi;i=i#l; lM;N}S$

55 : ;tvi=l;~(i<l){tvi;i=i#l;? M;N}S$

56 :M-> ;tvi=l;~(i<l){tvi;i=i#l;? M;N}S$

57 : SAVESTATE: 13

57 : ;tvi=l;~(i<l){tvi;i=i#l;? ;N}S$

58 : tvi=l;~(i<l){tvi;i=i#l;?( N}S$

59 :N->tY;N tvi=l;~(i<l){tvi;i=i#l;?( N}S$

60 : SAVESTATE: 14

60 : tvi=l;~(i<l){tvi;i=i#l;?( tY;N}S$

Листинг 4 (продолжение). Разбор исходного кода синтаксическим анализатором

61 : vi=l;~(i<l){tvi;i=i#l;?(i Y;N}S$

62 :Y->vi vi=l;~(i<l){tvi;i=i#l;?(i Y;N}S$

63 : SAVESTATE: 15

63 : vi=l;~(i<l){tvi;i=i#l;?(i vi;N}S$

64 : i=l;~(i<l){tvi;i=i#l;?(i& i;N}S$

65 : =l;~(i<l){tvi;i=i#l;?(i&l ;N}S$

66 : TNS\_NS\_NORULECHAIN/NS\_NORULE

66 : RESTATE

66 : vi=l;~(i<l){tvi;i=i#l;?(i Y;N}S$

67 :Y->vi=L vi=l;~(i<l){tvi;i=i#l;?(i Y;N}S$

68 : SAVESTATE: 15

68 : vi=l;~(i<l){tvi;i=i#l;?(i vi=L;N}S$

69 : i=l;~(i<l){tvi;i=i#l;?(i& i=L;N}S$

70 : =l;~(i<l){tvi;i=i#l;?(i&l =L;N}S$

71 : l;~(i<l){tvi;i=i#l;?(i&l) L;N}S$

72 :L->l l;~(i<l){tvi;i=i#l;?(i&l) L;N}S$

73 : SAVESTATE: 16

73 : l;~(i<l){tvi;i=i#l;?(i&l) l;N}S$

74 : ;~(i<l){tvi;i=i#l;?(i&l): ;N}S$

75 : ~(i<l){tvi;i=i#l;?(i&l):{ N}S$

76 :N->~KZN ~(i<l){tvi;i=i#l;?(i&l):{ N}S$

77 : SAVESTATE: 17

77 : ~(i<l){tvi;i=i#l;?(i&l):{ ~KZN}S$

78 : (i<l){tvi;i=i#l;?(i&l):{w KZN}S$

79 :K->(U<U) (i<l){tvi;i=i#l;?(i&l):{w KZN}S$

80 : SAVESTATE: 18

80 : (i<l){tvi;i=i#l;?(i&l):{w (U<U)ZN}S$

81 : i<l){tvi;i=i#l;?(i&l):{wi U<U)ZN}S$

82 :U->i i<l){tvi;i=i#l;?(i&l):{wi U<U)ZN}S$

83 : SAVESTATE: 19

83 : i<l){tvi;i=i#l;?(i&l):{wi i<U)ZN}S$

84 : <l){tvi;i=i#l;?(i&l):{wi; <U)ZN}S$

85 : l){tvi;i=i#l;?(i&l):{wi;} U)ZN}S$

86 :U->l l){tvi;i=i#l;?(i&l):{wi;} U)ZN}S$

87 : SAVESTATE: 20

87 : l){tvi;i=i#l;?(i&l):{wi;} l)ZN}S$

88 : ){tvi;i=i#l;?(i&l):{wi;}i )ZN}S$

89 : {tvi;i=i#l;?(i&l):{wi;}i= ZN}S$

90 :Z->{N} {tvi;i=i#l;?(i&l):{wi;}i= ZN}S$

91 : SAVESTATE: 21

91 : {tvi;i=i#l;?(i&l):{wi;}i= {N}N}S$

Листинг 4 (продолжение). Разбор исходного кода синтаксическим анализатором

92 : tvi;i=i#l;?(i&l):{wi;}i=i N}N}S$

93 :N->tY;N tvi;i=i#l;?(i&l):{wi;}i=i N}N}S$

94 : SAVESTATE: 22

94 : tvi;i=i#l;?(i&l):{wi;}i=i tY;N}N}S$

95 : vi;i=i#l;?(i&l):{wi;}i=i# Y;N}N}S$

96 :Y->vi vi;i=i#l;?(i&l):{wi;}i=i# Y;N}N}S$

97 : SAVESTATE: 23

97 : vi;i=i#l;?(i&l):{wi;}i=i# vi;N}N}S$

98 : i;i=i#l;?(i&l):{wi;}i=i#l i;N}N}S$

99 : ;i=i#l;?(i&l):{wi;}i=i#l; ;N}N}S$

100 : i=i#l;?(i&l):{wi;}i=i#l;} N}N}S$

101 :N->i=E;N i=i#l;?(i&l):{wi;}i=i#l;} N}N}S$

102 : SAVESTATE: 24

102 : i=i#l;?(i&l):{wi;}i=i#l;} i=E;N}N}S$

103 : =i#l;?(i&l):{wi;}i=i#l;}t =E;N}N}S$

104 : i#l;?(i&l):{wi;}i=i#l;}tv E;N}N}S$

105 :E->iM i#l;?(i&l):{wi;}i=i#l;}tv E;N}N}S$

106 : SAVESTATE: 25

106 : i#l;?(i&l):{wi;}i=i#l;}tv iM;N}N}S$

107 : #l;?(i&l):{wi;}i=i#l;}tvi M;N}N}S$

108 :M->#E #l;?(i&l):{wi;}i=i#l;}tvi M;N}N}S$

109 : SAVESTATE: 26

109 : #l;?(i&l):{wi;}i=i#l;}tvi #E;N}N}S$

110 : l;?(i&l):{wi;}i=i#l;}tvi= E;N}N}S$

111 :E->lM l;?(i&l):{wi;}i=i#l;}tvi= E;N}N}S$

112 : SAVESTATE: 27

112 : l;?(i&l):{wi;}i=i#l;}tvi= lM;N}N}S$

113 : ;?(i&l):{wi;}i=i#l;}tvi=l M;N}N}S$

114 :M-> ;?(i&l):{wi;}i=i#l;}tvi=l M;N}N}S$

115 : SAVESTATE: 28

115 : ;?(i&l):{wi;}i=i#l;}tvi=l ;N}N}S$

116 : ?(i&l):{wi;}i=i#l;}tvi=l; N}N}S$

117 :N->?KJN ?(i&l):{wi;}i=i#l;}tvi=l; N}N}S$

118 : SAVESTATE: 29

118 : ?(i&l):{wi;}i=i#l;}tvi=l; ?KJN}N}S$

119 : (i&l):{wi;}i=i#l;}tvi=l;t KJN}N}S$

120 :K->(U<U) (i&l):{wi;}i=i#l;}tvi=l;t KJN}N}S$

121 : SAVESTATE: 30

121 : (i&l):{wi;}i=i#l;}tvi=l;t (U<U)JN}N}S$

Листинг 4 (продолжение). Разбор исходного кода синтаксическим анализатором

122 : i&l):{wi;}i=i#l;}tvi=l;tv U<U)JN}N}S$

123 :U->i i&l):{wi;}i=i#l;}tvi=l;tv U<U)JN}N}S$

124 : SAVESTATE: 31

124 : i&l):{wi;}i=i#l;}tvi=l;tv i<U)JN}N}S$

125 : &l):{wi;}i=i#l;}tvi=l;tvi <U)JN}N}S$

126 : TNS\_NS\_NORULECHAIN/NS\_NORULE

126 : RESTATE

126 : i&l):{wi;}i=i#l;}tvi=l;tv U<U)JN}N}S$

127 :U->iU i&l):{wi;}i=i#l;}tvi=l;tv U<U)JN}N}S$

128 : SAVESTATE: 31

128 : i&l):{wi;}i=i#l;}tvi=l;tv iU<U)JN}N}S$

129 : &l):{wi;}i=i#l;}tvi=l;tvi U<U)JN}N}S$

130 : TNS\_NS\_NORULECHAIN/NS\_NORULE

130 : RESTATE

130 : i&l):{wi;}i=i#l;}tvi=l;tv U<U)JN}N}S$

131 :U->i,U i&l):{wi;}i=i#l;}tvi=l;tv U<U)JN}N}S$

132 : SAVESTATE: 31

132 : i&l):{wi;}i=i#l;}tvi=l;tv i,U<U)JN}N}S$

133 : &l):{wi;}i=i#l;}tvi=l;tvi ,U<U)JN}N}S$

134 : TNS\_NS\_NORULECHAIN/NS\_NORULE

134 : RESTATE

134 : i&l):{wi;}i=i#l;}tvi=l;tv U<U)JN}N}S$

135 : TNS\_NS\_NORULECHAIN/NS\_NORULE

135 : RESTATE

135 : (i&l):{wi;}i=i#l;}tvi=l;t KJN}N}S$

136 :K->(U>U) (i&l):{wi;}i=i#l;}tvi=l;t KJN}N}S$

137 : SAVESTATE: 30

137 : (i&l):{wi;}i=i#l;}tvi=l;t (U>U)JN}N}S$

Листинг 4 (продолжение). Разбор исходного кода синтаксическим анализатором

138 : i&l):{wi;}i=i#l;}tvi=l;tv U>U)JN}N}S$

139 :U->i i&l):{wi;}i=i#l;}tvi=l;tv U>U)JN}N}S$

140 : SAVESTATE: 31

140 : i&l):{wi;}i=i#l;}tvi=l;tv i>U)JN}N}S$

141 : &l):{wi;}i=i#l;}tvi=l;tvi >U)JN}N}S$

142 : TNS\_NS\_NORULECHAIN/NS\_NORULE

142 : RESTATE

142 : i&l):{wi;}i=i#l;}tvi=l;tv U>U)JN}N}S$

143 :U->iU i&l):{wi;}i=i#l;}tvi=l;tv U>U)JN}N}S$

144 : SAVESTATE: 31

144 : i&l):{wi;}i=i#l;}tvi=l;tv iU>U)JN}N}S$

145 : &l):{wi;}i=i#l;}tvi=l;tvi U>U)JN}N}S$

146 : TNS\_NS\_NORULECHAIN/NS\_NORULE

146 : RESTATE

146 : i&l):{wi;}i=i#l;}tvi=l;tv U>U)JN}N}S$

147 :U->i,U i&l):{wi;}i=i#l;}tvi=l;tv U>U)JN}N}S$

148 : SAVESTATE: 31

148 : i&l):{wi;}i=i#l;}tvi=l;tv i,U>U)JN}N}S$

149 : &l):{wi;}i=i#l;}tvi=l;tvi ,U>U)JN}N}S$

150 : TNS\_NS\_NORULECHAIN/NS\_NORULE

150 : RESTATE

150 : i&l):{wi;}i=i#l;}tvi=l;tv U>U)JN}N}S$

151 : TNS\_NS\_NORULECHAIN/NS\_NORULE

151 : RESTATE

151 : (i&l):{wi;}i=i#l;}tvi=l;t KJN}N}S$

152 :K->(U&U) (i&l):{wi;}i=i#l;}tvi=l;t KJN}N}S$

153 : SAVESTATE: 30

153 : (i&l):{wi;}i=i#l;}tvi=l;t (U&U)JN}N}S$

Листинг 4 (продолжение). Разбор исходного кода синтаксическим анализатором

154 : i&l):{wi;}i=i#l;}tvi=l;tv U&U)JN}N}S$

155 :U->i i&l):{wi;}i=i#l;}tvi=l;tv U&U)JN}N}S$

156 : SAVESTATE: 31

156 : i&l):{wi;}i=i#l;}tvi=l;tv i&U)JN}N}S$

157 : &l):{wi;}i=i#l;}tvi=l;tvi &U)JN}N}S$

158 : l):{wi;}i=i#l;}tvi=l;tvi= U)JN}N}S$

159 :U->l l):{wi;}i=i#l;}tvi=l;tvi= U)JN}N}S$

160 : SAVESTATE: 32

160 : l):{wi;}i=i#l;}tvi=l;tvi= l)JN}N}S$

161 : ):{wi;}i=i#l;}tvi=l;tvi=i )JN}N}S$

162 : :{wi;}i=i#l;}tvi=l;tvi=i( JN}N}S$

163 :J->:{N} :{wi;}i=i#l;}tvi=l;tvi=i( JN}N}S$

164 : SAVESTATE: 33

164 : :{wi;}i=i#l;}tvi=l;tvi=i( :{N}N}N}S$

165 : {wi;}i=i#l;}tvi=l;tvi=i(i {N}N}N}S$

166 : wi;}i=i#l;}tvi=l;tvi=i(i, N}N}N}S$

167 :N->wE;N wi;}i=i#l;}tvi=l;tvi=i(i, N}N}N}S$

168 : SAVESTATE: 34

168 : wi;}i=i#l;}tvi=l;tvi=i(i, wE;N}N}N}S$

169 : i;}i=i#l;}tvi=l;tvi=i(i,i E;N}N}N}S$

170 :E->iM i;}i=i#l;}tvi=l;tvi=i(i,i E;N}N}N}S$

171 : SAVESTATE: 35

171 : i;}i=i#l;}tvi=l;tvi=i(i,i iM;N}N}N}S$

172 : ;}i=i#l;}tvi=l;tvi=i(i,i, M;N}N}N}S$

173 :M-> ;}i=i#l;}tvi=l;tvi=i(i,i, M;N}N}N}S$

174 : SAVESTATE: 36

174 : ;}i=i#l;}tvi=l;tvi=i(i,i, ;N}N}N}S$

175 : }i=i#l;}tvi=l;tvi=i(i,i,l N}N}N}S$

176 :N-> }i=i#l;}tvi=l;tvi=i(i,i,l N}N}N}S$

177 : SAVESTATE: 37

177 : }i=i#l;}tvi=l;tvi=i(i,i,l }N}N}S$

Листинг 4 (продолжение). Разбор исходного кода синтаксическим анализатором

178 : i=i#l;}tvi=l;tvi=i(i,i,l) N}N}S$

179 :N->i=E;N i=i#l;}tvi=l;tvi=i(i,i,l) N}N}S$

180 : SAVESTATE: 38

180 : i=i#l;}tvi=l;tvi=i(i,i,l) i=E;N}N}S$

181 : =i#l;}tvi=l;tvi=i(i,i,l); =E;N}N}S$

182 : i#l;}tvi=l;tvi=i(i,i,l);w E;N}N}S$

183 :E->iM i#l;}tvi=l;tvi=i(i,i,l);w E;N}N}S$

184 : SAVESTATE: 39

184 : i#l;}tvi=l;tvi=i(i,i,l);w iM;N}N}S$

185 : #l;}tvi=l;tvi=i(i,i,l);wl M;N}N}S$

186 :M->#E #l;}tvi=l;tvi=i(i,i,l);wl M;N}N}S$

187 : SAVESTATE: 40

187 : #l;}tvi=l;tvi=i(i,i,l);wl #E;N}N}S$

188 : l;}tvi=l;tvi=i(i,i,l);wl; E;N}N}S$

189 :E->lM l;}tvi=l;tvi=i(i,i,l);wl; E;N}N}S$

190 : SAVESTATE: 41

190 : l;}tvi=l;tvi=i(i,i,l);wl; lM;N}N}S$

191 : ;}tvi=l;tvi=i(i,i,l);wl;w M;N}N}S$

192 :M-> ;}tvi=l;tvi=i(i,i,l);wl;w M;N}N}S$

193 : SAVESTATE: 42

193 : ;}tvi=l;tvi=i(i,i,l);wl;w ;N}N}S$

194 : }tvi=l;tvi=i(i,i,l);wl;wl N}N}S$

195 :N-> }tvi=l;tvi=i(i,i,l);wl;wl N}N}S$

196 : SAVESTATE: 43

196 : }tvi=l;tvi=i(i,i,l);wl;wl }N}S$

197 : tvi=l;tvi=i(i,i,l);wl;wl; N}S$

198 :N->tY;N tvi=l;tvi=i(i,i,l);wl;wl; N}S$

199 : SAVESTATE: 44

199 : tvi=l;tvi=i(i,i,l);wl;wl; tY;N}S$

200 : vi=l;tvi=i(i,i,l);wl;wl;w Y;N}S$

201 :Y->vi vi=l;tvi=i(i,i,l);wl;wl;w Y;N}S$

202 : SAVESTATE: 45

202 : vi=l;tvi=i(i,i,l);wl;wl;w vi;N}S$

203 : i=l;tvi=i(i,i,l);wl;wl;wi i;N}S$

204 : =l;tvi=i(i,i,l);wl;wl;wi; ;N}S$

205 : TNS\_NS\_NORULECHAIN/NS\_NORULE

205 : RESTATE

205 : vi=l;tvi=i(i,i,l);wl;wl;w Y;N}S$

206 :Y->vi=L vi=l;tvi=i(i,i,l);wl;wl;w Y;N}S$

207 : SAVESTATE: 45

207 : vi=l;tvi=i(i,i,l);wl;wl;w vi=L;N}S$

Листинг 4 (продолжение). Разбор исходного кода синтаксическим анализатором

208 : i=l;tvi=i(i,i,l);wl;wl;wi i=L;N}S$

209 : =l;tvi=i(i,i,l);wl;wl;wi; =L;N}S$

210 : l;tvi=i(i,i,l);wl;wl;wi;t L;N}S$

211 :L->l l;tvi=i(i,i,l);wl;wl;wi;t L;N}S$

212 : SAVESTATE: 46

212 : l;tvi=i(i,i,l);wl;wl;wi;t l;N}S$

213 : ;tvi=i(i,i,l);wl;wl;wi;tv ;N}S$

214 : tvi=i(i,i,l);wl;wl;wi;tvi N}S$

215 :N->tY;N tvi=i(i,i,l);wl;wl;wi;tvi N}S$

216 : SAVESTATE: 47

216 : tvi=i(i,i,l);wl;wl;wi;tvi tY;N}S$

217 : vi=i(i,i,l);wl;wl;wi;tvi= Y;N}S$

218 :Y->vi vi=i(i,i,l);wl;wl;wi;tvi= Y;N}S$

219 : SAVESTATE: 48

219 : vi=i(i,i,l);wl;wl;wi;tvi= vi;N}S$

220 : i=i(i,i,l);wl;wl;wi;tvi=l i;N}S$

221 : =i(i,i,l);wl;wl;wi;tvi=l; ;N}S$

222 : TNS\_NS\_NORULECHAIN/NS\_NORULE

222 : RESTATE

222 : vi=i(i,i,l);wl;wl;wi;tvi= Y;N}S$

223 :Y->vi=L vi=i(i,i,l);wl;wl;wi;tvi= Y;N}S$

224 : SAVESTATE: 48

224 : vi=i(i,i,l);wl;wl;wi;tvi= vi=L;N}S$

225 : i=i(i,i,l);wl;wl;wi;tvi=l i=L;N}S$

226 : =i(i,i,l);wl;wl;wi;tvi=l; =L;N}S$

227 : i(i,i,l);wl;wl;wi;tvi=l;t L;N}S$

228 :L->i i(i,i,l);wl;wl;wi;tvi=l;t L;N}S$

229 : SAVESTATE: 49

229 : i(i,i,l);wl;wl;wi;tvi=l;t i;N}S$

230 : (i,i,l);wl;wl;wi;tvi=l;tv ;N}S$

231 : TNS\_NS\_NORULECHAIN/NS\_NORULE

231 : RESTATE

231 : i(i,i,l);wl;wl;wi;tvi=l;t L;N}S$

232 :L->i(U) i(i,i,l);wl;wl;wi;tvi=l;t L;N}S$

233 : SAVESTATE: 49

233 : i(i,i,l);wl;wl;wi;tvi=l;t i(U);N}S$

234 : (i,i,l);wl;wl;wi;tvi=l;tv (U);N}S$

235 : i,i,l);wl;wl;wi;tvi=l;tvi U);N}S$

236 :U->i i,i,l);wl;wl;wi;tvi=l;tvi U);N}S$

237 : SAVESTATE: 50

237 : i,i,l);wl;wl;wi;tvi=l;tvi i);N}S$

238 : ,i,l);wl;wl;wi;tvi=l;tvi= );N}S$

Листинг 4 (продолжение). Разбор исходного кода синтаксическим анализатором

239 : TNS\_NS\_NORULECHAIN/NS\_NORULE

239 : RESTATE

239 : i,i,l);wl;wl;wi;tvi=l;tvi U);N}S$

240 :U->iU i,i,l);wl;wl;wi;tvi=l;tvi U);N}S$

241 : SAVESTATE: 50

241 : i,i,l);wl;wl;wi;tvi=l;tvi iU);N}S$

242 : ,i,l);wl;wl;wi;tvi=l;tvi= U);N}S$

243 : TNS\_NS\_NORULECHAIN/NS\_NORULE

243 : RESTATE

243 : i,i,l);wl;wl;wi;tvi=l;tvi U);N}S$

244 :U->i,U i,i,l);wl;wl;wi;tvi=l;tvi U);N}S$

245 : SAVESTATE: 50

245 : i,i,l);wl;wl;wi;tvi=l;tvi i,U);N}S$

246 : ,i,l);wl;wl;wi;tvi=l;tvi= ,U);N}S$

247 : i,l);wl;wl;wi;tvi=l;tvi=i U);N}S$

248 :U->i i,l);wl;wl;wi;tvi=l;tvi=i U);N}S$

249 : SAVESTATE: 51

249 : i,l);wl;wl;wi;tvi=l;tvi=i i);N}S$

250 : ,l);wl;wl;wi;tvi=l;tvi=i( );N}S$

251 : TNS\_NS\_NORULECHAIN/NS\_NORULE

251 : RESTATE

251 : i,l);wl;wl;wi;tvi=l;tvi=i U);N}S$

252 :U->iU i,l);wl;wl;wi;tvi=l;tvi=i U);N}S$

253 : SAVESTATE: 51

253 : i,l);wl;wl;wi;tvi=l;tvi=i iU);N}S$

254 : ,l);wl;wl;wi;tvi=l;tvi=i( U);N}S$

255 : TNS\_NS\_NORULECHAIN/NS\_NORULE

255 : RESTATE

255 : i,l);wl;wl;wi;tvi=l;tvi=i U);N}S$

256 :U->i,U i,l);wl;wl;wi;tvi=l;tvi=i U);N}S$

257 : SAVESTATE: 51

257 : i,l);wl;wl;wi;tvi=l;tvi=i i,U);N}S$

258 : ,l);wl;wl;wi;tvi=l;tvi=i( ,U);N}S$

259 : l);wl;wl;wi;tvi=l;tvi=i(i U);N}S$

260 :U->l l);wl;wl;wi;tvi=l;tvi=i(i U);N}S$

261 : SAVESTATE: 52

261 : l);wl;wl;wi;tvi=l;tvi=i(i l);N}S$

262 : );wl;wl;wi;tvi=l;tvi=i(i) );N}S$

263 : ;wl;wl;wi;tvi=l;tvi=i(i); ;N}S$

264 : wl;wl;wi;tvi=l;tvi=i(i);w N}S$

265 :N->wE;N wl;wl;wi;tvi=l;tvi=i(i);w N}S$

266 : SAVESTATE: 53

Листинг 4 (продолжение). Разбор исходного кода синтаксическим анализатором

266 : wl;wl;wi;tvi=l;tvi=i(i);w wE;N}S$

267 : l;wl;wi;tvi=l;tvi=i(i);wl E;N}S$

268 :E->lM l;wl;wi;tvi=l;tvi=i(i);wl E;N}S$

269 : SAVESTATE: 54

269 : l;wl;wi;tvi=l;tvi=i(i);wl lM;N}S$

270 : ;wl;wi;tvi=l;tvi=i(i);wl; M;N}S$

271 :M-> ;wl;wi;tvi=l;tvi=i(i);wl; M;N}S$

272 : SAVESTATE: 55

272 : ;wl;wi;tvi=l;tvi=i(i);wl; ;N}S$

273 : wl;wi;tvi=l;tvi=i(i);wl;w N}S$

274 :N->wE;N wl;wi;tvi=l;tvi=i(i);wl;w N}S$

275 : SAVESTATE: 56

275 : wl;wi;tvi=l;tvi=i(i);wl;w wE;N}S$

276 : l;wi;tvi=l;tvi=i(i);wl;wl E;N}S$

277 :E->lM l;wi;tvi=l;tvi=i(i);wl;wl E;N}S$

278 : SAVESTATE: 57

278 : l;wi;tvi=l;tvi=i(i);wl;wl lM;N}S$

279 : ;wi;tvi=l;tvi=i(i);wl;wl; M;N}S$

280 :M-> ;wi;tvi=l;tvi=i(i);wl;wl; M;N}S$

281 : SAVESTATE: 58

281 : ;wi;tvi=l;tvi=i(i);wl;wl; ;N}S$

282 : wi;tvi=l;tvi=i(i);wl;wl;w N}S$

283 :N->wE;N wi;tvi=l;tvi=i(i);wl;wl;w N}S$

284 : SAVESTATE: 59

284 : wi;tvi=l;tvi=i(i);wl;wl;w wE;N}S$

285 : i;tvi=l;tvi=i(i);wl;wl;wi E;N}S$

286 :E->iM i;tvi=l;tvi=i(i);wl;wl;wi E;N}S$

287 : SAVESTATE: 60

287 : i;tvi=l;tvi=i(i);wl;wl;wi iM;N}S$

288 : ;tvi=l;tvi=i(i);wl;wl;wi; M;N}S$

289 :M-> ;tvi=l;tvi=i(i);wl;wl;wi; M;N}S$

290 : SAVESTATE: 61

290 : ;tvi=l;tvi=i(i);wl;wl;wi; ;N}S$

291 : tvi=l;tvi=i(i);wl;wl;wi;} N}S$

292 :N->tY;N tvi=l;tvi=i(i);wl;wl;wi;} N}S$

293 : SAVESTATE: 62

293 : tvi=l;tvi=i(i);wl;wl;wi;} tY;N}S$

294 : vi=l;tvi=i(i);wl;wl;wi;}$ Y;N}S$

295 :Y->vi vi=l;tvi=i(i);wl;wl;wi;}$ Y;N}S$

296 : SAVESTATE: 63

296 : vi=l;tvi=i(i);wl;wl;wi;}$ vi;N}S$

297 : i=l;tvi=i(i);wl;wl;wi;}$ i;N}S$

Листинг 4 (продолжение). Разбор исходного кода синтаксическим анализатором

298 : =l;tvi=i(i);wl;wl;wi;}$ ;N}S$

299 : TNS\_NS\_NORULECHAIN/NS\_NORULE

299 : RESTATE

299 : vi=l;tvi=i(i);wl;wl;wi;}$ Y;N}S$

300 :Y->vi=L vi=l;tvi=i(i);wl;wl;wi;}$ Y;N}S$

301 : SAVESTATE: 63

301 : vi=l;tvi=i(i);wl;wl;wi;}$ vi=L;N}S$

302 : i=l;tvi=i(i);wl;wl;wi;}$ i=L;N}S$

303 : =l;tvi=i(i);wl;wl;wi;}$ =L;N}S$

304 : l;tvi=i(i);wl;wl;wi;}$ L;N}S$

305 :L->l l;tvi=i(i);wl;wl;wi;}$ L;N}S$

306 : SAVESTATE: 64

306 : l;tvi=i(i);wl;wl;wi;}$ l;N}S$

307 : ;tvi=i(i);wl;wl;wi;}$ ;N}S$

308 : tvi=i(i);wl;wl;wi;}$ N}S$

309 :N->tY;N tvi=i(i);wl;wl;wi;}$ N}S$

310 : SAVESTATE: 65

310 : tvi=i(i);wl;wl;wi;}$ tY;N}S$

311 : vi=i(i);wl;wl;wi;}$ Y;N}S$

312 :Y->vi vi=i(i);wl;wl;wi;}$ Y;N}S$

313 : SAVESTATE: 66

313 : vi=i(i);wl;wl;wi;}$ vi;N}S$

314 : i=i(i);wl;wl;wi;}$ i;N}S$

315 : =i(i);wl;wl;wi;}$ ;N}S$

316 : TNS\_NS\_NORULECHAIN/NS\_NORULE

316 : RESTATE

316 : vi=i(i);wl;wl;wi;}$ Y;N}S$

317 :Y->vi=L vi=i(i);wl;wl;wi;}$ Y;N}S$

318 : SAVESTATE: 66

318 : vi=i(i);wl;wl;wi;}$ vi=L;N}S$

319 : i=i(i);wl;wl;wi;}$ i=L;N}S$

320 : =i(i);wl;wl;wi;}$ =L;N}S$

321 : i(i);wl;wl;wi;}$ L;N}S$

322 :L->i i(i);wl;wl;wi;}$ L;N}S$

323 : SAVESTATE: 67

323 : i(i);wl;wl;wi;}$ i;N}S$

324 : (i);wl;wl;wi;}$ ;N}S$

325 : TNS\_NS\_NORULECHAIN/NS\_NORULE

325 : RESTATE

325 : i(i);wl;wl;wi;}$ L;N}S$

326 :L->i(U) i(i);wl;wl;wi;}$ L;N}S$

Листинг 4 (продолжение). Разбор исходного кода синтаксическим анализатором

327 : SAVESTATE: 67

327 : i(i);wl;wl;wi;}$ i(U);N}S$

328 : (i);wl;wl;wi;}$ (U);N}S$

329 : i);wl;wl;wi;}$ U);N}S$

330 :U->i i);wl;wl;wi;}$ U);N}S$

331 : SAVESTATE: 68

331 : i);wl;wl;wi;}$ i);N}S$

332 : );wl;wl;wi;}$ );N}S$

333 : ;wl;wl;wi;}$ ;N}S$

334 : wl;wl;wi;}$ N}S$

335 :N->wE;N wl;wl;wi;}$ N}S$

336 : SAVESTATE: 69

336 : wl;wl;wi;}$ wE;N}S$

337 : l;wl;wi;}$ E;N}S$

338 :E->lM l;wl;wi;}$ E;N}S$

339 : SAVESTATE: 70

339 : l;wl;wi;}$ lM;N}S$

340 : ;wl;wi;}$ M;N}S$

341 :M-> ;wl;wi;}$ M;N}S$

342 : SAVESTATE: 71

342 : ;wl;wi;}$ ;N}S$

343 : wl;wi;}$ N}S$

344 :N->wE;N wl;wi;}$ N}S$

345 : SAVESTATE: 72

345 : wl;wi;}$ wE;N}S$

346 : l;wi;}$ E;N}S$

347 :E->lM l;wi;}$ E;N}S$

348 : SAVESTATE: 73

348 : l;wi;}$ lM;N}S$

349 : ;wi;}$ M;N}S$

350 :M-> ;wi;}$ M;N}S$

351 : SAVESTATE: 74

351 : ;wi;}$ ;N}S$

352 : wi;}$ N}S$

353 :N->wE;N wi;}$ N}S$

354 : SAVESTATE: 75

354 : wi;}$ wE;N}S$

355 : i;}$ E;N}S$

356 :E->iM i;}$ E;N}S$

357 : SAVESTATE: 76

357 : i;}$ iM;N}S$

358 : ;}$ M;N}S$

Листинг 4 (продолжение). Разбор исходного кода синтаксическим анализатором

359 :M-> ;}$ M;N}S$

360 : SAVESTATE: 77

360 : ;}$ ;N}S$

361 : }$ N}S$

362 :N-> }$ N}S$

363 : SAVESTATE: 78

363 : }$ }S$

364 : $ S$

365 :S-> $ S$

366 : SAVESTATE: 79

366 : $ $

367 :

368 : LENTA\_END

369 : ------>LENTA\_END

# **Приложение Г**

Листинг 1. Программная реализация механизма преобразования в ПОЛИЗ

bool polishNotation(int lextable\_pos, LT::LexTable& lextable, IT::IdTable& idtable)

{

stack<LT::Entry\*> stk; //создаем стек для хранения временных операций

queue<LT::Entry\*> result;

bool function = false;

int quantityParm = 0;

int i = ++lextable\_pos;

for (; lextable.table[i]->lexema != LEX\_SEMICOLON && (lextable.table[i]->lexema != LEX\_RIGHTHESIS || !stk.empty()); i++)

{

switch (lextable.table[i]->lexema)

{

case LEX\_ID: //операнды

case LEX\_LITERAL:

if (idtable.table[lextable.table[i]->idxTI]->idtype == IT::IDTYPE::F)

{

quantityParm = 0;

function = true;

result.push(lextable.table[i]);

break;

}

if (function && !quantityParm)

quantityParm++;

result.push(lextable.table[i]);

break;

case LEX\_OPERATOR:

if (stk.empty() || stk.top()->lexema == LEX\_LEFTHESIS)

stk.push(lextable.table[i]);

else

{

int prioritet = priority(lextable.table[i]->sign);

if (priority(stk.top()->sign) >= prioritet)

{

result.push(stk.top());

stk.pop();

}

stk.push(lextable.table[i]);

Листинг 1 (продолжение). Программная реализация механизма преобразования в ПОЛИЗ

}

break;

case LEX\_LEFTHESIS:

stk.push(lextable.table[i]);

break;

case LEX\_RIGHTHESIS:

while (stk.top()->lexema != LEX\_LEFTHESIS)

{

result.push(stk.top());

stk.pop();

}

stk.pop();

if (function)

{

result.push(new LT::Entry('@'));

result.push(new LT::Entry('0' + quantityParm));

function = false;

}

break;

case LEX\_COMMA:

if (function)

quantityParm++;

while (stk.top()->lexema != LEX\_LEFTHESIS)

{

result.push(stk.top());

stk.pop();

}

break;

case LEX\_MORE:

case LEX\_LESS:

result.push(lextable.table[i]);

break;

}

}

while (!stk.empty())

{

result.push(stk.top());

stk.pop();

}

for (int j = lextable\_pos; j < i; j++)

Листинг 1 (продолжение). Программная реализация механизма преобразования в ПОЛИЗ

{

if (!result.empty())

{

lextable.table[j] = result.front();

lextable.table[j]->sn = lextable.table[j - 1]->sn;

lextable.table[j]->tn = lextable.table[j - 1]->tn + 1;

result.pop();

}

else

{

lextable.table[j] = new LT::Entry('@', lextable.table[j]->sn = lextable.table[j - 1]->sn, lextable.table[j]->tn = lextable.table[j - 1]->tn + 1);

}

}

return true;

}

# **Приложение Д**

Листинг 1. Исходный код на языке ассемблера для контрольного примера

.586 ; система команд (процессор Pentium)

.model flat, stdcall ; модель памяти, соглашение о вызовах

includelib kernel32.lib

includelib libucrt.lib

includelib StaticLib.lib

ExitProcess PROTO: dword ; прототип функции для завершения процесса Windows

EXTRN lenght: proc

EXTRN write\_int: proc

EXTRN write\_str : proc

EXTRN copy: proc

EXTRN getLocalTimeAndDate: proc

EXTRN random: proc

EXTRN squareOfNumber: proc

EXTRN factorialOfNumber: proc

EXTRN powNumber: proc

.stack 4096

.const ; сегмент констант - литералы

nulError byte 'error divided by zero', 0

nul sdword 0, 0

L0 byte "Текущее время:", 0

L1 byte "вывод чётных чисел в диапазоне от 10 до 50:", 0

L2 sdword 10

L3 sdword 50

L4 sdword 2

L5 sdword 0

L6 sdword 1

L7 byte "notebook", 0

L8 sdword 4

L9 byte " ", 0

L10 byte "Копирование строк:", 0

L11 byte "Факториал 10:", 0

.data ; сегмент данных - переменные и параметры

timeNow\_main dword ?

i\_main sdword 0

result\_repeat0 sdword 0

s\_main dword ?

res\_main dword ?

num\_main sdword 0

Листинг 1 (продолжение). Исходный код на языке ассемблера для контрольного примера

r\_main sdword 0

.code ; сегмент кода

;----------- MAIN ------------

main PROC

push offset L0

call write\_str

call getLocalTimeAndDate

mov timeNow\_main, eax

push timeNow\_main

call write\_str

push offset L1

call write\_str

push L2

pop i\_main

cyclenext0:

mov edx, i\_main

cmp edx, L3

jg cycle0

push i\_main

push L4

pop ebx

pop eax

cdq

idiv ebx

push edx

pop result\_repeat0

mov edx, result\_repeat0

cmp edx, L5

jz true1

jmp next1

true1:

push i\_main

call write\_int

next1:

push i\_main

push L6

pop ebx

Листинг 1 (продолжение). Исходный код на языке ассемблера для контрольного примера

pop eax

add eax, ebx

push eax

pop i\_main

jmp cyclenext0

cycle0:

mov s\_main, offset L7

push L8

push s\_main

push res\_main

call copy

mov res\_main, eax

push offset L9

call write\_str

push offset L10

call write\_str

push res\_main

call write\_str

push L2

pop num\_main

push num\_main

call factorialOfNumber

push eax

pop r\_main

push offset L9

call write\_str

push offset L11

call write\_str

push r\_main

call write\_int

jmp goodExit

errorExit:

push offset nulError

call write\_str

goodExit:

push 0

call ExitProcess

main ENDP

end main

# **Приложение E**

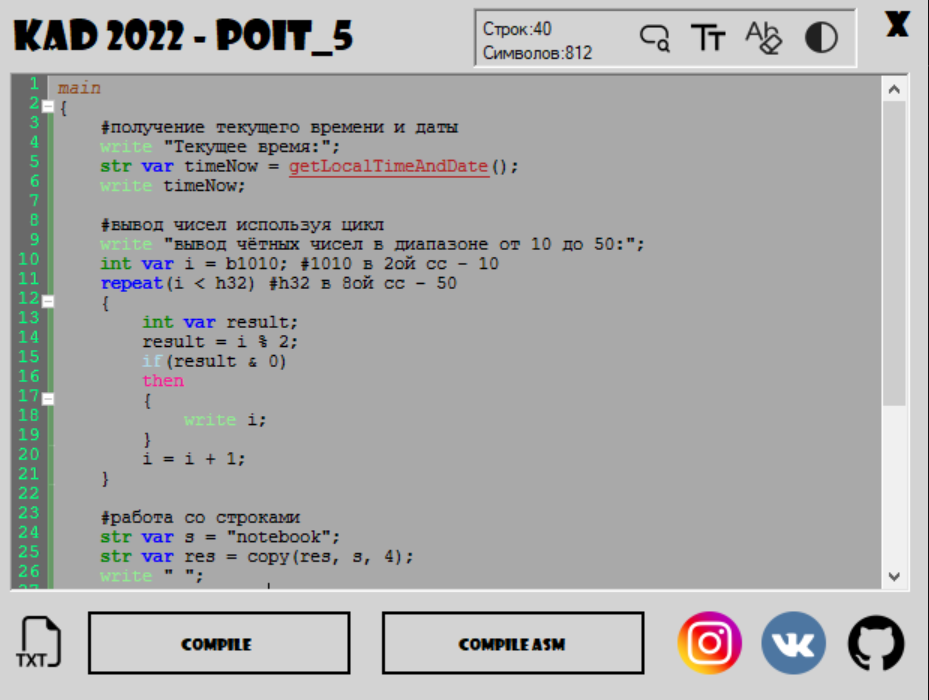


Рисунок 1. Графический интерфейс для написания кода на языке KAD-2022

# **Приложение Ж**

Графический материал. Граф дерева разбора.