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 4**](#_Toc59100349)

[**Глава 1. Спецификация языка программирования 5**](#_Toc59100350)

[1.1 Характеристика языка программирования 5](#_Toc59100351)

[1.2 Алфавит языка 5](#_Toc59100352)

[1.3 Применяемые сепараторы 5](#_Toc59100353)

[1.4 Применяемые кодировки 6](#_Toc59100354)

[1.5 Типы данных 6](#_Toc59100355)

[1.6 Преобразование типов данных 7](#_Toc59100356)

[1.7 Идентификаторы 7](#_Toc59100357)

[1.8 Литералы 8](#_Toc59100358)

[1.9 Объявления данных и область видимости 8](#_Toc59100359)

[1.10 Инициализация данных 8](#_Toc59100360)

[1.11 Инструкции языка 8](#_Toc59100361)

[1.12 Операции языка 9](#_Toc59100362)

[1.13 Выражения и их вычисления 9](#_Toc59100363)

[1.14 Программные конструкции языка 9](#_Toc59100364)

[1.15 Область видимости идентификаторов 10](#_Toc59100365)

[1.16 Семантические проверки 10](#_Toc59100366)

[1.17 Распределение оперативной памяти на этапе выполнения 11](#_Toc59100367)

[1.18 Стандартные библиотеки и их состав 11](#_Toc59100368)

[1.19 Ввод и вывод данных 12](#_Toc59100369)

[1.20 Точка входа 12](#_Toc59100370)

[1.21 Препроцессор 12](#_Toc59100371)

[1.22 Соглашения о вызовах 12](#_Toc59100372)

[1.23 Объектный код 12](#_Toc59100373)

[1.24 Классификация сообщений транслятора 12](#_Toc59100374)

[1.25 Контрольный пример 13](#_Toc59100375)

[**2 Структура транслятора 14**](#_Toc59100376)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 14](#_Toc59100377)

[2.2 Перечень входных параметров транслятора 15](#_Toc59100378)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 15](#_Toc59100379)

[**3 Разработка лексического анализатора 17**](#_Toc59100380)

[3.1 Структура лексического анализатора 17](#_Toc59100381)

[3.2. Контроль входных символов 18](#_Toc59100382)

[3.3 Удаление избыточных символов 18](#_Toc59100383)

[3.4 Перечень ключевых слов 19](#_Toc59100384)

[3.5 Основные структуры данных 20](#_Toc59100385)

[3.6 Принцип обработки ошибок 21](#_Toc59100386)

[3.7 Структура и перечень сообщений лексического анализатора 22](#_Toc59100387)

[3.8 Параметры лексического анализатора 22](#_Toc59100388)

[3.9 Алгоритм лексического анализа 22](#_Toc59100389)

[3.10 Контрольный пример 23](#_Toc59100390)

[**4. Разработка синтаксического анализатора 24**](#_Toc59100391)

[4.1 Структура синтаксического анализатора 24](#_Toc59100392)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 24](#_Toc59100393)

[4.3 Построение конечного магазинного автомата 26](#_Toc59100394)

[4.4 Основные структуры данных 26](#_Toc59100395)

[4.5 Описание алгоритма синтаксического разбора 27](#_Toc59100396)

[4.6 Структура и перечень сообщений синтаксического анализатора 27](#_Toc59100397)

[4.7. Параметры синтаксического анализатора и режимы его работы 27](#_Toc59100398)

[4.8. Принцип обработки ошибок 27](#_Toc59100399)

[4.9. Контрольный пример 28](#_Toc59100400)

[**5 Разработка семантического анализатора 29**](#_Toc59100401)

[5.1 Структура семантического анализатора 29](#_Toc59100402)

[5.2 Функции семантического анализатора 29](#_Toc59100403)

[5.3 Структура и перечень сообщений семантического анализатора 29](#_Toc59100404)

[5.4 Принцип обработки ошибок 29](#_Toc59100405)

[5.5 Контрольный пример 30](#_Toc59100406)

[**6. Вычисление выражений 31**](#_Toc59100407)

[6.1 Выражения, допускаемые языком 31](#_Toc59100408)

[6.2 Польская запись и принцип её построения 31](#_Toc59100409)

[6.3 Программная реализация обработки выражений 31](#_Toc59100410)

[6.4 Контрольный пример 32](#_Toc59100411)

[**7. Генерация кода 33**](#_Toc59100412)

[7.1 Структура генератора кода 33](#_Toc59100413)

[7.2 Представление типов данных в оперативной памяти 33](#_Toc59100414)

[7.3 Статическая библиотека 34](#_Toc59100415)

[7.4 Особенности алгоритма генерации кода 34](#_Toc59100416)

[7.5 Входные параметры генератора кода 34](#_Toc59100417)

[7.6 Контрольный пример 34](#_Toc59100418)

[**8. Тестирование транслятора** 35](#_Toc59100419)

[8.1 Тестирование проверки на допустимость символов 35](#_Toc59100420)

[8.2 Тестирование лексического анализатора 36](#_Toc59100421)

[8.3 Тестирование синтаксического анализатора 36](#_Toc59100422)

[8.4 Тестирование семантического анализатора 38](#_Toc59100423)

[**Заключение 40**](#_Toc59100424)

[**Список использованных источников 41**](#_Toc59100425)

[**Приложение А 42**](#_Toc59100426)

[**Приложение Б 44**](#_Toc59100427)

[**Приложение В 45**](#_Toc59100428)

[**Приложение Г 64**](#_Toc59100429)

[**Приложение Д 66**](#_Toc59100430)

# **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.

Глава 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.

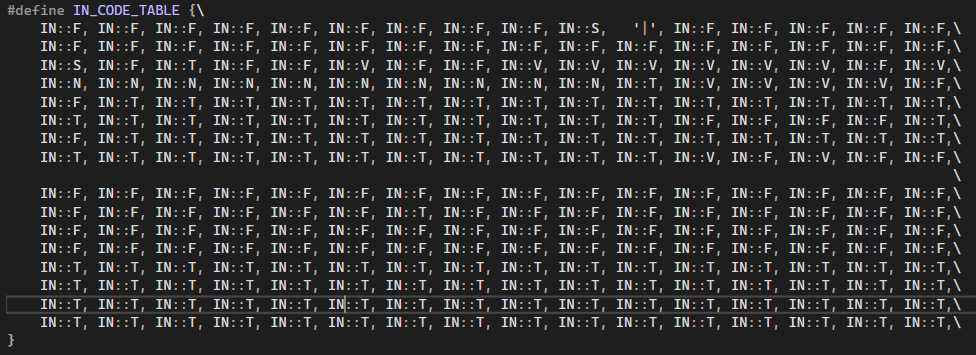


Рисунок 1.1 – Символы, разрешенные к использованию

* 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 |

* 1. **Применяемые кодировки**

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



Рисунок 1.2

* 1. Типы данных

Допускается использование фундаментальных типов данных, определенных в таблице 1.2. Пользовательские типы данных не поддерживаются.

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.  Возможные **арифметические** операции:  **+** (бинарный) – операция сложения;  - (бинарный) – операция вычитания;  **\*** (бинарный) – операция произведения;  **/** (бинарный) – операция деления;  **%** (бинарный) – операция деления нацело;  **=** (бинарный) – оператор присваивания.  Возможные варианты присваивания:  Присваивание целочисленному идентификатору значения другого целочисленного идентификатора, целочисленного литерала или значения целочисленной функции.  Возможные **логические** операции:  **>** (бинарный) – оператор «больше»;  **<** (бинарный) – оператор «меньше»; |
| **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.  Возможные операции:  **=** (бинарный) – оператор присваивания.  Возможные варианты присваивания:  Присваивание строковому идентификатору значения другого строкового идентификатора, строкового литерала или значения строковой функции, а также использование библиотечных функций. |

* 1. **Преобразование типов данных**

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

Предусмотрены целочисленные и строковые литералы. Все литералы являются rvalue. Правила записи приведены в Таблице 1.3.

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. |

**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**.

**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 |

**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>} |

**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 | Наличие функции main |
| 2 | Усечение слишком длинных идентификаторов до 20 символов |
| 3 | Сначала осуществляется проверка на ключевые слова, а затем на  идентификатор. Не допускаются идентификаторы, совпадающие с  ключевыми словами |
| 4 | Нет повторяющихся наименований функций |
| 5 | Нет повторяющихся объявлений идентификаторов |
| 6 | Предварительное объявление, применяемых функций |
| 7 | Предварительное объявление, применяемых идентификаторов. |
| 8 | Соответствие типов формальных и фактических параметров при вызове  функций |
| 9 | Усечение слишком длинного значения str-литерала |
| 10 | Округление слишком большого значения int-литерала |
| 11 | Если ошибка возникает на этапе лексического анализа, синтаксический  анализ не выполняется |
| 12 | Если 3 подряд фразы не разобраны, то работа транслятора останавливается |
| 13 | 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. |

**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\_str | integer | str parm1 - строка | Функция вычисляет длину строки parm1 |
| copy\_str | str\* parm1 | 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> |

**1.19 Ввод и вывод данных**

Ввод данных в языке KAD-2022 не предусмотрен. Вывод данных осуществляется с помощью ключевых слова “write”.

## 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, то есть все параметры передаются в стек справа налево, память освобождает вызываемым кодом.

**1.23 Объектный код**

Язык KAD-2022 транслируется в ассемблер.

**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 |

**1.25 Контрольный пример**

Контрольный пример демонстрирует главные 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 программирования 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 |

## 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. |

# **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.

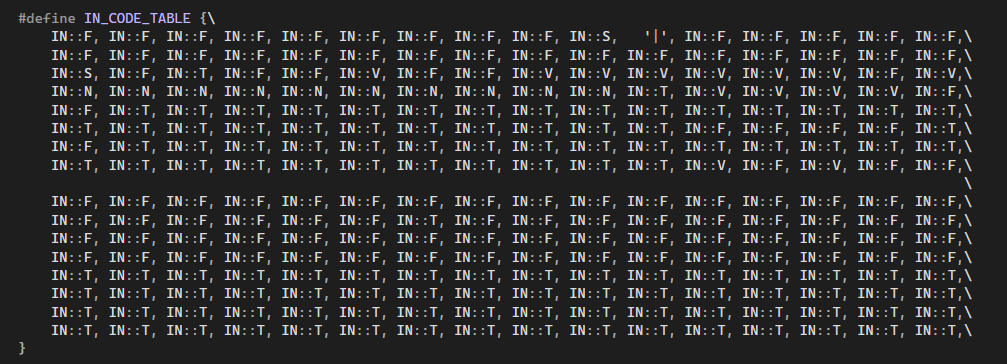
Рисунок 3.2. Таблица контроля входных символов

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 |

Пример реализации таблицы лексем представлен в приложении Б.

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся токен и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Структура конечного автомата и пример графа перехода конечного автомата изображены на рисунках 3.3 и 3.4 соответственно.

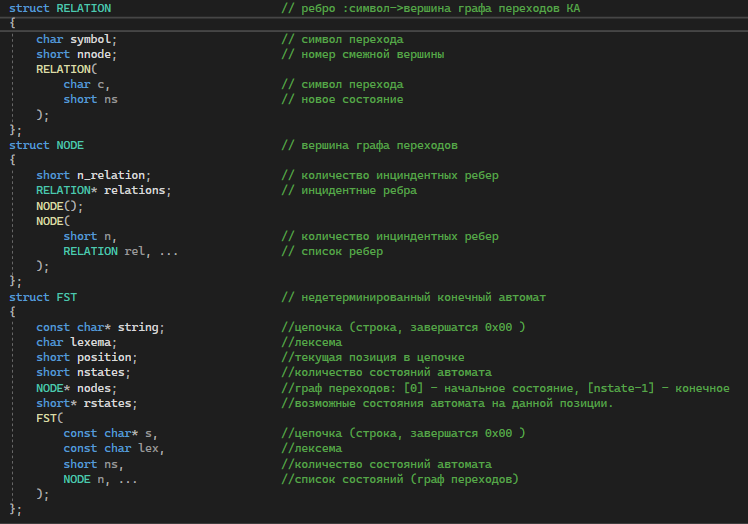


Рисунок 3.3 Структура конечного автомата

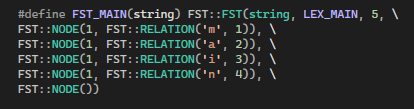


Рисунок 3.4 Пример реализации графа КА для токена main

## 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). 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 лексем представлен на рисунке 3.5. Код C++ со структурой таблицы идентификаторов представлен на рисунке 3.6.

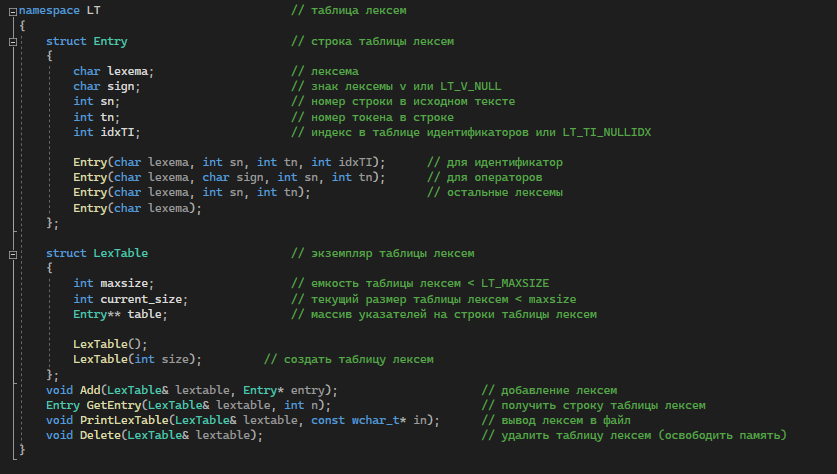


Рисунок 3.5 Структура таблицы лексем

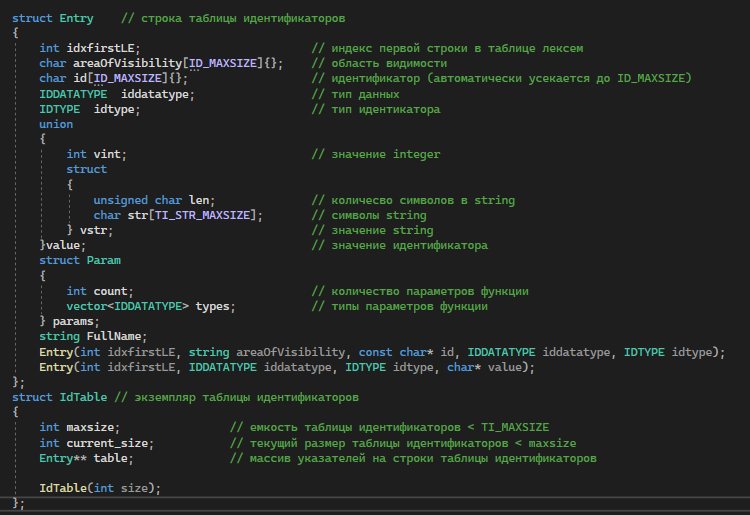


Рисунок 3.6 Структура таблицы идентификаторов

## 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. При возникновении сообщения, лексический анализатор выбрасывает исключение – работа программы останавливается. Перечень сообщений представлен на рисунке 3.7.

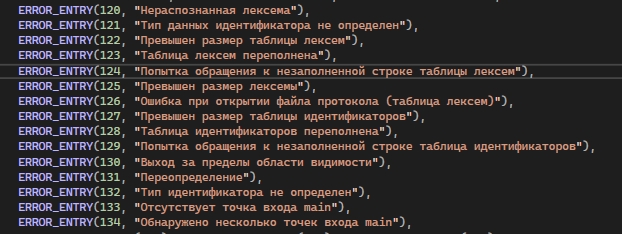


Рисунок 3.7 - Сообщения лексического анализатора

## 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 как в файл журнала, так и в командную строку.

## 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.

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.

S T T R

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

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

Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

Рисунок 4.1 Структура синтаксического анализатора.

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка 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) и правила  имеют вид:

, где ; (или , или );

, где — начальный символ, при этом если такое правило существует, то нетерминал  не встречается в правой части 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.

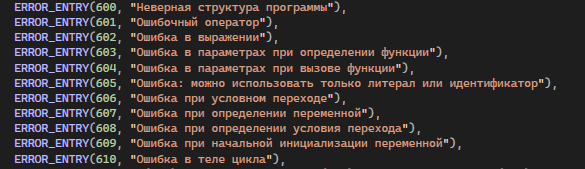


Рисунок 4.3 - Сообщения синтаксического анализатора

## 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.

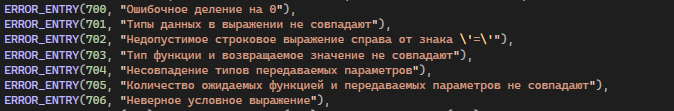


Рисунок 5.2 – Перечень сообщений семантического анализатора

## 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;  x = 1 + “test1”;  } | 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 |

# **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# |

# **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 нулевым символом. |

## 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. Объявление функций статической библиотеки генерируется автоматически.

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\_str(char\* str) | Вычисление длины строки |
| char\* copy\_str(char\* str1, char\* str2, int count) | Копирование определенного количества символов из str2 в строку str1 |

## 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.

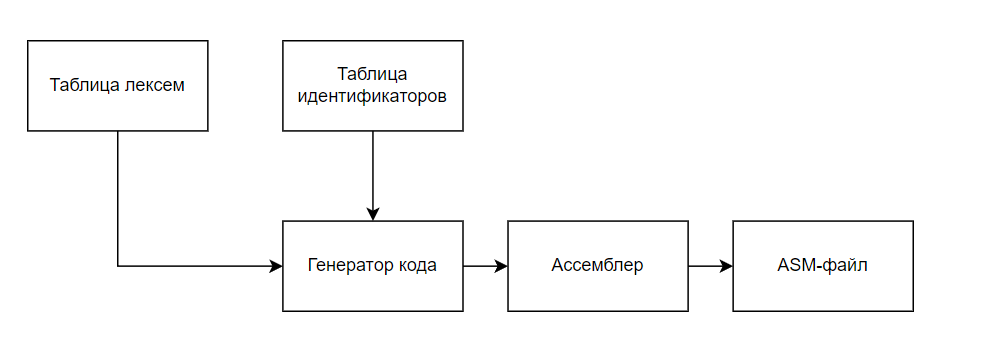


Рисунок 7.2 – Структура генератора кода

## 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 генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Д. Результат работы контрольного примера приведён на рисунке 7.2.

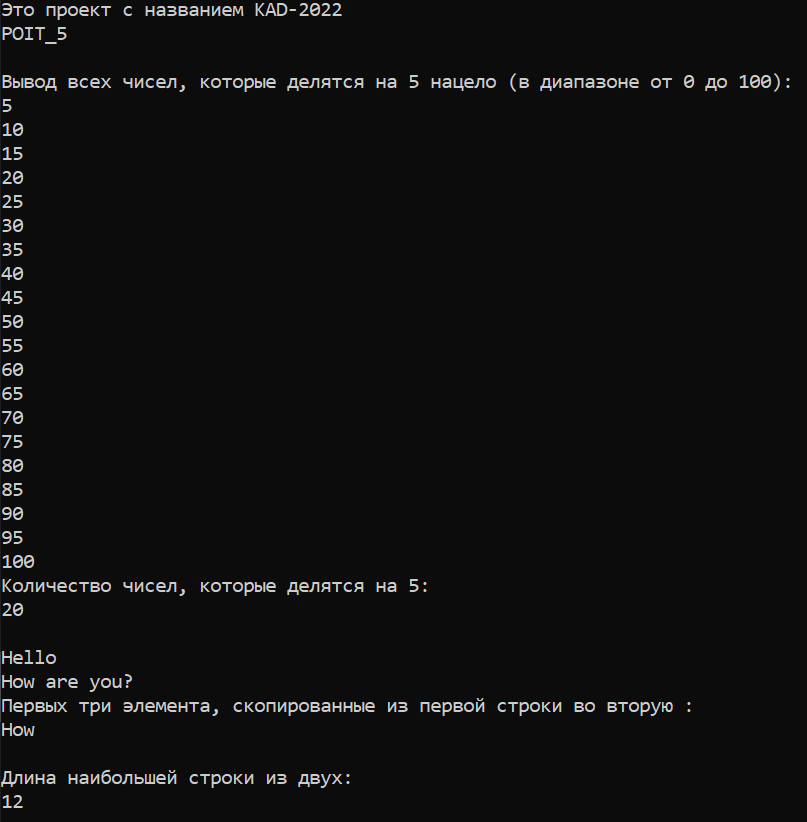


Рисунок 7.2 Результат работы программы на языке KAD-2022

# **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 15 |

## 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 |
| int function FindMaxLen(str param x, str param y){} | Ошибка 133: Отсутствует точка входа main |

Продолжение таблицы 8.2 - Тестирование лексического анализатора

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

Семантический анализ в языке KAD-2022 содержит множество проверок по семантическим правилам, описанным в пункте 1.16. 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;  x = 1 + “test1”;  } | 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 |

# **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 функций стандартной библиотеки;
4. Наличие 5 арифметических операторов для вычисления выражений;
5. Поддержка функций, процедур, операторов цикла и условия;
6. Структурированная и классифицированная система для обработки ошибок пользователя.

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

# **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

int function DivisionByFive(int param count)

{

int var i; i = count;

int var countOfNumber;

repeat(i < h64)

{

int var result;

result = i % 5;

result = result - 1;

if(result < 0)

then

{

write i;

countOfNumber = countOfNumber + 1;

}

i = i + 1;

}

return countOfNumber;

}

int function findMaxStroke(str param strOne, str param strTwo)

{

int var lenghtOfFirstStroke = lenght\_str(strOne);

int var lenghtOfSecondStroke = lenght\_str(strTwo);

int var result;

if(lenghtOfFirstStroke > lenghtOfSecondStroke)

then

{

result = lenghtOfFirstStroke;

}

else

{

result = lenghtOfSecondStroke;

}

return result;

}

main

{

write "Это проект с названием KAD-2022";

str var group = "POIT\_5";

write group;

write " ";

write "Вывод всех чисел, которые делятся на 5 нацело (в диапазоне от 0 до 100):";

int var count = DivisionByFive(b100);

write "Количество чисел, которые делятся на 5: ";

write count;

write " ";

str var strokeOne = "Hello";

str var strokeSecond = "How are you?";

write strokeOne;

write strokeSecond;

write "Первых три элемента, скопированных из первой строки во вторую :";

str var ns = copy\_str(strokeOne, strokeSecond, 3 );

write ns;

write " ";

write "Длина наибольшей строки из двух: ";

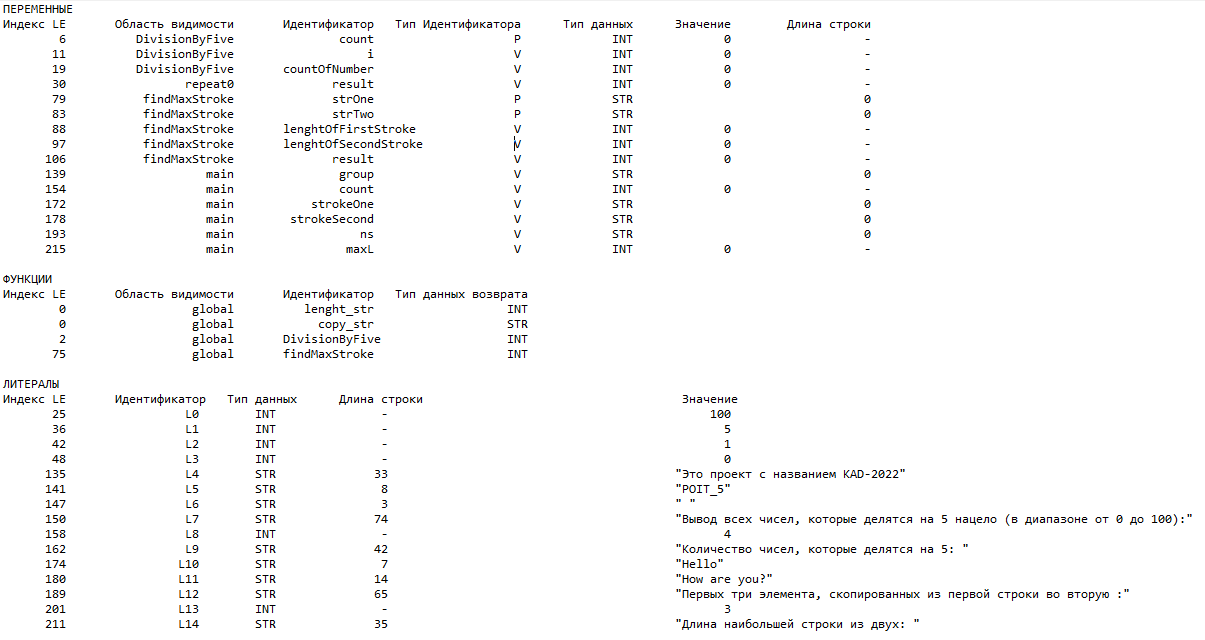
int var maxL = findMaxStroke(strokeOne, strokeSecond);

write maxL;

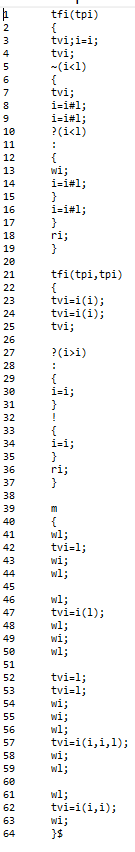
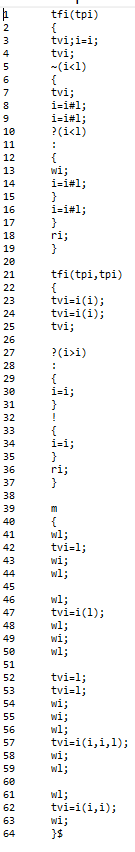
}

# **Приложение Б**

Листинг 1. Таблица идентификаторов контрольного примера



Листинг 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

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(')'))

),

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

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

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 Грейбах

struct Greibach // грамматика Грейбах

{

short size; // количество правил

GRBALPHABET startN; // стартовый символ

GRBALPHABET stbottomT; // дно стека

Rule\* rules; // множество правил

Greibach()

{

size = 0;

startN = 0;

stbottomT = 0;

rules = 0;

};

Greibach(GRBALPHABET pstartN, // стартовый символ

GRBALPHABET pstbottomT, // дно стека

short psize, // количество правил

Rule r, ...); // правила

short getRule( // получить правило, возвращается номер правила или -1

GRBALPHABET pnn, // левый символ правила

Rule& prule); // возвращаемое правило грамматики

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

};

Листинг 4. Разбор исходного кода синтаксическим анализатором

Шаг :Правило Входная лента Стек

0 :S->tfi(F){NrU;}S tfi(tpi){tvi;i=i;tvi;~(i< S$

1 : SAVESTATE: 1

1 : tfi(tpi){tvi;i=i;tvi;~(i< tfi(F){NrU;}S$

2 : fi(tpi){tvi;i=i;tvi;~(i<l fi(F){NrU;}S$

3 : i(tpi){tvi;i=i;tvi;~(i<l) i(F){NrU;}S$

4 : (tpi){tvi;i=i;tvi;~(i<l){ (F){NrU;}S$

5 : tpi){tvi;i=i;tvi;~(i<l){t F){NrU;}S$

6 :F->tP tpi){tvi;i=i;tvi;~(i<l){t F){NrU;}S$

7 : SAVESTATE: 2

7 : tpi){tvi;i=i;tvi;~(i<l){t tP){NrU;}S$

8 : pi){tvi;i=i;tvi;~(i<l){tv P){NrU;}S$

9 :P->pi pi){tvi;i=i;tvi;~(i<l){tv P){NrU;}S$

10 : SAVESTATE: 3

10 : pi){tvi;i=i;tvi;~(i<l){tv pi){NrU;}S$

11 : i){tvi;i=i;tvi;~(i<l){tvi i){NrU;}S$

12 : ){tvi;i=i;tvi;~(i<l){tvi; ){NrU;}S$

13 : {tvi;i=i;tvi;~(i<l){tvi;i {NrU;}S$

14 : tvi;i=i;tvi;~(i<l){tvi;i= NrU;}S$

15 :N->tY;N tvi;i=i;tvi;~(i<l){tvi;i= NrU;}S$

16 : SAVESTATE: 4

16 : tvi;i=i;tvi;~(i<l){tvi;i= tY;NrU;}S$

17 : vi;i=i;tvi;~(i<l){tvi;i=i Y;NrU;}S$

18 :Y->vi vi;i=i;tvi;~(i<l){tvi;i=i Y;NrU;}S$

19 : SAVESTATE: 5

19 : vi;i=i;tvi;~(i<l){tvi;i=i vi;NrU;}S$

20 : i;i=i;tvi;~(i<l){tvi;i=i# i;NrU;}S$

21 : ;i=i;tvi;~(i<l){tvi;i=i#l ;NrU;}S$

22 : i=i;tvi;~(i<l){tvi;i=i#l; NrU;}S$

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

24 : SAVESTATE: 6

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

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

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

27 :E->iM i;tvi;~(i<l){tvi;i=i#l;i= E;NrU;}S$

28 : SAVESTATE: 7

28 : i;tvi;~(i<l){tvi;i=i#l;i= iM;NrU;}S$

29 : ;tvi;~(i<l){tvi;i=i#l;i=i M;NrU;}S$

30 :M-> ;tvi;~(i<l){tvi;i=i#l;i=i M;NrU;}S$

31 : SAVESTATE: 8

31 : ;tvi;~(i<l){tvi;i=i#l;i=i ;NrU;}S$

32 : tvi;~(i<l){tvi;i=i#l;i=i# NrU;}S$

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

34 : SAVESTATE: 9

34 : tvi;~(i<l){tvi;i=i#l;i=i# tY;NrU;}S$

Листинг 4. Разбор исходного кода синтаксическим анализатором (продолжение)

35 : vi;~(i<l){tvi;i=i#l;i=i#l Y;NrU;}S$

36 :Y->vi vi;~(i<l){tvi;i=i#l;i=i#l Y;NrU;}S$

37 : SAVESTATE: 10

37 : vi;~(i<l){tvi;i=i#l;i=i#l vi;NrU;}S$

38 : i;~(i<l){tvi;i=i#l;i=i#l; i;NrU;}S$

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

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

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

42 : SAVESTATE: 11

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

43 : (i<l){tvi;i=i#l;i=i#l;?(i KZNrU;}S$

44 :K->(U<U) (i<l){tvi;i=i#l;i=i#l;?(i KZNrU;}S$

45 : SAVESTATE: 12

45 : (i<l){tvi;i=i#l;i=i#l;?(i (U<U)ZNrU;}S$

46 : i<l){tvi;i=i#l;i=i#l;?(i< U<U)ZNrU;}S$

47 :U->i i<l){tvi;i=i#l;i=i#l;?(i< U<U)ZNrU;}S$

48 : SAVESTATE: 13

48 : i<l){tvi;i=i#l;i=i#l;?(i< i<U)ZNrU;}S$

49 : <l){tvi;i=i#l;i=i#l;?(i<l <U)ZNrU;}S$

50 : l){tvi;i=i#l;i=i#l;?(i<l) U)ZNrU;}S$

51 :U->l l){tvi;i=i#l;i=i#l;?(i<l) U)ZNrU;}S$

52 : SAVESTATE: 14

52 : l){tvi;i=i#l;i=i#l;?(i<l) l)ZNrU;}S$

53 : ){tvi;i=i#l;i=i#l;?(i<l): )ZNrU;}S$

54 : {tvi;i=i#l;i=i#l;?(i<l):{ ZNrU;}S$

55 :Z->{N} {tvi;i=i#l;i=i#l;?(i<l):{ ZNrU;}S$

56 : SAVESTATE: 15

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

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

58 :N->tY;N tvi;i=i#l;i=i#l;?(i<l):{w N}NrU;}S$

59 : SAVESTATE: 16

59 : tvi;i=i#l;i=i#l;?(i<l):{w tY;N}NrU;}S$

60 : vi;i=i#l;i=i#l;?(i<l):{wi Y;N}NrU;}S$

61 :Y->vi vi;i=i#l;i=i#l;?(i<l):{wi Y;N}NrU;}S$

62 : SAVESTATE: 17

62 : vi;i=i#l;i=i#l;?(i<l):{wi vi;N}NrU;}S$

63 : i;i=i#l;i=i#l;?(i<l):{wi; i;N}NrU;}S$

64 : ;i=i#l;i=i#l;?(i<l):{wi;i ;N}NrU;}S$

65 : i=i#l;i=i#l;?(i<l):{wi;i= N}NrU;}S$

66 :N->i=E;N i=i#l;i=i#l;?(i<l):{wi;i= N}NrU;}S$

67 : SAVESTATE: 18

67 : i=i#l;i=i#l;?(i<l):{wi;i= i=E;N}NrU;}S$

68 : =i#l;i=i#l;?(i<l):{wi;i=i =E;N}NrU;}S$

69 : i#l;i=i#l;?(i<l):{wi;i=i# E;N}NrU;}S$

70 :E->iM i#l;i=i#l;?(i<l):{wi;i=i# E;N}NrU;}S$

71 : SAVESTATE: 19

71 : i#l;i=i#l;?(i<l):{wi;i=i# iM;N}NrU;}S$

72 : #l;i=i#l;?(i<l):{wi;i=i#l M;N}NrU;}S$

73 :M->#E #l;i=i#l;?(i<l):{wi;i=i#l M;N}NrU;}S$

74 : SAVESTATE: 20

74 : #l;i=i#l;?(i<l):{wi;i=i#l #E;N}NrU;}S$

75 : l;i=i#l;?(i<l):{wi;i=i#l; E;N}NrU;}S$

76 :E->lM l;i=i#l;?(i<l):{wi;i=i#l; E;N}NrU;}S$

77 : SAVESTATE: 21

77 : l;i=i#l;?(i<l):{wi;i=i#l; lM;N}NrU;}S$

78 : ;i=i#l;?(i<l):{wi;i=i#l;} M;N}NrU;}S$

79 :M-> ;i=i#l;?(i<l):{wi;i=i#l;} M;N}NrU;}S$

80 : SAVESTATE: 22

80 : ;i=i#l;?(i<l):{wi;i=i#l;} ;N}NrU;}S$

81 : i=i#l;?(i<l):{wi;i=i#l;}i N}NrU;}S$

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

82 :N->i=E;N i=i#l;?(i<l):{wi;i=i#l;}i N}NrU;}S$

83 : SAVESTATE: 23

83 : i=i#l;?(i<l):{wi;i=i#l;}i i=E;N}NrU;}S$

84 : =i#l;?(i<l):{wi;i=i#l;}i= =E;N}NrU;}S$

85 : i#l;?(i<l):{wi;i=i#l;}i=i E;N}NrU;}S$

86 :E->iM i#l;?(i<l):{wi;i=i#l;}i=i E;N}NrU;}S$

87 : SAVESTATE: 24

87 : i#l;?(i<l):{wi;i=i#l;}i=i iM;N}NrU;}S$

88 : #l;?(i<l):{wi;i=i#l;}i=i# M;N}NrU;}S$

89 :M->#E #l;?(i<l):{wi;i=i#l;}i=i# M;N}NrU;}S$

90 : SAVESTATE: 25

90 : #l;?(i<l):{wi;i=i#l;}i=i# #E;N}NrU;}S$

91 : l;?(i<l):{wi;i=i#l;}i=i#l E;N}NrU;}S$

92 :E->lM l;?(i<l):{wi;i=i#l;}i=i#l E;N}NrU;}S$

93 : SAVESTATE: 26

93 : l;?(i<l):{wi;i=i#l;}i=i#l lM;N}NrU;}S$

94 : ;?(i<l):{wi;i=i#l;}i=i#l; M;N}NrU;}S$

95 :M-> ;?(i<l):{wi;i=i#l;}i=i#l; M;N}NrU;}S$

96 : SAVESTATE: 27

96 : ;?(i<l):{wi;i=i#l;}i=i#l; ;N}NrU;}S$

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

98 :N->?KJN ?(i<l):{wi;i=i#l;}i=i#l;} N}NrU;}S$

99 : SAVESTATE: 28

99 : ?(i<l):{wi;i=i#l;}i=i#l;} ?KJN}NrU;}S$

100 : (i<l):{wi;i=i#l;}i=i#l;}r KJN}NrU;}S$

101 :K->(U<U) (i<l):{wi;i=i#l;}i=i#l;}r KJN}NrU;}S$

102 : SAVESTATE: 29

102 : (i<l):{wi;i=i#l;}i=i#l;}r (U<U)JN}NrU;}S$

103 : i<l):{wi;i=i#l;}i=i#l;}ri U<U)JN}NrU;}S$

104 :U->i i<l):{wi;i=i#l;}i=i#l;}ri U<U)JN}NrU;}S$

105 : SAVESTATE: 30

105 : i<l):{wi;i=i#l;}i=i#l;}ri i<U)JN}NrU;}S$

106 : <l):{wi;i=i#l;}i=i#l;}ri; <U)JN}NrU;}S$

107 : l):{wi;i=i#l;}i=i#l;}ri;} U)JN}NrU;}S$

108 :U->l l):{wi;i=i#l;}i=i#l;}ri;} U)JN}NrU;}S$

109 : SAVESTATE: 31

109 : l):{wi;i=i#l;}i=i#l;}ri;} l)JN}NrU;}S$

110 : ):{wi;i=i#l;}i=i#l;}ri;}t )JN}NrU;}S$

111 : :{wi;i=i#l;}i=i#l;}ri;}tf JN}NrU;}S$

112 :J->:{N} :{wi;i=i#l;}i=i#l;}ri;}tf JN}NrU;}S$

113 : SAVESTATE: 32

113 : :{wi;i=i#l;}i=i#l;}ri;}tf :{N}N}NrU;}S$

114 : {wi;i=i#l;}i=i#l;}ri;}tfi {N}N}NrU;}S$

115 : wi;i=i#l;}i=i#l;}ri;}tfi( N}N}NrU;}S$

116 :N->wE;N wi;i=i#l;}i=i#l;}ri;}tfi( N}N}NrU;}S$

117 : SAVESTATE: 33

117 : wi;i=i#l;}i=i#l;}ri;}tfi( wE;N}N}NrU;}S$

118 : i;i=i#l;}i=i#l;}ri;}tfi(t E;N}N}NrU;}S$

119 :E->iM i;i=i#l;}i=i#l;}ri;}tfi(t E;N}N}NrU;}S$

120 : SAVESTATE: 34

120 : i;i=i#l;}i=i#l;}ri;}tfi(t iM;N}N}NrU;}S$

121 : ;i=i#l;}i=i#l;}ri;}tfi(tp M;N}N}NrU;}S$

122 :M-> ;i=i#l;}i=i#l;}ri;}tfi(tp M;N}N}NrU;}S$

123 : SAVESTATE: 35

123 : ;i=i#l;}i=i#l;}ri;}tfi(tp ;N}N}NrU;}S$

124 : i=i#l;}i=i#l;}ri;}tfi(tpi N}N}NrU;}S$

125 :N->i=E;N i=i#l;}i=i#l;}ri;}tfi(tpi N}N}NrU;}S$

126 : SAVESTATE: 36

126 : i=i#l;}i=i#l;}ri;}tfi(tpi i=E;N}N}NrU;}S$

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

127 : =i#l;}i=i#l;}ri;}tfi(tpi, =E;N}N}NrU;}S$

128 : i#l;}i=i#l;}ri;}tfi(tpi,t E;N}N}NrU;}S$

129 :E->iM i#l;}i=i#l;}ri;}tfi(tpi,t E;N}N}NrU;}S$

130 : SAVESTATE: 37

130 : i#l;}i=i#l;}ri;}tfi(tpi,t iM;N}N}NrU;}S$

131 : #l;}i=i#l;}ri;}tfi(tpi,tp M;N}N}NrU;}S$

132 :M->#E #l;}i=i#l;}ri;}tfi(tpi,tp M;N}N}NrU;}S$

133 : SAVESTATE: 38

133 : #l;}i=i#l;}ri;}tfi(tpi,tp #E;N}N}NrU;}S$

134 : l;}i=i#l;}ri;}tfi(tpi,tpi E;N}N}NrU;}S$

135 :E->lM l;}i=i#l;}ri;}tfi(tpi,tpi E;N}N}NrU;}S$

136 : SAVESTATE: 39

136 : l;}i=i#l;}ri;}tfi(tpi,tpi lM;N}N}NrU;}S$

137 : ;}i=i#l;}ri;}tfi(tpi,tpi) M;N}N}NrU;}S$

138 :M-> ;}i=i#l;}ri;}tfi(tpi,tpi) M;N}N}NrU;}S$

139 : SAVESTATE: 40

139 : ;}i=i#l;}ri;}tfi(tpi,tpi) ;N}N}NrU;}S$

140 : }i=i#l;}ri;}tfi(tpi,tpi){ N}N}NrU;}S$

141 :N-> }i=i#l;}ri;}tfi(tpi,tpi){ N}N}NrU;}S$

142 : SAVESTATE: 41

142 : }i=i#l;}ri;}tfi(tpi,tpi){ }N}NrU;}S$

143 : i=i#l;}ri;}tfi(tpi,tpi){t N}NrU;}S$

144 :N->i=E;N i=i#l;}ri;}tfi(tpi,tpi){t N}NrU;}S$

145 : SAVESTATE: 42

145 : i=i#l;}ri;}tfi(tpi,tpi){t i=E;N}NrU;}S$

146 : =i#l;}ri;}tfi(tpi,tpi){tv =E;N}NrU;}S$

147 : i#l;}ri;}tfi(tpi,tpi){tvi E;N}NrU;}S$

148 :E->iM i#l;}ri;}tfi(tpi,tpi){tvi E;N}NrU;}S$

149 : SAVESTATE: 43

149 : i#l;}ri;}tfi(tpi,tpi){tvi iM;N}NrU;}S$

150 : #l;}ri;}tfi(tpi,tpi){tvi= M;N}NrU;}S$

151 :M->#E #l;}ri;}tfi(tpi,tpi){tvi= M;N}NrU;}S$

152 : SAVESTATE: 44

152 : #l;}ri;}tfi(tpi,tpi){tvi= #E;N}NrU;}S$

153 : l;}ri;}tfi(tpi,tpi){tvi=i E;N}NrU;}S$

154 :E->lM l;}ri;}tfi(tpi,tpi){tvi=i E;N}NrU;}S$

155 : SAVESTATE: 45

155 : l;}ri;}tfi(tpi,tpi){tvi=i lM;N}NrU;}S$

156 : ;}ri;}tfi(tpi,tpi){tvi=i( M;N}NrU;}S$

157 :M-> ;}ri;}tfi(tpi,tpi){tvi=i( M;N}NrU;}S$

158 : SAVESTATE: 46

158 : ;}ri;}tfi(tpi,tpi){tvi=i( ;N}NrU;}S$

159 : }ri;}tfi(tpi,tpi){tvi=i(i N}NrU;}S$

160 :N-> }ri;}tfi(tpi,tpi){tvi=i(i N}NrU;}S$

161 : SAVESTATE: 47

161 : }ri;}tfi(tpi,tpi){tvi=i(i }NrU;}S$

162 : ri;}tfi(tpi,tpi){tvi=i(i) NrU;}S$

163 :N-> ri;}tfi(tpi,tpi){tvi=i(i) NrU;}S$

164 : SAVESTATE: 48

164 : ri;}tfi(tpi,tpi){tvi=i(i) rU;}S$

165 : i;}tfi(tpi,tpi){tvi=i(i); U;}S$

166 :U->i i;}tfi(tpi,tpi){tvi=i(i); U;}S$

167 : SAVESTATE: 49

167 : i;}tfi(tpi,tpi){tvi=i(i); i;}S$

168 : ;}tfi(tpi,tpi){tvi=i(i);t ;}S$

169 : }tfi(tpi,tpi){tvi=i(i);tv }S$

170 : tfi(tpi,tpi){tvi=i(i);tvi S$

171 :S->tfi(F){NrU;}S tfi(tpi,tpi){tvi=i(i);tvi S$

172 : SAVESTATE: 50

172 : tfi(tpi,tpi){tvi=i(i);tvi tfi(F){NrU;}S$

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

173 : fi(tpi,tpi){tvi=i(i);tvi= fi(F){NrU;}S$

174 : i(tpi,tpi){tvi=i(i);tvi=i i(F){NrU;}S$

175 : (tpi,tpi){tvi=i(i);tvi=i( (F){NrU;}S$

176 : tpi,tpi){tvi=i(i);tvi=i(i F){NrU;}S$

177 :F->tP tpi,tpi){tvi=i(i);tvi=i(i F){NrU;}S$

178 : SAVESTATE: 51

178 : tpi,tpi){tvi=i(i);tvi=i(i tP){NrU;}S$

179 : pi,tpi){tvi=i(i);tvi=i(i) P){NrU;}S$

180 :P->pi pi,tpi){tvi=i(i);tvi=i(i) P){NrU;}S$

181 : SAVESTATE: 52

181 : pi,tpi){tvi=i(i);tvi=i(i) pi){NrU;}S$

182 : i,tpi){tvi=i(i);tvi=i(i); i){NrU;}S$

183 : ,tpi){tvi=i(i);tvi=i(i);t ){NrU;}S$

184 : TNS\_NS\_NORULECHAIN/NS\_NORULE

184 : RESTATE

184 : pi,tpi){tvi=i(i);tvi=i(i) P){NrU;}S$

185 : TNS\_NS\_NORULECHAIN/NS\_NORULE

185 : RESTATE

185 : tpi,tpi){tvi=i(i);tvi=i(i F){NrU;}S$

186 :F->tP,F tpi,tpi){tvi=i(i);tvi=i(i F){NrU;}S$

187 : SAVESTATE: 51

187 : tpi,tpi){tvi=i(i);tvi=i(i tP,F){NrU;}S$

188 : pi,tpi){tvi=i(i);tvi=i(i) P,F){NrU;}S$

189 :P->pi pi,tpi){tvi=i(i);tvi=i(i) P,F){NrU;}S$

190 : SAVESTATE: 52

190 : pi,tpi){tvi=i(i);tvi=i(i) pi,F){NrU;}S$

191 : i,tpi){tvi=i(i);tvi=i(i); i,F){NrU;}S$

192 : ,tpi){tvi=i(i);tvi=i(i);t ,F){NrU;}S$

193 : tpi){tvi=i(i);tvi=i(i);tv F){NrU;}S$

194 :F->tP tpi){tvi=i(i);tvi=i(i);tv F){NrU;}S$

195 : SAVESTATE: 53

195 : tpi){tvi=i(i);tvi=i(i);tv tP){NrU;}S$

196 : pi){tvi=i(i);tvi=i(i);tvi P){NrU;}S$

197 :P->pi pi){tvi=i(i);tvi=i(i);tvi P){NrU;}S$

198 : SAVESTATE: 54

198 : pi){tvi=i(i);tvi=i(i);tvi pi){NrU;}S$

199 : i){tvi=i(i);tvi=i(i);tvi; i){NrU;}S$

200 : ){tvi=i(i);tvi=i(i);tvi;? ){NrU;}S$

201 : {tvi=i(i);tvi=i(i);tvi;?( {NrU;}S$

202 : tvi=i(i);tvi=i(i);tvi;?(i NrU;}S$

203 :N->tY;N tvi=i(i);tvi=i(i);tvi;?(i NrU;}S$

204 : SAVESTATE: 55

204 : tvi=i(i);tvi=i(i);tvi;?(i tY;NrU;}S$

205 : vi=i(i);tvi=i(i);tvi;?(i> Y;NrU;}S$

206 :Y->vi vi=i(i);tvi=i(i);tvi;?(i> Y;NrU;}S$

207 : SAVESTATE: 56

207 : vi=i(i);tvi=i(i);tvi;?(i> vi;NrU;}S$

208 : i=i(i);tvi=i(i);tvi;?(i>i i;NrU;}S$

209 : =i(i);tvi=i(i);tvi;?(i>i) ;NrU;}S$

210 : TNS\_NS\_NORULECHAIN/NS\_NORULE

210 : RESTATE

210 : vi=i(i);tvi=i(i);tvi;?(i> Y;NrU;}S$

211 :Y->vi=L vi=i(i);tvi=i(i);tvi;?(i> Y;NrU;}S$

212 : SAVESTATE: 56

212 : vi=i(i);tvi=i(i);tvi;?(i> vi=L;NrU;}S$

213 : i=i(i);tvi=i(i);tvi;?(i>i i=L;NrU;}S$

214 : =i(i);tvi=i(i);tvi;?(i>i) =L;NrU;}S$

215 : i(i);tvi=i(i);tvi;?(i>i): L;NrU;}S$

216 :L->i i(i);tvi=i(i);tvi;?(i>i): L;NrU;}S$

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

217 : SAVESTATE: 57

217 : i(i);tvi=i(i);tvi;?(i>i): i;NrU;}S$

218 : (i);tvi=i(i);tvi;?(i>i):{ ;NrU;}S$

219 : TNS\_NS\_NORULECHAIN/NS\_NORULE

219 : RESTATE

219 : i(i);tvi=i(i);tvi;?(i>i): L;NrU;}S$

220 :L->i(U) i(i);tvi=i(i);tvi;?(i>i): L;NrU;}S$

221 : SAVESTATE: 57

221 : i(i);tvi=i(i);tvi;?(i>i): i(U);NrU;}S$

222 : (i);tvi=i(i);tvi;?(i>i):{ (U);NrU;}S$

223 : i);tvi=i(i);tvi;?(i>i):{i U);NrU;}S$

224 :U->i i);tvi=i(i);tvi;?(i>i):{i U);NrU;}S$

225 : SAVESTATE: 58

225 : i);tvi=i(i);tvi;?(i>i):{i i);NrU;}S$

226 : );tvi=i(i);tvi;?(i>i):{i= );NrU;}S$

227 : ;tvi=i(i);tvi;?(i>i):{i=i ;NrU;}S$

228 : tvi=i(i);tvi;?(i>i):{i=i; NrU;}S$

229 :N->tY;N tvi=i(i);tvi;?(i>i):{i=i; NrU;}S$

230 : SAVESTATE: 59

230 : tvi=i(i);tvi;?(i>i):{i=i; tY;NrU;}S$

231 : vi=i(i);tvi;?(i>i):{i=i;} Y;NrU;}S$

232 :Y->vi vi=i(i);tvi;?(i>i):{i=i;} Y;NrU;}S$

233 : SAVESTATE: 60

233 : vi=i(i);tvi;?(i>i):{i=i;} vi;NrU;}S$

234 : i=i(i);tvi;?(i>i):{i=i;}! i;NrU;}S$

235 : =i(i);tvi;?(i>i):{i=i;}!{ ;NrU;}S$

236 : TNS\_NS\_NORULECHAIN/NS\_NORULE

236 : RESTATE

236 : vi=i(i);tvi;?(i>i):{i=i;} Y;NrU;}S$

237 :Y->vi=L vi=i(i);tvi;?(i>i):{i=i;} Y;NrU;}S$

238 : SAVESTATE: 60

238 : vi=i(i);tvi;?(i>i):{i=i;} vi=L;NrU;}S$

239 : i=i(i);tvi;?(i>i):{i=i;}! i=L;NrU;}S$

240 : =i(i);tvi;?(i>i):{i=i;}!{ =L;NrU;}S$

241 : i(i);tvi;?(i>i):{i=i;}!{i L;NrU;}S$

242 :L->i i(i);tvi;?(i>i):{i=i;}!{i L;NrU;}S$

243 : SAVESTATE: 61

243 : i(i);tvi;?(i>i):{i=i;}!{i i;NrU;}S$

244 : (i);tvi;?(i>i):{i=i;}!{i= ;NrU;}S$

245 : TNS\_NS\_NORULECHAIN/NS\_NORULE

245 : RESTATE

245 : i(i);tvi;?(i>i):{i=i;}!{i L;NrU;}S$

246 :L->i(U) i(i);tvi;?(i>i):{i=i;}!{i L;NrU;}S$

247 : SAVESTATE: 61

247 : i(i);tvi;?(i>i):{i=i;}!{i i(U);NrU;}S$

248 : (i);tvi;?(i>i):{i=i;}!{i= (U);NrU;}S$

249 : i);tvi;?(i>i):{i=i;}!{i=i U);NrU;}S$

250 :U->i i);tvi;?(i>i):{i=i;}!{i=i U);NrU;}S$

251 : SAVESTATE: 62

251 : i);tvi;?(i>i):{i=i;}!{i=i i);NrU;}S$

252 : );tvi;?(i>i):{i=i;}!{i=i; );NrU;}S$

253 : ;tvi;?(i>i):{i=i;}!{i=i;} ;NrU;}S$

254 : tvi;?(i>i):{i=i;}!{i=i;}r NrU;}S$

255 :N->tY;N tvi;?(i>i):{i=i;}!{i=i;}r NrU;}S$

256 : SAVESTATE: 63

256 : tvi;?(i>i):{i=i;}!{i=i;}r tY;NrU;}S$

257 : vi;?(i>i):{i=i;}!{i=i;}ri Y;NrU;}S$

258 :Y->vi vi;?(i>i):{i=i;}!{i=i;}ri Y;NrU;}S$

259 : SAVESTATE: 64

259 : vi;?(i>i):{i=i;}!{i=i;}ri vi;NrU;}S$

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

260 : i;?(i>i):{i=i;}!{i=i;}ri; i;NrU;}S$

261 : ;?(i>i):{i=i;}!{i=i;}ri;} ;NrU;}S$

262 : ?(i>i):{i=i;}!{i=i;}ri;}m NrU;}S$

263 :N->?KJN ?(i>i):{i=i;}!{i=i;}ri;}m NrU;}S$

264 : SAVESTATE: 65

264 : ?(i>i):{i=i;}!{i=i;}ri;}m ?KJNrU;}S$

265 : (i>i):{i=i;}!{i=i;}ri;}m{ KJNrU;}S$

266 :K->(U<U) (i>i):{i=i;}!{i=i;}ri;}m{ KJNrU;}S$

267 : SAVESTATE: 66

267 : (i>i):{i=i;}!{i=i;}ri;}m{ (U<U)JNrU;}S$

268 : i>i):{i=i;}!{i=i;}ri;}m{w U<U)JNrU;}S$

269 :U->i i>i):{i=i;}!{i=i;}ri;}m{w U<U)JNrU;}S$

270 : SAVESTATE: 67

270 : i>i):{i=i;}!{i=i;}ri;}m{w i<U)JNrU;}S$

271 : >i):{i=i;}!{i=i;}ri;}m{wl <U)JNrU;}S$

272 : TNS\_NS\_NORULECHAIN/NS\_NORULE

272 : RESTATE

272 : i>i):{i=i;}!{i=i;}ri;}m{w U<U)JNrU;}S$

273 :U->iU i>i):{i=i;}!{i=i;}ri;}m{w U<U)JNrU;}S$

274 : SAVESTATE: 67

274 : i>i):{i=i;}!{i=i;}ri;}m{w iU<U)JNrU;}S$

275 : >i):{i=i;}!{i=i;}ri;}m{wl U<U)JNrU;}S$

276 : TNS\_NS\_NORULECHAIN/NS\_NORULE

276 : RESTATE

276 : i>i):{i=i;}!{i=i;}ri;}m{w U<U)JNrU;}S$

277 :U->i,U i>i):{i=i;}!{i=i;}ri;}m{w U<U)JNrU;}S$

278 : SAVESTATE: 67

278 : i>i):{i=i;}!{i=i;}ri;}m{w i,U<U)JNrU;}S$

279 : >i):{i=i;}!{i=i;}ri;}m{wl ,U<U)JNrU;}S$

280 : TNS\_NS\_NORULECHAIN/NS\_NORULE

280 : RESTATE

280 : i>i):{i=i;}!{i=i;}ri;}m{w U<U)JNrU;}S$

281 : TNS\_NS\_NORULECHAIN/NS\_NORULE

281 : RESTATE

281 : (i>i):{i=i;}!{i=i;}ri;}m{ KJNrU;}S$

282 :K->(U>U) (i>i):{i=i;}!{i=i;}ri;}m{ KJNrU;}S$

283 : SAVESTATE: 66

283 : (i>i):{i=i;}!{i=i;}ri;}m{ (U>U)JNrU;}S$

284 : i>i):{i=i;}!{i=i;}ri;}m{w U>U)JNrU;}S$

285 :U->i i>i):{i=i;}!{i=i;}ri;}m{w U>U)JNrU;}S$

286 : SAVESTATE: 67

286 : i>i):{i=i;}!{i=i;}ri;}m{w i>U)JNrU;}S$

287 : >i):{i=i;}!{i=i;}ri;}m{wl >U)JNrU;}S$

288 : i):{i=i;}!{i=i;}ri;}m{wl; U)JNrU;}S$

289 :U->i i):{i=i;}!{i=i;}ri;}m{wl; U)JNrU;}S$

290 : SAVESTATE: 68

290 : i):{i=i;}!{i=i;}ri;}m{wl; i)JNrU;}S$

291 : ):{i=i;}!{i=i;}ri;}m{wl;t )JNrU;}S$

292 : :{i=i;}!{i=i;}ri;}m{wl;tv JNrU;}S$

293 :J->:{N} :{i=i;}!{i=i;}ri;}m{wl;tv JNrU;}S$

294 : SAVESTATE: 69

294 : :{i=i;}!{i=i;}ri;}m{wl;tv :{N}NrU;}S$

295 : {i=i;}!{i=i;}ri;}m{wl;tvi {N}NrU;}S$

296 : i=i;}!{i=i;}ri;}m{wl;tvi= N}NrU;}S$

297 :N->i=E;N i=i;}!{i=i;}ri;}m{wl;tvi= N}NrU;}S$

298 : SAVESTATE: 70

298 : i=i;}!{i=i;}ri;}m{wl;tvi= i=E;N}NrU;}S$

299 : =i;}!{i=i;}ri;}m{wl;tvi=l =E;N}NrU;}S$

300 : i;}!{i=i;}ri;}m{wl;tvi=l; E;N}NrU;}S$

301 :E->iM i;}!{i=i;}ri;}m{wl;tvi=l; E;N}NrU;}S$

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

302 : SAVESTATE: 71

302 : i;}!{i=i;}ri;}m{wl;tvi=l; iM;N}NrU;}S$

303 : ;}!{i=i;}ri;}m{wl;tvi=l;w M;N}NrU;}S$

304 :M-> ;}!{i=i;}ri;}m{wl;tvi=l;w M;N}NrU;}S$

305 : SAVESTATE: 72

305 : ;}!{i=i;}ri;}m{wl;tvi=l;w ;N}NrU;}S$

306 : }!{i=i;}ri;}m{wl;tvi=l;wi N}NrU;}S$

307 :N-> }!{i=i;}ri;}m{wl;tvi=l;wi N}NrU;}S$

308 : SAVESTATE: 73

308 : }!{i=i;}ri;}m{wl;tvi=l;wi }NrU;}S$

309 : !{i=i;}ri;}m{wl;tvi=l;wi; NrU;}S$

310 :N-> !{i=i;}ri;}m{wl;tvi=l;wi; NrU;}S$

311 : SAVESTATE: 74

311 : !{i=i;}ri;}m{wl;tvi=l;wi; rU;}S$

312 : TNS\_NS\_NORULECHAIN/NS\_NORULE

312 : RESTATE

312 : !{i=i;}ri;}m{wl;tvi=l;wi; NrU;}S$

313 : TNS\_NS\_NORULECHAIN/NS\_NORULE

313 : RESTATE

313 : }!{i=i;}ri;}m{wl;tvi=l;wi N}NrU;}S$

314 : TNS\_NS\_NORULECHAIN/NS\_NORULE

314 : RESTATE

314 : ;}!{i=i;}ri;}m{wl;tvi=l;w M;N}NrU;}S$

315 : TNS\_NS\_NORULECHAIN/NS\_NORULE

315 : RESTATE

315 : i;}!{i=i;}ri;}m{wl;tvi=l; E;N}NrU;}S$

316 :E->i(W)M i;}!{i=i;}ri;}m{wl;tvi=l; E;N}NrU;}S$

317 : SAVESTATE: 71

317 : i;}!{i=i;}ri;}m{wl;tvi=l; i(W)M;N}NrU;}S$

318 : ;}!{i=i;}ri;}m{wl;tvi=l;w (W)M;N}NrU;}S$

319 : TNS\_NS\_NORULECHAIN/NS\_NORULE

319 : RESTATE

319 : i;}!{i=i;}ri;}m{wl;tvi=l; E;N}NrU;}S$

320 : TNS\_NS\_NORULECHAIN/NS\_NORULE

320 : RESTATE

320 : i=i;}!{i=i;}ri;}m{wl;tvi= N}NrU;}S$

321 :N-> i=i;}!{i=i;}ri;}m{wl;tvi= N}NrU;}S$

322 : SAVESTATE: 70

322 : i=i;}!{i=i;}ri;}m{wl;tvi= }NrU;}S$

323 : TNS\_NS\_NORULECHAIN/NS\_NORULE

323 : RESTATE

323 : i=i;}!{i=i;}ri;}m{wl;tvi= N}NrU;}S$

324 : TNS\_NS\_NORULECHAIN/NS\_NORULE

324 : RESTATE

324 : :{i=i;}!{i=i;}ri;}m{wl;tv JNrU;}S$

325 :J->:{N}!{N} :{i=i;}!{i=i;}ri;}m{wl;tv JNrU;}S$

326 : SAVESTATE: 69

326 : :{i=i;}!{i=i;}ri;}m{wl;tv :{N}!{N}NrU;}S$

327 : {i=i;}!{i=i;}ri;}m{wl;tvi {N}!{N}NrU;}S$

328 : i=i;}!{i=i;}ri;}m{wl;tvi= N}!{N}NrU;}S$

329 :N->i=E;N i=i;}!{i=i;}ri;}m{wl;tvi= N}!{N}NrU;}S$

330 : SAVESTATE: 70

330 : i=i;}!{i=i;}ri;}m{wl;tvi= i=E;N}!{N}NrU;}S$

331 : =i;}!{i=i;}ri;}m{wl;tvi=l =E;N}!{N}NrU;}S$

332 : i;}!{i=i;}ri;}m{wl;tvi=l; E;N}!{N}NrU;}S$

333 :E->iM i;}!{i=i;}ri;}m{wl;tvi=l; E;N}!{N}NrU;}S$

334 : SAVESTATE: 71

334 : i;}!{i=i;}ri;}m{wl;tvi=l; iM;N}!{N}NrU;}S$

335 : ;}!{i=i;}ri;}m{wl;tvi=l;w M;N}!{N}NrU;}S$

336 :M-> ;}!{i=i;}ri;}m{wl;tvi=l;w M;N}!{N}NrU;}S$

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

337 : SAVESTATE: 72

337 : ;}!{i=i;}ri;}m{wl;tvi=l;w ;N}!{N}NrU;}S$

338 : }!{i=i;}ri;}m{wl;tvi=l;wi N}!{N}NrU;}S$

339 :N-> }!{i=i;}ri;}m{wl;tvi=l;wi N}!{N}NrU;}S$

340 : SAVESTATE: 73

340 : }!{i=i;}ri;}m{wl;tvi=l;wi }!{N}NrU;}S$

341 : !{i=i;}ri;}m{wl;tvi=l;wi; !{N}NrU;}S$

342 : {i=i;}ri;}m{wl;tvi=l;wi;w {N}NrU;}S$

343 : i=i;}ri;}m{wl;tvi=l;wi;wl N}NrU;}S$

344 :N->i=E;N i=i;}ri;}m{wl;tvi=l;wi;wl N}NrU;}S$

345 : SAVESTATE: 74

345 : i=i;}ri;}m{wl;tvi=l;wi;wl i=E;N}NrU;}S$

346 : =i;}ri;}m{wl;tvi=l;wi;wl; =E;N}NrU;}S$

347 : i;}ri;}m{wl;tvi=l;wi;wl;w E;N}NrU;}S$

348 :E->iM i;}ri;}m{wl;tvi=l;wi;wl;w E;N}NrU;}S$

349 : SAVESTATE: 75

349 : i;}ri;}m{wl;tvi=l;wi;wl;w iM;N}NrU;}S$

350 : ;}ri;}m{wl;tvi=l;wi;wl;wl M;N}NrU;}S$

351 :M-> ;}ri;}m{wl;tvi=l;wi;wl;wl M;N}NrU;}S$

352 : SAVESTATE: 76

352 : ;}ri;}m{wl;tvi=l;wi;wl;wl ;N}NrU;}S$

353 : }ri;}m{wl;tvi=l;wi;wl;wl; N}NrU;}S$

354 :N-> }ri;}m{wl;tvi=l;wi;wl;wl; N}NrU;}S$

355 : SAVESTATE: 77

355 : }ri;}m{wl;tvi=l;wi;wl;wl; }NrU;}S$

356 : ri;}m{wl;tvi=l;wi;wl;wl;t NrU;}S$

357 :N-> ri;}m{wl;tvi=l;wi;wl;wl;t NrU;}S$

358 : SAVESTATE: 78

358 : ri;}m{wl;tvi=l;wi;wl;wl;t rU;}S$

359 : i;}m{wl;tvi=l;wi;wl;wl;tv U;}S$

360 :U->i i;}m{wl;tvi=l;wi;wl;wl;tv U;}S$

361 : SAVESTATE: 79

361 : i;}m{wl;tvi=l;wi;wl;wl;tv i;}S$

362 : ;}m{wl;tvi=l;wi;wl;wl;tvi ;}S$

363 : }m{wl;tvi=l;wi;wl;wl;tvi= }S$

364 : m{wl;tvi=l;wi;wl;wl;tvi=i S$

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

366 : SAVESTATE: 80

366 : m{wl;tvi=l;wi;wl;wl;tvi=i m{N}S$

367 : {wl;tvi=l;wi;wl;wl;tvi=i( {N}S$

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

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

370 : SAVESTATE: 81

370 : wl;tvi=l;wi;wl;wl;tvi=i(l wE;N}S$

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

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

373 : SAVESTATE: 82

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

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

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

376 : SAVESTATE: 83

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

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

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

379 : SAVESTATE: 84

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

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

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

382 : SAVESTATE: 85

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

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

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

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

385 : TNS\_NS\_NORULECHAIN/NS\_NORULE

385 : RESTATE

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

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

387 : SAVESTATE: 85

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

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

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

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

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

392 : SAVESTATE: 86

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

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

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

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

396 : SAVESTATE: 87

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

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

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

399 : SAVESTATE: 88

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

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

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

402 : SAVESTATE: 89

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

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

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

405 : SAVESTATE: 90

405 : wl;wl;tvi=i(l);wl;wi;wl;t wE;N}S$

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

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

408 : SAVESTATE: 91

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

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

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

411 : SAVESTATE: 92

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

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

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

414 : SAVESTATE: 93

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

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

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

417 : SAVESTATE: 94

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

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

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

420 : SAVESTATE: 95

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

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

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

423 : SAVESTATE: 96

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

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

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

426 : SAVESTATE: 97

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

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

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

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

429 : TNS\_NS\_NORULECHAIN/NS\_NORULE

429 : RESTATE

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

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

431 : SAVESTATE: 97

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

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

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

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

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

436 : SAVESTATE: 98

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

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

438 : TNS\_NS\_NORULECHAIN/NS\_NORULE

438 : RESTATE

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

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

440 : SAVESTATE: 98

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

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

442 : l);wl;wi;wl;tvi=l;tvi=l;w U);N}S$

443 :U->l l);wl;wi;wl;tvi=l;tvi=l;w U);N}S$

444 : SAVESTATE: 99

444 : l);wl;wi;wl;tvi=l;tvi=l;w l);N}S$

445 : );wl;wi;wl;tvi=l;tvi=l;wi );N}S$

446 : ;wl;wi;wl;tvi=l;tvi=l;wi; ;N}S$

447 : wl;wi;wl;tvi=l;tvi=l;wi;w N}S$

448 :N->wE;N wl;wi;wl;tvi=l;tvi=l;wi;w N}S$

449 : SAVESTATE: 100

449 : wl;wi;wl;tvi=l;tvi=l;wi;w wE;N}S$

450 : l;wi;wl;tvi=l;tvi=l;wi;wi E;N}S$

451 :E->lM l;wi;wl;tvi=l;tvi=l;wi;wi E;N}S$

452 : SAVESTATE: 101

452 : l;wi;wl;tvi=l;tvi=l;wi;wi lM;N}S$

453 : ;wi;wl;tvi=l;tvi=l;wi;wi; M;N}S$

454 :M-> ;wi;wl;tvi=l;tvi=l;wi;wi; M;N}S$

455 : SAVESTATE: 102

455 : ;wi;wl;tvi=l;tvi=l;wi;wi; ;N}S$

456 : wi;wl;tvi=l;tvi=l;wi;wi;w N}S$

457 :N->wE;N wi;wl;tvi=l;tvi=l;wi;wi;w N}S$

458 : SAVESTATE: 103

458 : wi;wl;tvi=l;tvi=l;wi;wi;w wE;N}S$

459 : i;wl;tvi=l;tvi=l;wi;wi;wl E;N}S$

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

461 : SAVESTATE: 104

461 : i;wl;tvi=l;tvi=l;wi;wi;wl iM;N}S$

462 : ;wl;tvi=l;tvi=l;wi;wi;wl; M;N}S$

463 :M-> ;wl;tvi=l;tvi=l;wi;wi;wl; M;N}S$

464 : SAVESTATE: 105

464 : ;wl;tvi=l;tvi=l;wi;wi;wl; ;N}S$

465 : wl;tvi=l;tvi=l;wi;wi;wl;t N}S$

466 :N->wE;N wl;tvi=l;tvi=l;wi;wi;wl;t N}S$

467 : SAVESTATE: 106

467 : wl;tvi=l;tvi=l;wi;wi;wl;t wE;N}S$

468 : l;tvi=l;tvi=l;wi;wi;wl;tv E;N}S$

469 :E->lM l;tvi=l;tvi=l;wi;wi;wl;tv E;N}S$

470 : SAVESTATE: 107

470 : l;tvi=l;tvi=l;wi;wi;wl;tv lM;N}S$

471 : ;tvi=l;tvi=l;wi;wi;wl;tvi M;N}S$

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

472 :M-> ;tvi=l;tvi=l;wi;wi;wl;tvi M;N}S$

473 : SAVESTATE: 108

473 : ;tvi=l;tvi=l;wi;wi;wl;tvi ;N}S$

474 : tvi=l;tvi=l;wi;wi;wl;tvi= N}S$

475 :N->tY;N tvi=l;tvi=l;wi;wi;wl;tvi= N}S$

476 : SAVESTATE: 109

476 : tvi=l;tvi=l;wi;wi;wl;tvi= tY;N}S$

477 : vi=l;tvi=l;wi;wi;wl;tvi=i Y;N}S$

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

479 : SAVESTATE: 110

479 : vi=l;tvi=l;wi;wi;wl;tvi=i vi;N}S$

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

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

482 : TNS\_NS\_NORULECHAIN/NS\_NORULE

482 : RESTATE

482 : vi=l;tvi=l;wi;wi;wl;tvi=i Y;N}S$

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

484 : SAVESTATE: 110

484 : vi=l;tvi=l;wi;wi;wl;tvi=i vi=L;N}S$

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

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

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

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

489 : SAVESTATE: 111

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

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

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

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

493 : SAVESTATE: 112

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

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

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

496 : SAVESTATE: 113

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

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

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

499 : TNS\_NS\_NORULECHAIN/NS\_NORULE

499 : RESTATE

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

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

501 : SAVESTATE: 113

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

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

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

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

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

506 : SAVESTATE: 114

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

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

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

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

510 : SAVESTATE: 115

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

511 : i;wi;wl;tvi=i(i,i,l);wi;w E;N}S$

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

513 : SAVESTATE: 116

513 : i;wi;wl;tvi=i(i,i,l);wi;w iM;N}S$

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

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

516 : SAVESTATE: 117

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

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

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

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

519 : SAVESTATE: 118

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

520 : i;wl;tvi=i(i,i,l);wi;wl;w E;N}S$

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

522 : SAVESTATE: 119

522 : i;wl;tvi=i(i,i,l);wi;wl;w iM;N}S$

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

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

525 : SAVESTATE: 120

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

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

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

528 : SAVESTATE: 121

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

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

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

531 : SAVESTATE: 122

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

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

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

534 : SAVESTATE: 123

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

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

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

537 : SAVESTATE: 124

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

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

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

540 : SAVESTATE: 125

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

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

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

543 : TNS\_NS\_NORULECHAIN/NS\_NORULE

543 : RESTATE

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

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

545 : SAVESTATE: 125

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

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

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

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

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

550 : SAVESTATE: 126

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

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

552 : TNS\_NS\_NORULECHAIN/NS\_NORULE

552 : RESTATE

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

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

554 : SAVESTATE: 126

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

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

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

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

558 : SAVESTATE: 127

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

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

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

560 : TNS\_NS\_NORULECHAIN/NS\_NORULE

560 : RESTATE

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

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

562 : SAVESTATE: 127

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

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

564 : TNS\_NS\_NORULECHAIN/NS\_NORULE

564 : RESTATE

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

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

566 : SAVESTATE: 127

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

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

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

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

570 : SAVESTATE: 128

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

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

572 : TNS\_NS\_NORULECHAIN/NS\_NORULE

572 : RESTATE

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

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

574 : SAVESTATE: 128

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

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

576 : TNS\_NS\_NORULECHAIN/NS\_NORULE

576 : RESTATE

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

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

578 : SAVESTATE: 128

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

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

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

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

582 : SAVESTATE: 129

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

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

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

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

586 :N->wE;N wi;wl;wl;tvi=i(i,i);wi;}$ N}S$

587 : SAVESTATE: 130

587 : wi;wl;wl;tvi=i(i,i);wi;}$ wE;N}S$

588 : i;wl;wl;tvi=i(i,i);wi;}$ E;N}S$

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

590 : SAVESTATE: 131

590 : i;wl;wl;tvi=i(i,i);wi;}$ iM;N}S$

591 : ;wl;wl;tvi=i(i,i);wi;}$ M;N}S$

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

593 : SAVESTATE: 132

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

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

595 :N->wE;N wl;wl;tvi=i(i,i);wi;}$ N}S$

596 : SAVESTATE: 133

596 : wl;wl;tvi=i(i,i);wi;}$ wE;N}S$

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

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

599 : SAVESTATE: 134

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

600 : ;wl;tvi=i(i,i);wi;}$ M;N}S$

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

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

602 : SAVESTATE: 135

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

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

604 :N->wE;N wl;tvi=i(i,i);wi;}$ N}S$

605 : SAVESTATE: 136

605 : wl;tvi=i(i,i);wi;}$ wE;N}S$

606 : l;tvi=i(i,i);wi;}$ E;N}S$

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

608 : SAVESTATE: 137

608 : l;tvi=i(i,i);wi;}$ lM;N}S$

609 : ;tvi=i(i,i);wi;}$ M;N}S$

610 :M-> ;tvi=i(i,i);wi;}$ M;N}S$

611 : SAVESTATE: 138

611 : ;tvi=i(i,i);wi;}$ ;N}S$

612 : tvi=i(i,i);wi;}$ N}S$

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

614 : SAVESTATE: 139

614 : tvi=i(i,i);wi;}$ tY;N}S$

615 : vi=i(i,i);wi;}$ Y;N}S$

616 :Y->vi vi=i(i,i);wi;}$ Y;N}S$

617 : SAVESTATE: 140

617 : vi=i(i,i);wi;}$ vi;N}S$

618 : i=i(i,i);wi;}$ i;N}S$

619 : =i(i,i);wi;}$ ;N}S$

620 : TNS\_NS\_NORULECHAIN/NS\_NORULE

620 : RESTATE

620 : vi=i(i,i);wi;}$ Y;N}S$

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

622 : SAVESTATE: 140

622 : vi=i(i,i);wi;}$ vi=L;N}S$

623 : i=i(i,i);wi;}$ i=L;N}S$

624 : =i(i,i);wi;}$ =L;N}S$

625 : i(i,i);wi;}$ L;N}S$

626 :L->i i(i,i);wi;}$ L;N}S$

627 : SAVESTATE: 141

627 : i(i,i);wi;}$ i;N}S$

628 : (i,i);wi;}$ ;N}S$

629 : TNS\_NS\_NORULECHAIN/NS\_NORULE

629 : RESTATE

629 : i(i,i);wi;}$ L;N}S$

630 :L->i(U) i(i,i);wi;}$ L;N}S$

631 : SAVESTATE: 141

631 : i(i,i);wi;}$ i(U);N}S$

632 : (i,i);wi;}$ (U);N}S$

633 : i,i);wi;}$ U);N}S$

634 :U->i i,i);wi;}$ U);N}S$

635 : SAVESTATE: 142

635 : i,i);wi;}$ i);N}S$

636 : ,i);wi;}$ );N}S$

637 : TNS\_NS\_NORULECHAIN/NS\_NORULE

637 : RESTATE

637 : i,i);wi;}$ U);N}S$

638 :U->iU i,i);wi;}$ U);N}S$

639 : SAVESTATE: 142

639 : i,i);wi;}$ iU);N}S$

640 : ,i);wi;}$ U);N}S$

641 : TNS\_NS\_NORULECHAIN/NS\_NORULE

641 : RESTATE

641 : i,i);wi;}$ U);N}S$

Листинг 4. Разбор исходного кода синтаксическим анализатором(продолжение)

642 :U->i,U i,i);wi;}$ U);N}S$

643 : SAVESTATE: 142

643 : i,i);wi;}$ i,U);N}S$

644 : ,i);wi;}$ ,U);N}S$

645 : i);wi;}$ U);N}S$

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

647 : SAVESTATE: 143

647 : i);wi;}$ i);N}S$

648 : );wi;}$ );N}S$

649 : ;wi;}$ ;N}S$

650 : wi;}$ N}S$

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

652 : SAVESTATE: 144

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

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

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

655 : SAVESTATE: 145

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

656 : ;}$ M;N}S$

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

658 : SAVESTATE: 146

658 : ;}$ ;N}S$

659 : }$ N}S$

660 :N-> }$ N}S$

661 : SAVESTATE: 147

661 : }$ }S$

662 : $ S$

663 :S-> $ S$

664 : SAVESTATE: 148

664 : $ $

665 :

666 : LENTA\_END

667 : ------>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]);

}

break;

case LEX\_LEFTHESIS:

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

break;

case LEX\_RIGHTHESIS:

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

{

result.push(stk.top());

stk.pop();

}

stk.pop();

if (function)

{

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

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

function = false;

}

break;

case LEX\_COMMA:

if (function)

quantityParm++;

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

{

result.push(stk.top());

stk.pop();

}

break;

case LEX\_MORE:

case LEX\_LESS:

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

break;

}

}

while (!stk.empty())

{

result.push(stk.top());

stk.pop();

}

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

{

if (!result.empty())

{

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

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

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

result.pop();

}

else

{

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

}

}

return true;

}

# **Приложение Д**

.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\_str: proc

EXTRN write\_int: proc

EXTRN write\_str : proc

EXTRN copy\_str: proc

.stack 4096

.const ; сегмент констант - литералы

L0 sdword 100

L1 sdword 5

L2 sdword 1

L3 sdword 0

L4 byte "Это проект с названием KAD-2022", 0

L5 byte "POIT\_5", 0

L6 byte " ", 0

L7 byte "Вывод всех чисел, которые делятся на 5 нацело (в диапазоне от 0 до 100):", 0

L8 sdword 4

L9 byte "Количество чисел, которые делятся на 5: ", 0

L10 byte "Hello", 0

L11 byte "How are you?", 0

L12 byte "Первых три элемента, скопированных из первой строки во вторую :", 0

L13 sdword 3

L14 byte "Длина наибольшей строки из двух: ", 0

.data ; сегмент данных - переменные и параметры

i\_DivisionByFive sdword 0

countOfNumber\_DivisionByFive sdword 0

result\_repeat0 sdword 0

lenghtOfFirstStroke\_findMaxStroke sdword 0

lenghtOfSecondStroke\_findMaxStroke sdword 0

result\_findMaxStroke sdword 0

group\_main dword ?

count\_main sdword 0

strokeOne\_main dword ?

strokeSecond\_main dword ?

ns\_main dword ?

maxL\_main sdword 0

.code ; сегмент кода

;----------- DivisionByFive ------------

DivisionByFive PROC, count\_DivisionByFive : sdword

; --- сохранить регистры ---

push ebx

push edx

; ----------------------

push count\_DivisionByFive

pop i\_DivisionByFive

cyclenext0:

mov edx, i\_DivisionByFive

cmp edx, L0

jg cycle0

push i\_DivisionByFive

push L1

pop ebx

pop eax

cdq

idiv ebx

push edx

pop result\_repeat0

push result\_repeat0

push L2

pop ebx

pop eax

sub eax, ebx

push eax

pop result\_repeat0

mov edx, result\_repeat0

cmp edx, L3

jl true1

jmp next1

true1:

push i\_DivisionByFive

call write\_int

push countOfNumber\_DivisionByFive

push L2

pop ebx

pop eax

add eax, ebx

push eax

pop countOfNumber\_DivisionByFive

next1:

push i\_DivisionByFive

push L2

pop ebx

pop eax

add eax, ebx

push eax

pop i\_DivisionByFive

jmp cyclenext0

cycle0:

; --- восстановить регистры ---

pop edx

pop ebx

; -------------------------

mov eax, countOfNumber\_DivisionByFive

ret

DivisionByFive ENDP

;------------------------------

;----------- findMaxStroke ------------

findMaxStroke PROC, strOne\_findMaxStroke : dword, strTwo\_findMaxStroke : dword

; --- сохранить регистры ---

push ebx

push edx

; ----------------------

push strOne\_findMaxStroke

call lenght\_str

push eax

pop lenghtOfFirstStroke\_findMaxStroke

push strTwo\_findMaxStroke

call lenght\_str

push eax

pop lenghtOfSecondStroke\_findMaxStroke

mov edx, lenghtOfFirstStroke\_findMaxStroke

cmp edx, lenghtOfSecondStroke\_findMaxStroke

jg true2

jl false2

true2:

push lenghtOfFirstStroke\_findMaxStroke

pop result\_findMaxStroke

jmp next2

false2:

push lenghtOfSecondStroke\_findMaxStroke

pop result\_findMaxStroke

next2:

; --- восстановить регистры ---

pop edx

pop ebx

; -------------------------

mov eax, result\_findMaxStroke

ret

findMaxStroke ENDP

;------------------------------

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

main PROC

push offset L4

call write\_str

mov group\_main, offset L5

push group\_main

call write\_str

push offset L6

call write\_str

push offset L7

call write\_str

push L8

call DivisionByFive

push eax

pop count\_main

push offset L9

call write\_str

push count\_main

call write\_int

push offset L6

call write\_str

mov strokeOne\_main, offset L10

mov strokeSecond\_main, offset L11

push strokeOne\_main

call write\_str

push strokeSecond\_main

call write\_str

push offset L12

call write\_str

push L13

push strokeSecond\_main

push strokeOne\_main

call copy\_str

mov ns\_main, eax

push ns\_main

call write\_str

push offset L6

call write\_str

push offset L14

call write\_str

push strokeSecond\_main

push strokeOne\_main

call findMaxStroke

push eax

pop maxL\_main

push maxL\_main

call write\_int

push 0

call ExitProcess

main ENDP

end main