МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«**ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**»**

|  |  |
| --- | --- |
| Институт (факультет) | Институт Информационных Технологий |
| Кафедра | Математического и программного обеспечения ЭВМ |

**КУРСОВАЯ РАБОТА**

по дисциплине «Модуль: Информатика»

|  |  |
| --- | --- |
| на тему | «Разработка алгоритмического обеспечения и построение лексического |
| анализатора компилятора» | |

|  |
| --- |
| Выполнил студент группы |
| 1ПИб-01-1оп-21 |
| направление подготовки (специальности) |
| 09.03.04, Программная инженерия |
| *шифр, наименование* |
| Трофимов Никита Владиславович |
| *фамилия, имя, отчество* |

|  |
| --- |
| Руководители |
| Ганичева Оксана Георгиевна  Пышницкий Константин Михайлович |
| *фамилия, имя, отчество* |
| Доцент  Старший преподаватель |
| *должность* |

|  |
| --- |
| Дата представления работы |
| « » 20 г. |
| Заключение о допуске к защите |
|  |
|  |
| Оценка , |
| *количество баллов* |
| Подпись преподавателя |

Череповец, 2023 год

ОГЛАВЛЕНИЕ

[Введение 3](#_bookmark0)

[Изучение и описание предметной области 4](#_bookmark1)

* 1. [Лексический анализ 4](#_bookmark2)
  2. [Синтаксический анализ 18](#_bookmark3)

[Основная часть 25](#_bookmark4)

* 1. [Постановка задачи 25](#_bookmark5)
  2. [Описание класса автоматных грамматик в классификации Хомского 26](#_bookmark6)
  3. [Связь лексического анализа с автоматными грамматиками и конечными](#_bookmark7) [автоматами 29](#_bookmark7)
  4. [Описание конечного автомата, лежащего в основе лексического](#_bookmark8) [анализатора 32](#_bookmark8)
  5. [Построение грамматики по конечному автомату 35](#_bookmark9)
  6. [Построение КС-грамматики 59](#_bookmark10)

1. [Логическое проектирование 64](#_bookmark11)
   1. [Блок-схемы 64](#_bookmark12)
   2. [Словесное описание работы алгоритма на примере разбора входных](#_bookmark13) [последовательностей 68](#_bookmark13)
      1. [Словесное описание алгоритма нахождения комментариев и лишних](#_bookmark14) [символов в тексте 68](#_bookmark14)

[4.3. Определение сложности полученного алгоритма 69](#_bookmark15)

[Физическое проектирование 70](#_bookmark16)

* 1. [Выбор структур данных 70](#_bookmark17)
  2. [Спецификация функций 70](#_bookmark18)

[Проектирование интерфейса 71](#_bookmark19)

[Тестирование программного обеспечения 75](#_bookmark20)

[Заключение 79](#_bookmark21)

[Список литературы 80](#_bookmark22)

[Приложение 1 Техническое задание 81](#_bookmark23)

[Приложение 2 Текст программы 88](#_bookmark24)

[Приложение 3 Руководство пользователя 126](#_bookmark25)

[Приложение 4 Граф конечного автомата 128](#_bookmark26)

Введение

Компиляторы составляют существенную часть программного обеспечения ЭВМ. Это связано с тем, что языки высокого уровня стали основным средством разработки и реализации программ. Лексический и синтаксический анализы, являются этапами разбора программы при компиляции. Оптимизированное и безошибочное проведение анализа является необходимым условием для успешного проведения дальнейших этапов компиляции.

Теория построения трансляторов в значительной части посвящена методам выявления структуры входного текста и использования этой структуры для преобразования входа с сохранением его смысла. Эти методы базируются на идеях, высказанных Наомом Хомским в середине ХХ века, с которых начиналась структурная лингвистика.

Согласно этой идее, смысл текста существенно связан с его структурой. Двусмысленные тексты имеют несколько структур, недвусмысленные – одну структуру. И для двусмысленных и односмысленных предложений языка их смысл определяется структурой предложений. В современных трансляторах искусственных языков эти идеи и понятия являются базовыми.

В ходе выполнения курсовой работы требуется написать код для программы, в результате, которая выполняет анализ входного текста и порождает таблицу лексем с указанием их типов и значений. Работа лексического анализатора обязательно моделируется конечным автоматом. Также в ходе анализа входного текста программа должна выводить прохождение по состояниям соответствующего автомата, сообщения об ошибках с указанием строки и символа, дескрипторный код и псевдокод. Главной задачей построения лексического анализатора является выполнение всех требований к курсовой работе.

Изучение и описание предметной области

* 1. Лексический анализ

Компилятор — это специальная программа, которая переводит текст программы, написанный на языке программирования, в набор машинных кодов. Компиляция — сборка программы, включающая трансляцию всех модулей программы, написанных на одном или нескольких исходных языках программирования высокого уровня и/или языке ассемблера, в эквивалентные программные модули на низкоуровневом языке, близком машинному коду или непосредственно на машинном языке или ином двоичнокодовом низкоуровневом командном языке и последующую сборку исполняемой

машинной программы.

Концептуально компилятор работает пофазно. В процессе каждой фазы про- исходит преобразование исходной программы из одного представления в другое. Типичное разбиение компилятора на фазы показано на рис. 1.

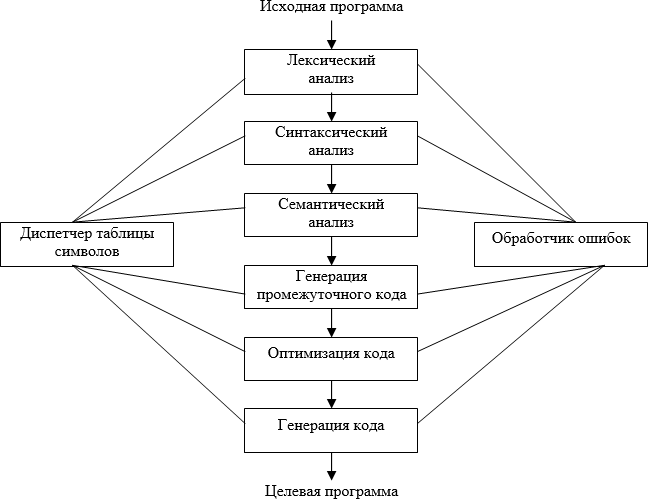


Рис. 1. Фазы компилятора

Процесс компиляции состоит из следующих этапов:

* + 1. Лексический анализ. Последовательность символов исходного файла преобразуется в последовательность лексем.
    2. Синтаксический анализ. Последовательность лексем преобразуется в дерево разбора.
    3. Семантический анализ. Дерево разбора обрабатывается с целью установления его семантики (смысла) — например, привязка идентификаторов к их декларациям, типам, проверка совместимости, определение типов выражений и т. д.
    4. Оптимизация. Выполняется удаление излишних конструкций и упрощение кода с сохранением его смысла.
    5. Генерация кода. Из промежуточного представления порождается объектный код.

Результатом компиляции является объектный код.

Объектный код — это программа на языке машинных кодов с частичным сохранением символьной информации, необходимой в процессе сборки.

При отладочной сборке возможно сохранение большого количества символьной информации (идентификаторов переменных, функций, а также типов).

Более подробная схема работы компилятора представлена на рисунке (Рис.2)

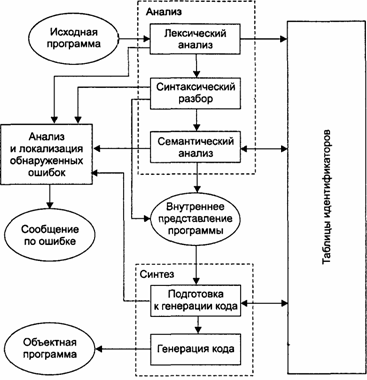


Рис.2 Работа компилятора

Первый этап разбора исходного текста программы, осуществляемого компилятором, - это лексический анализ. Лексический анализатор считывает последовательно все слова (токены, лексемы) в тексте программы, преобразуя их в конструкции, которые затем уже используются для дальнейшего разбора текста. Делается это для того, чтобы различить в последующем различные идентификаторы и непосредственные конструкции самого языка (такие, как зарезервированные слова).

Следом за лексическим анализом может быть препроцессор. Основная задача препроцессора - это замена одних лексем другими, которые были заранее определены в тексте программы. Используется препроцессор также для условной компиляции (т.е. когда часть кода должна быть откомпилирована только при выполнении определённых условий - для определённой платформы, только при отладочном билде и т.д. и т.п.), для выполнения определённых макросов и некоторых других подобных вещей. Препроцессор не является обязательной частью компилятора, поскольку многие языки программирования

не нуждаются в нём.

Следующий этап - это синтаксический анализ, или парсинг. Этот этап компиляции выполняется синтаксическим анализатором, или парсером, и является, пожалуй, самым важным и, если можно так сказать, ответственным этапом компиляции. Компилятор рассматривает все токены, и, в зависимости от их значения и положения в тексте программы, формирует так называемое дерево разбора. То есть программа, бывшая до этого в недрах компилятора просто линейным набором символов, становится деревом, элементы которого расположены в соответствии с грамматикой того языка программирования, для которого написан данный конкретный компилятор.

Следом за синтаксическим анализом следует этап анализа семантического. Если синтаксический анализатор строил скелет нашей программы, то семантический помогает этому скелету обрасти плотью. Программа наполняется смыслом: переменные становятся переменными, объекты - объектами, а баги - багами. На самом деле, никакого волшебства не происходит

- просто дерево разбора, терпеливо построенное парсером, дополняется семантической информацией о значении идентификаторов. Кстати, на этом этапе возникают и многие ошибки компиляции - например, такие, как несоответствие типов. Хотя, конечно, на парсинг тоже приходится изрядное количество ошибок, без которых, к сожалению, текст свеженаписанной программы обходится крайне редко даже у очень опытных программистов.

Дальше пути различных компиляторов расходятся. В большинстве компиляторов следом за этапом семантического анализа идёт перевод программы в некоторый промежуточный код, который может использоваться для генерации кода под разные аппаратные платформы. Если компилятор выполняет компиляцию только для какой-то одной аппаратной платформы, то программа может транслироваться в коды на языке Ассемблера соответствующей процессорной архитектуры, или, если компилятор трудится для какой-то виртуальной машины, то переводиться программа может затем в специальный байт-код, понятный соответствующей виртуальной машине. Тем

не менее, в большинстве современных компиляторов нет непосредственной трансляции в ассемблерный код - даже если в итоге компилятор не должен стараться для создания кросс-платформенных программ, всё равно, сначала идёт трансляция программы в какой-то промежуточный код, а только потом уже в исполняемый. Причина этого в оптимизации кода.

Лексический анализ – процесс аналитического разбора входной последовательности символов на распознанные группы — лексемы — с целью получения на выходе идентифицированных последовательностей (подобно группировке букв в словах).

Лексический анализ представляет собой первую фазу компиляции. Его основная задача состоит в чтении новых символов и выдачи последовательности лексем, используемых синтаксическим анализатором в своей работе. Лексической единицей языка является лексема.

Лексема — это структурная единица языка, которая состоит из элементарных символов языка и не содержит в своем составе других структурных единиц языка.

Лексемами языков естественного языка общения между людьми являются слова. Лексемами языков программирования являются идентификаторы, константы, ключевые слова языка, знаки операций и т.п. Состав возможных лексем каждого конкретного языка программирования определяется синтаксисом этого языка.

При лексическом анализе символы исходной программы считываются и группируются в поток лексем, в котором каждая лексема представляет логически связанную последовательность символов — идентификатор, ключевое слово (if, while и т.п.), символ пунктуации или многосимвольный оператор типа :=. Последовательность символов, формирующая слово образует лексему.

Некоторые лексемы дополняются "лексическим значением". Например, когда найден идентификатор, лексический анализатор не только генерирует лексему, но и вносит лексему в таблицу символов, если ее там нет. Лексическое

значение, связанное с этим появлением лексемы, указывает на запись в таблице символов.

Основная задача лексического анализа - разбить входной текст, состоящий из последовательности одиночных символов, на последовательность слов, или лексем, т.е. выделить эти слова из непрерывной последовательности символов. Все символы входной последовательности с этой точки зрения разделяются на символы, принадлежащие каким-либо лексемам, и символы, разделяющие лексемы (разделители). В некоторых случаях между лексемами может и не быть разделителей.

Обычно все лексемы делятся на классы. Примерами таких классов являются числа (целые, восьмеричные, шестнадцатеричные, действительные и т.д.), идентификаторы, строки. Отдельно выделяются ключевые слова и символы пунктуации (иногда их называют символы-ограничители). Как правило, ключевые слова - это некоторое конечное подмножество идентификаторов.

Каждый класс лексем описывается правилом, называемым шаблоном. Шаблон соответствует каждой строке в наборе-классе лексем. Лексема же представляет собой последовательность символов исходной программы, которая соответствует шаблону.

Шаблон – правило, описывающее набор лексем, которые могут представлять определенную лексему в исходной программе.

С точки зрения дальнейших фаз анализа лексический анализатор выдает информацию двух сортов: для синтаксического анализатора, работающего вслед за лексическим, существенна информация о последовательности классов лексем, ограничителей и ключевых слов, а для контекстного анализа, работающего вслед за синтаксическим, важна информация о конкретных значениях отдельных лексем (идентификаторов, чисел и т.д.).

Таким образом, общая схема работы лексического анализатора такова. Сначала выделяется отдельная лексема (возможно, используя символы- разделители). Ключевые слова распознаются либо явным выделением непосредственно из текста, либо сначала выделяется идентификатор, а затем

делается проверка на принадлежность его множеству ключевых слов.

Если выделенная лексема является ограничителем, то он (точнее, некоторый его признак) выдается как результат лексического анализа. Если выделенная лексема является ключевым словом, то выдается признак соответствующего ключевого слова. Если выделенная лексема является идентификатором - выдается признак идентификатора, а сам идентификатор сохраняется отдельно. Наконец, если выделенная лексема принадлежит какому-либо из других классов лексем (например, лексема представляет собой число, строку и т.д.), то выдается признак соответствующего класса, а значение лексемы сохраняется отдельно.

Работа лексического анализатора задается некоторым конечным автоматом. Однако, непосредственное описание конечного автомата неудобно с практической точки зрения. Поэтому для задания лексического анализатора, как правило, используется либо регулярное выражение, либо праволинейная грамматика. Все три формализма (конечных автоматов, регулярных выражений и праволинейных грамматик) имеют одинаковую выразительную мощность. В частности, по регулярному выражению или праволинейной грамматике можно сконструировать конечный автомат, распознающий тот же язык.

Лексема (лексическая единица языка) — это структурная единица языка, которая состоит из элементарных символов языка и не содержит в своем составе других структурных единиц языка. Лексемами языков естественного общения являются слова.

Лексемами языков программирования являются идентификаторы, константы, ключевые слова языка, знаки операций и разделители. Состав возможных лексем каждого конкретного языка программирования определяется синтаксисом этого языка.

Лексический анализатор (или сканер) — это часть компилятора, которая читает исходную программу и выделяет в ее тексте лексемы входного языка. На вход лексического анализатора поступает текст исходной программы, а выходная информация передается для дальнейшей обработки компилятором на

этапе синтаксического анализа и разбора.

С теоретической точки зрения лексический анализатор не является обязательной частью компилятора. Все его функции могут выполняться на этапе синтаксического разбора. Однако лексический анализ включают в состав практически всех компиляторов по следующим причинам:

* применение лексического анализатора упрощает работу с текстом исходной программы на этапе синтаксического разбора и сокращает объем обрабатываемой информации;
* для выделения в тексте и разбора лексем применяется простая и эффективная техника анализа, в то время как на этапе синтаксического анализа конструкций исходного языка используются достаточно сложные алгоритмы разбора;
* при конструкции компилятора, когда лексический анализ реализован отдельно от синтаксического, для перехода от одной версии языка программирования к другой достаточно только перестроить относительно простой лексический анализатор.

Результатом работы лексического анализатора является перечень всех обнаруженных в тексте исходной программы лексем. Этот перечень лексем представляется в виде таблицы, называемой таблицей лексем. Каждой лексеме в таблице лексем соответствует некий уникальный условный номер, зависящий от типа лексемы, а также дескрипторный и псевдокод.

Кроме того, информация о некоторых типах лексем, найденных в исходной программе, должна помещаться в таблицу идентификаторов (или в одну из таблиц идентификаторов, если компилятор предусматривает различные таблицы идентификаторов для различных типов лексем).

Кроме того, каждая лексема (идентификатор или константа) может встречаться в таблице идентификаторов только один раз. Также можно отметить, что лексемы в таблице лексем обязательно располагаются в том же порядке, что и в исходной программе (порядок лексем в ней не меняется), а в таблице идентификаторов лексемы располагаются в любом порядке так, чтобы

обеспечить удобство поиска.

Роль лексического анализатора:

1. Выделение из текста входной программы всех лексем, входящих конструкции функций обработки строк языка C++;
2. Обработка входной программы: удаление комментариев и лишних пробелов;
3. Распределение распознанных лексем;
4. Выявление ошибок в написании лексем, в случае обнаружения таковой – выдать сообщение, содержащее ошибочную лексему и номер строки.

Функции, выполняемые лексическим анализатором, и состав лексем, которые он выделяет в тексте исходной программы, могут меняться в зависимости от версии компилятора. Какие функций должен выполнять лексический анализатор, и какие типы лексем он должен выделять во входной программе решают разработчики компилятора.

Поскольку лексический анализатор является частью компилятора, который считывает исходный текст, он может так же выполнять некоторые второстепенные задачи. К ним, например, относятся удаление из текста исходной программы комментариев и не несущих смысловой нагрузки пробелов (а так же символов табуляции и новой строки). Еще одна задача состоит в согласовании сообщений об ошибках компиляции и текста исходной программы. Например, лексический анализатор может подсчитывать количество считанных строк и указать строку, вызвавшую ошибку. В некоторых компиляторах лексический анализатор создает копию текста исходной программы с указанием ошибок. Если исходный язык поддерживает макросы и директивы процессора, то они так же могут реализовываться лексическим анализатором.

Основные функции лексического анализатора:

1. упрощается работа с текстом исходной программы на этапе синтаксического разбора и сокращается объём обрабатываемой информации, так как лексический анализатор структурирует поступающий на вход исходный

текст программы и удаляет всю незначащую информацию;

1. Для выделения в тексте и разбора лексем можно применять простую, эффективную и теоретически хорошо проработанную технику анализа, в то время как на этапе синтаксического анализа конструкций исходного языка используются достаточно сложные алгоритмы разбора;
2. Сканер отделяет сложный по конструкции синтаксический анализатор от работы непосредственно с текстом исходной программы, структура которого может варьироваться в зависимости от версии входного языка – при такой конструкции компилятора при переходе от одной версии языка к другой достаточно только перестроить относительно простой сканер.

Функции, выполняемые лексическим анализатором, и состав типов лексем, которые он выделяет в тексте исходной программы, могут меняться в зависимости от реализации компилятора. То, какие функции должен выполнять лексический анализатор, а какие оставлять для этапа синтаксического разбора, решают разработчики компилятора.

В основном лексические анализаторы выполняют удаление из текста исходной программы комментариев и незначащих символов, а также выделение лексем следующих типов: идентификаторов, строковых, символьных и числовых констант, ключевых слов входного языка, знаков операций и разделителей.

В простейшем случае фазы лексического и синтаксического анализа могут выполняться компилятором последовательно. Но для многих языков программирования на этапе лексического анализа может быть недостаточно информации для однозначного определения типа и границ очередной лексемы.

Поэтому в большинстве компиляторов лексический и синтаксический анализаторы - это взаимосвязанные части. Возможны два метода организации взаимосвязи лексического анализа и синтаксического разбора:

* последовательный;
* параллельный.

При последовательном варианте лексический анализатор просматривает

весь текст исходной программы от начала до конца один раз и преобразует его в структурированный набор данных. Этот набор данных называют также таблицей лексем. В таблице лексем ключевые слова языка, идентификаторы и константы, как правило, заменятся на специально оговоренные коды, им соответствующие (конкретная кодировка определяется разработчиком при реализации компилятора). Таблица лексем строится полностью вся сразу, и больше к ней компилятор не возвращается. Всю дальнейшую обработку выполняют следующие фазы компиляции.

При параллельном варианте лексический анализ исходного текста выполняется поэтапно так, что синтаксический анализатор, выполнив разбор очередной конструкции языка, обращается к сканеру за следующей лексемой. При этом он может сообщить информацию о том, какую лексему следует ожидать. В процессе разбора при возникновении ошибки может происходить

«откат назад», чтобы попытаться выполнить анализ текста на другой основе. И только, после того, как синтаксический анализатор успешно выполнит разбор очередной конструкции языка, лексический анализатор помещает найденные лексемы в таблицу лексем и таблицу идентификаторов и продолжает разбор дальше в том же порядке.

Работа синтаксического и лексического анализаторов при их параллельном взаимодействия изображена на рисунке 3.

Идентификаторы

Лексический

анализатор

Очередная

лексема

Обращение за

лексемой

Синтаксический

анализатор

Исходная программа

Таблица

идентификаторов

Рис. 3. Параллельное взаимодействие лексического и синтаксического анализаторов

Причины, по которым фаза анализа компиляции разделяется на лексический и синтаксический анализы:

1. Упрощение разработки. Отделение лексического анализатора от синтаксического часто позволяет упростить одну из фаз анализа. Например, включение в синтаксический анализатор комментарием и пробелов существенно сложнее, чем удаление их лексическим анализатором.
2. Увеличивается эффективность компилятора. Отдельный лексический анализатор позволяет создать специализированный и потенциально более эффективный процессор для решения поставленной задачи.
3. Увеличивается переносимость компилятора. Особенности входного алфавита и другие специфичные характеристики используемых устройств могут ограничивать возможности лексического анализатора.

Общие принципы построения лексических анализаторов

Лексический анализатор имеет дело с такими объектами, как различного рода константы и идентификаторы (к последним относятся и ключевые слова). Язык констант и идентификаторов в большинстве случаев является регулярным

— то есть может быть описан с помощью регулярных грамматик. Распознавателями для регулярных языков являются конечные автоматы.

Существуют правила, с помощью которых для любой регулярной грамматики может быть построен недетерминированный конечный автомат, распознающий цепочки языка, заданного этой грамматикой. Конечный автомат для каждой входной цепочки языка дает ответ на вопрос о том, принадлежит или нет цепочка языку, заданному автоматом.

В общем случае задача сканера несколько шире, чем просто проверка цепочки символов лексемы на соответствие ее входному языку. Кроме этого, сканер должен выполнить следующие действия:

1. Четко определить границы лексемы, которые в исходном тексте явно не заданы;
2. Выполнить действия для сохранения информации об обнаруженной лексеме (или выдать сообщение об ошибке, если лексема неверна).

Определение границ лексем

Выделение границ лексем представляет определенную проблему. Ведь во входном тексте программы лексемы не ограничены никакими специальными символами. Определение границ лексем – это выделение тех строк в общем потоке входных символов, для которых надо выполнять распознавание. B общем случае эта задача может быть сложной и тогда требуется параллельная работа лексического анализатора, синтаксического разбора и, возможно, - семантического анализа. Для большинства входных языков границы лексем распознаются по заданным терминальным символам. Эти символы – пробелы, знаки операций, символы комментариев, а также разделители (запятые, точки с запятой и т.п.) Набор таких терминальных символов может варьироваться в зависимости от синтаксиса входного языка.

Как правило, лексические анализаторы действуют по следующему принципу:

1. очередной символ из входного потока данных добавляется в лексему всегда, когда он может быть туда добавлен;
2. как только символ не может быть добавлен в лексему, то считается, что он является границей лексемы и началом следующей лексемы.

При этом от пользователя требуется явно указать с помощью пробелов (или других незначащих символов) границы лексем. Такой подход возможен для большинства входных языков.

Выполнение действий, связанных с лексемами

Основная задача лексического анализатора – обнаружение лексемы во входном потоке и ее распознавание, т.е. определение класса к которому она относится.

В основе распознавателя лексем лежит конечный автомат, который должен иметь не только входной язык, но и выходной. Он должен не только уметь распознать правильную лексему на входе, но и породить связанную с ней последовательность символов на выходе. В такой конфигурации КА преобразуется в конечный преобразователь.

Таким образом, кроме порождения цепочки символов выходного языка, лексический анализатор должен, уметь выполнять такие действия, как запись найденной лексемы в таблицу лексем, поиск ее таблице символов и запись новой лексемы в таблицу символов. Набор действий определяется реализацией компилятора. Обычно эти действия выполняются сразу же по обнаружению конца распознаваемой лексемы.

Практическая реализация лексических анализаторов

В принципе компилятор может иметь в своем составе не один, а несколько лексических анализаторов, каждый из которых предназначен для выборки и проверки определённого типа лексем.

Таким образом, обобщенный алгоритм работы простейшего лексического анализатора в компиляторе можно описать следующим образом:

* из входного потока выбирается очередной символ, в зависимости от которого запускается тот или иной лексический анализатор (символ может быть также проигнорирован либо признан ошибочным);
* запущенный лексический анализатор просматривает входной поток символов программы на исходном языке, выделяя символы, входящие в

требуемую лексему, до обнаружения очередного символа, который может ограничивать лексему, либо до обнаружения ошибочного символа;

* при успешном распознавании информация, о выделенной лексеме заносится в таблицу лексем и таблицу идентификаторов, алгоритм возвращается к первому этапу и продолжает рассматривать входной поток символов с того места, на котором остановился лексический анализатор;

при неуспешном распознавании выдается сообщение об ошибке, а дальнейшие действия зависят от реализации лексического анализатора — либо его выполнение прекращается, либо делается попытка распознать следующую лексему (идет возврат к первому этапу алгоритма).

В целом техника построения лексических анализаторов основывается на моделировании работы детерминированных и недетерминированных КА с дополнением функций распознавателя вызовами функций обработки ошибок, а также заполнения таблиц символов и таблиц идентификаторов. Такая техника не требует сложной математической обработки и принципиально важных преобразований входных грамматик. Для разработчиков лексических анализаторов важно только решить, где кончаются функции лексического анализа и начинаются функции синтаксического разбора. После этого процесс построения лексического анализатора легко поддается автоматизации.

* 1. Синтаксический анализ

Синтаксический анализ - это процесс, который определяет, принадлежит ли некоторая последовательность лексем языку, порождаемому грамматикой. В принципе, по любой грамматике можно построить синтаксический анализатор, но грамматики, используемые на практике, имеют специальную форму.

Например, известно, что для любой контекстно-свободной грамматики может быть построен анализатор, сложность которого не превышает O(n3) для входной строки длины n, но в большинстве случаев по заданному языку программирования мы можем построить такую грамматику,

которая позволит сконструировать и более быстрый анализатор.

Синтаксический анализатор (синтаксический разбор) - это часть компилятора, которая отвечает за выявление основных синтаксических конструкций входного языка. В задачу синтаксического анализа входит: найти и выделить основные синтаксические конструкции в тексте входной программы, установить тип и проверить правильность каждой синтаксической конструкции, наконец, представить синтаксические конструкции в виде, удобном для дальнейшей генерации текста результирующей программы.

В основе синтаксического анализатора лежит распознаватель текста входной программы на основе грамматики входного языка. Как правило, синтаксические конструкции языков программирования могут быть описаны с помощью КС-грамматик, реже встречаются языки, которые могут быть описаны с помощью регулярных грамматик. Чаще всего регулярные грамматики применимы к языкам ассемблера, а языки высокого уровня построены на основе синтаксиса КС-языков.

Распознаватель дает ответ на вопрос о том, принадлежит или нет цепочка входных символов заданному языку – это основная задача синтаксического анализатора. Кроме этого, синтаксический анализатор должен иметь некий выходной язык, с помощью которого он передает следующим фазам компиляции всю информацию о найденных и разобранных синтаксических структурах.

Синтаксический разбор — это основная часть компилятора на этапе анализа. Без выполнения синтаксического разбора работа компилятора бессмысленна, в то время как лексический разбор в принципе является необязательной фазой. Все задачи по проверке синтаксиса входного языка могут быть решены на этапе синтаксического разбора. Лексический анализатор только позволяет избавить сложный по структуре синтаксический анализатор от решения примитивных задач по выявлению и запоминанию лексем входной программы.

Выходом лексического анализатора является таблица лексем (или цепочка

лексем). Эта таблица образует вход синтаксического анализатора, который исследует только один компонент каждой лексемы — ее тип. Остальная информация о лексемах используется на более поздних фазах компиляции при семантическом анализе, подготовке к генерации и генерации кода результирующей программы. Синтаксический анализ (или разбор) — это процесс, в котором исследуется таблица лексем и устанавливается, удовлетворяет ли она структурным условиям, явно сформулированным в определении синтаксиса языка.

Синтаксический анализатор воспринимает выход лексического анализатора и разбирает его в соответствии с грамматикой входного языка. Однако в грамматике входного языка программирования обычно не уточняется, какие конструкции следует считать лексемами. Примерами конструкций, которые обычно распознаются во время лексического анализа, служат ключевые слова, константы и идентификаторы. Но эти же конструкции могут распознаваться и синтаксическим анализатором. На практике не существует жесткого правила, определяющего, какие конструкции должны распознаваться на лексическом уровне, а какие надо оставлять синтаксическому анализатору. Обычно это определяет разработчик компилятора исходя из технологических аспектов программирования, а также синтаксиса и семантики входного языка.

Основу любого синтаксического анализатора всегда составляет распознаватель, построенный на основе какого-либо класса КС-грамматик. Поэтому главную роль в том, как функционирует синтаксический анализатор и какой алгоритм лежит в его основе, играют принципы построения распознавателей КС-языков. Без применения этих принципов невозможно выполнить эффективный синтаксический разбор предложений входного языка. Каждый язык программирования имеет правила, которые предписывают синтаксическую структуру корректных программ. В Pascal, например, программа состоит из блоков, блок - из инструкций, инструкции - из выражений, выражения - из лексем и т.д. Синтаксис конструкций языка программирования может быть описан с помощью контекстно-свободных

грамматик или нотации БНФ (форма Бэкуса Наура). Грамматики обеспечивают значительные преимущества разработчикам языков программирования и создателям компиляторов.

Грамматика дает точную и при этом простую для понимания синтаксическую спецификацию языка программирования.

Для некоторых классов грамматик можно автоматически построить эффективный синтаксический анализатор, который определяет, корректна ли структура исходной программы. Дополнительным преимуществом автоматического создания анализатора является возможность обнаружения синтаксических неоднозначностей и других сложных для распознавания конструкций языка, которые иначе могли бы остаться не замеченными на начальных фазах создания языка и его компилятора.

Правильно построенная грамматика придает языку программирования структуру, которая способствует облегчению трансляции исходной программы в объектный код, выявлению ошибок. Для преобразования описаний трансляции, основанных на грамматике языка, в рабочие программы имеется соответствующий программный инструментарий.

Со временем языки эволюционируют, обогащаясь новыми конструкциями, и выполняют новые задачи. Добавление конструкций в язык окажется более простой задачей, если существующая реализация языка основана на его грамматическом описании.

В модели компилятора синтаксический анализатор получает строку лексем с выхода лексического анализатора, как показано на рис. 21, и проверяет, может ли эта строка порождаться грамматикой исходного языка. Он также сообщает обо всех выявленных ошибках. Кроме того, он должен уметь обрабатывать обычно часто встречающиеся ошибки и продолжать работу с оставшейся частью программы (рис.4).

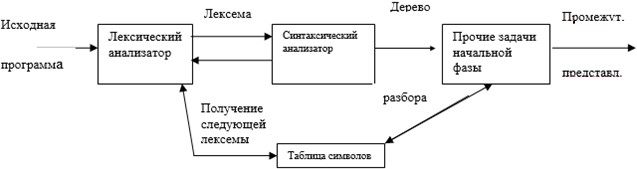


Рис. 4. Обработка лексемы

Имеется три основных типа синтаксических анализаторов грамматик:

1. Универсальные методы разбора, такие как алгоритмы Кока-Янгера- Касами или Эрли, могут работать с любой грамматикой. Однако эти методы слишком неэффективны для использования в промышленных компиляторах.
2. Нисходящие (сверху вниз) методы синтаксического анализа. Нисходящие синтаксические анализаторы строят дерево разбора сверху (от корня) вниз (к листьям). Входной поток синтаксического анализатора сканируется посимвольно слева направо.
3. Восходящие (снизу вверх) методы синтаксического анализа. Восходящие методы начинают построение дерева разбора с листьев и идут к корню. Входной поток также сканируется посимвольно слева направо.

Восходящие и нисходящие методы синтаксического анализа наиболее распространены в компиляторах.

Наиболее эффективные нисходящие и восходящие методы работают только с подклассами грамматик, однако некоторые из этих подклассов, такие как LL- и LR-грамматики, достаточно выразительны для описания большинства синтаксических конструкций языков программирования. Реализованные вручную синтаксические анализаторы чаще работают с LL-грамматиками. Синтаксические анализаторы для несколько большего класса LR-грамматик обычно создаются с помощью автоматизированных инструментов.

Выходом синтаксического анализатора является некоторое представление дерева разбора входного потока лексем, выданного лексическим анализатором. На практике имеется множество задач, которые могут сопровождать процесс разбора, - например, сбор информации о различных лексемах в таблице

символов, исполнение проверки типов и других видов семантического анализа, а также создание промежуточного кода.

Обработка синтаксических ошибок

Если компилятор будет иметь дело исключительно с корректными программами, его разработка и реализация существенно упрощаются. Однако программисты пишут программы с ошибками, и хороший компилятор должен помочь программисту обнаружить их и локализовать. Большинство специфика- ций языков программирования не определяет реакции компилятора на ошибки - этот вопрос отдается на откуп разработчикам компилятора. Однако планиро- вание системы обработки ошибок с самого начала работы над компилятором может, как упростить его структуру, так и улучшить его реакцию на ошибки.

Любая программа потенциально содержит множество ошибок самого разного уровня. Ошибки могут быть:

* лексическими, такими как неверно записанные идентификаторы, ключевые слова или операторы;
* синтаксическими, например, арифметические выражения с несбалансированными скобками;
* семантическими, такими как операторы, применяемые к несовместимым с ними операндам;
* логическими, например, бесконечная рекурсия.

Основные действия по выявлению ошибок и восстановлению после них решаются на этапе синтаксического анализа. Одна из причин этого состоит в том, что многие ошибки по природе своей являются синтаксическими или проявляются, когда поток лексем, идущий от лексического анализатора, нарушает определяющие язык программирования грамматические правила. Вторая причина заключается в точности современных методов разбора - они очень эффективно выявляют синтаксические ошибки в программе. Определение присутствия в программе семантических или логических ошибок

* задача более сложная.

Обработчик ошибок синтаксического анализатора имеет очень просто

формулируемые цели:

* + он должен ясно и точно сообщать о наличии ошибок;
  + он должен обеспечивать быстрое восстановление после ошибки, чтобы продолжить поиск последующих ошибок.

Основная часть

* 1. Постановка задачи

Вариант задания курсовой работы – «Операция оператора *if* и условия ?: языка С++».

В курсовой работе нужно создать программу, моделирующую работу лексического анализатора компилятора. Моделирование лексического анализатора базируется на конечном автомате, построенном в соответствии с требованиями компилятора языка С++ и вариантом к курсовой работе.

Код анализируемой программы:

#include <iostream> using namespace std; void main()

{

cout << "Battle of the random digits!" << endl<<endl; int a;

int b; int c;

a = rand();

b = rand();

c = rand(); a = a + 100; b = b - 50;

c = c / 2;

cout << "Round 1. A versus B. Fight!" << endl;

if (a > b) {

cout << " A - winner!" << endl<<endl;

cout << "Round 2. A versus C. Final fight!" << endl << endl; if (a > c)

}

else {

cout << "A - winner winner chicken dinner!" << endl << endl; cout << "Round 3. Loosers fight! B versus C." << endl << endl;

b > c ? cout << "B - the best of the bad" << endl : cout << "C - the best of the bad" << endl;

cout << "B - winner!" << endl << endl;

cout << "Round 2. B versus C. Final fight!" << endl; if (b > c)

cout << "B - winner winner chicken dinner!" << endl << endl; cout << "Round 3. Loosers fight! A versus C." << endl << endl;

a > c ? cout << "A - the best of the bad" << endl : cout << "C - the best of the bad" << endl;

}

system("pause");

}

Алгоритм программы:

Инициализируются 3 переменные, которым присваиваются 3 случайных числа. Далее сравнивается первое число со вторым, если первое побеждает, то на экран выводится победитель, затем сравнивается первое с третьим и если снова первое больше, то выводится сообщение с его победой, иначе сравнивается второе число с третьим и если второе больше оно зонимает второе место, а если нет, то второе место занимает третье число. Если же изначально второе больше первого, то выводится сообщение об этом и далее сравнивается уже второе с третьим, если второе больше, то оно считается победителем и далее сравнивается первое с третьим и выявляется какое из них больше.

Результат работы программы представлен на рис.5.

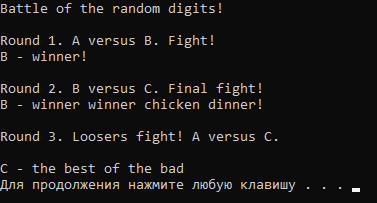


Рис.5. Результат работы программы

* 1. Описание класса автоматных грамматик в классификации Хомского Иерархия Хомского — классификация формальных языков и формальных

грамматик, согласно которой они делятся на 4 типа по их условной сложности. Предложена профессором Массачусетского технологического института, лингвистом Ноамом Хомским.

Формальные грамматики классифицируются по структуре их правил. Если все без исключения правила грамматики удовлетворяют некоторой заданной структуре, то ее относят к определенному типу. В грамматике достаточно иметь одно правило, не удовлетворяющее требованию структуры, и она уже не попадает в заданный тип.

По классификации Хомского выделяют четыре типа грамматик.

Тип 0: грамматики с фразовой структурой. Это самый общий тип грамматик. В этот тип попадают все без исключения формальные грамматики, но часть из них может быть также отнесена и к другим классификационным типам. Грамматики, которые относятся только к типу 0 и не могут быть отнесены к другим типам и являются самыми сложными по структуре. Грамматики, относящиеся только к типу 0, не имеют практического применения.

Тип 1: контекстно-зависимые и неукорачивающие грамматики. Структура правил КЗ-грамматик такова, что при построении предложений заданного ими языка один и тот же нетерминальный символ может быть заменен на ту или иную цепочку символов в зависимости от контекста, в котором он встречается. Именно поэтому эти грамматики называются контекстно-зависимыми. Неукорачивающие грамматики имеют такую структуру правил, что при построении предложений языка, заданного грамматикой, любая цепочка символов может быть заменена на цепочку символов не меньшей длины. Отсюда и название «неукорачивающие». Доказано, что эти два класса грамматик эквивалентны. При построении компиляторов такие грамматики не применяются, поскольку языки программирования, рассматриваемые компиляторами, имеют более простую структуру и могут быть построены с помощью грамматик других типов.

Тип 2: контекстно-свободные грамматики. В правой части у них стоит всегда хотя бы один символ. Их отличие от предыдущих типов состоит в том, что левая часть правил должна состоять ровно из одного нетерминального символа. Такие грамматики также иногда называют неукорачивающими

контекстно-свободными (НКС) грамматиками (потому что в правой части правил этих грамматик всегда должен стоять как минимум один символ).

Тип 3: регулярные грамматики. В правой части правил грамматик этого типа может присутствовать не более одного нетерминального символа, причём он должен быть расположен во всех правилах одной грамматики с одной и той же стороны от цепочки терминалов, а требования к левой части правил совпадают с предыдущим типом. К типу регулярных относятся два эквивалентных класса грамматик: леволинейные и праволинейные. Эти два класса грамматик эквивалентны и относятся к типу регулярных грамматик. Леволинейные грамматики G =< T, N, P, S >; V = N 𝖴 T могут иметь правила двух видов: A → Bγ или A → γ, где A, B∈N; γ∈VT\*. Праволинейные грамматики G = < T, N, P, S >; V = N 𝖴 T могут иметь правила тоже двух видов: A → γB или A → γ, где A, B∈N; γ∈ V T\*. Эти два класса грамматик эквивалентны и относятся к типу регулярных грамматик. Регулярные грамматики используются при описании простейших конструкций языков программирования: идентификаторов, констант, строк, комментариев и т.д. Эти грамматики исключительно просты и удобны в использовании, поэтому в компиляторах на их основе строятся функции лексического анализа входного языка. Среди регулярных грамматик можно выделить отдельный класс - автоматные грамматики. Они могут быть леволинейными и праволинейными. Автоматные грамматики отличаются тем, что в их правилах вместо цепочки терминальных символов присутствует только один терминальный символ. Очевидно, что любая автоматная грамматика является регулярной, обратное же справедливо не всегда.

Типы грамматик соотносятся между собой особым образом. Из определения типов 2 и 3 видно, что любая регулярная грамматика является КС- грамматикой, а не наоборот. Также очевидно, что любая грамматика может быть отнесена и к типу 0, поскольку он не накладывает никаких ограничений на правила. В то же время существуют укорачивающие КС-грамматики (тип 2), которые не являются ни контекстно-зависимыми, ни неукорачивающими (тип

1. В целом сложность грамматики обратно пропорциональна тому максимально возможному номеру типа, к которому может быть отнесена грамматика. Грамматики, которые относятся только к типу 0, являются самыми сложными, а грамматики, которые можно отнести к типу 3, – самыми простыми.
   1. Связь лексического анализа с автоматными грамматиками и конечными автоматами

Лексический анализ (ЛА) является первой фазой трансляции. Термин

«лексика» в обычном языке подразумевает правила составления слов из букв. В языке программирования ЛА соответствует фаза трансляции, в которой из последовательности отдельных символов, букв языка выделяются слова (лексемы). Типичными словами в языке программирования являются такие компоненты как комментарии, идентификаторы, константы, служебные слова, знаки операций. Лексика имеет ряд принципиальных ограничений, которая делает возможной ее распознавание описанными ниже методами:

* + лексемы распознаются независимо друг от друга. Единственное, что возможно, это «перекрытие» процесса распознавания двух рядом расположенных лексем на заданное количество символов (которые участвуют в завершении распознавания текущей лексемы и начале распознавания следующей);
  + определение лексем включает элементы выбора одного из нескольких вариантов продолжения и повторения;
  + вложенность одной лексемы в другую или наличие произвольной вложенности элементов лексемы не допускается.

ЛА является наиболее простой и формализованной фазой трансляции. Любой алгоритм ЛА базируется на последовательном просмотре текста, с возвратом и перечитыванием из входной последовательности не фиксированного числа символов, поэтому программу ЛА иногда называют сканером. Формальной основой описания процесса ЛА являются конечные

автоматы.

Разработка лексического анализатора – сканера, выполняется достаточно просто, если воспользоваться хорошо разработанным математическим аппаратом – теорией регулярных языков и конечных автоматов. В рамках этой теории классы однотипных лексем (идентификаторы, ключевые слова, константы и т.д.) рассматриваются как формальные языки – язык идентификаторов, констант, ключевых слов и т.д. Причём эти языки настолько просты, что они порождаются простейшей из грамматик – регулярной грамматикой – формальная грамматика типа 3 по иерархии Хомского. Построенная регулярная грамматика является источником, по которому в дальнейшем конструируется вычислительное устройство, реализующее функцию распознавания предложений языка, порождаемого данной грамматикой. Для регулярных языков таким устройством является конечный автомат.

Математической моделью процесса распознавания регулярного языка является вычислительное устройство, которое называется конечным автоматом (КА). Термин «конечный» подчеркивает то, что вычислительное ус- тройство имеет фиксированный и конечный объем памяти и обрабатывает пос- ледовательность входных символов, принадлежащих некоторому конечному множеству. Существуют различные типы КА, если функцией выхода КА (ре- зультатом работы) является лишьуказание на то, допустима или нет входная последовательность символов, такой КА называют конечным распознавателем. Именно этот тип КА в дальнейшем мы и будем рассматривать.

Конечный автомат является принципиально более простой формальной моделью, нежели компьютерная программа (машина Тьюринга). Программа включает в себя две компоненты: алгоритм и произвольным образом организованные неограниченного объема данные. Конечный автомат не содержит данных и представляет собой алгоритмическую часть программы в чистом виде. Т. е. он представляет собой систему, которая реагирует только на изменение состояния внешней среды, но не способна к запоминанию. Она не

имеет изменяемой памяти, сама управляющая информация о структуре автомата хранится в постоянной памяти . Единственным хранимым параметром автомата является его текущее состояние, которое характеризует текущий шаг выполняемого алгоритма.

Отсутствие собственной памяти задает принципиально другую «модель поведения», аналог которой имеется в животном мире:

* + конечный автомат моделирует «инстинктивное» поведение, основанное на воспроизведении программы действий, меняющейся в зависимости от последовательности внешних событий, но не запоминающей информации из внешней среды;
  + компьютерная программа моделирует «рефлекторное» поведение, принципиально связанное с запоминанием и обработкой информации о внешних событиях и настройкой (адаптацией) к ним.

Работа такой системы выглядит так:

* + КА является дискретной системой, работающей по тактам (шагам), в каждом из которых он получает на вход один из символов входного алфавита;
  + шаг работы КА состоит в переходе из текущего состояния в новое состояние при получении на вход очередного символа входной цепочки. Этот переход определяется функцией переходов;
  + результат работы КА заключается в записи в выходную цепочку символов, генерируемых функцией выходов КА. Т. е. параллельно с переходом в новое состояние формируется выходной символ, определяемый парой

«текущее состояние – входной символ». Выходной символ может быть и пустым;

Работа КА заключается в преобразовании входной цепочки символов в выходную. Закон этого преобразования определяет поведение КА. Задача проектирования КА состоит в создании КА, обеспечивающем заданные правила преобразования цепочек. Очевидно, что отсутствие у КА внутренней памяти ограничивает его возможности по преобразованию цепочек (общепринятый термин - моделирующая способность). С другой стороны, эти же ограничения

позволяют решать в общем случае многие проблемы (термин – алгоритмическая разрешимость, т. е. принципиальная возможность разработать соответствующий алгоритм).

Хотя автоматы и можно представить в традиционной для алгоритма технологии блок-схем (управляющие КА так и строятся на основе разметки блок-схемы алгоритма), наиболее удобной формой их представления для человека является графическая - диаграмма состояний-переходов, а для программирования и формальных преобразований – табличная.

Конечным автоматом является определённое устройство, которое обладает конечным числом состояний и предназначено для прочтения слов заданного конечного алфавита. По умолчанию считается, что слово занесено на некоторую ленту, составленную из ячеек, в каждую из которых занесена одна буква алфавита. Чтение ленты выполняется в дискретные такты времени в направлении слева направо. Подразумевается, что считывание произвольного слова «a» автомат должен начинать из специально заданного начального состояния. Считывание очередного символа в текущем такте времени должно сопровождаться перемещением вправо к прочтению следующего символа и изменением текущего состояния, которое должно определяться считанной в текущем такте буквой и состоянием, в котором автомат находится в текущем такте. Слово, имеющее длину l, автомат обрабатывает в течение l тактов. Итог прочтения слова «а» должен определяться состоянием, в котором автомат может оказаться после завершения обработки данного слов.

* 1. Описание конечного автомата, лежащего в основе лексического анализатора

Конечным автоматом называется обобщенная диаграмма переходов между состояниями. Он задается следующей пятеркой:

𝐴 = (𝑋, 𝑆, δ, S0, F),

где X = {a1, a2, …, an} – конечное множество символов (входной алфавит);

S = {S0, S1, …,Sn-1} – конечное непустое множество состояний (алфавит

состояний);

δ: S×X →S – функция переходов;

S0 ∈ S – начальное состояние конечного автомата;

F ⊆ S – множество выделенных заключительных состояний.

НКА может использоваться в виде помеченного ориентированного графа, так называемого графа переходов, узлы которого представляют собой состояния, а помеченные дуги составляют функцию переходов.

Функция переходов НКА может быть реализована различными способами. Простейший из них — таблица переходов, в которой строки представляют состояния, а столбцы — входные символы. Запись в строке i для символа, а является множеством состояний, которые могут быть достигнуты переходом из состояния i при входном символе а.

Определения пятерки конечного автомата по варианту задания:

X = {A…Z, a…z, 0…9, \_, (, ), {, }, [, ], **,**,**.**, **;**, **:**, \*, %, ~,^,/, **.** , **,** , +, -, |, =, >, <,

!, “, &, #,?,$};

S = {S0, S1, S2… S20, S21, S22…S40, S41, S42… S60, S61, S62…S80, S81, S82…S100, S101, S102…S140, S141, S142 … S160, S162, S163, … S180, S181, S182 … S201, S202,S203};

δ = (

|  |  |  |
| --- | --- | --- |
| (S0, <)→S24 | (S0, :)→S22 | (S0, t)→S138 |
| (S0, >)→S24 | (S0, [)→S22 | (S0, u)→S149 |
| (S0, +)→S24 | (S0, ])→S22 | (S0, v)→S128 |
| (S0, **-**)→S24 | (S0, ;)→S22 | (S0, w)→S123 |
| (S0, ?)→S24 | (S0, цифра)→S1 | (S0, x)→S120 |
| (S0, /)→S24 | (S0, буква)→S2 | (S0, N)→S114 |
| (S0, &)→S24 | (S0, a)→S30 | (S1, /t)→S0 |
| (S0, !)→S24 | (S0, b)→S38 | (S1, .)→S27 |
| (S0, ~)→S24 | (S0, c)→S11 | (S1, e)→S28 |
| (S0, ^)→S24 | (S0, d)→S18 | (S1, ,)→S27 |
| (S0, %)→S24 | (S0, e)→S71 | (S1, цифра)→S0 |
| (S0, |)→S24 | (S0, f)→S3 | (S2, -)→S2 |
| (S0, \*)→S24 | (S0, g)→S84 | (S2, .)→S2 |
| (S0, =)→S24 | (S0, i)→S88 | (S2, ~)→S2 |
| (S0, ")→S23 | (S0, l)→S97 | (S2, $)→S2 |
| (S0, #)→S22 | (S0, n)→S101 | (S2, цифра)→S2 |
| (S0, })→S22 | (S0, o)→S118 | (S2, буква)→S2 |
| (S0, {)→S22 | (S0, p)→S185 | (S3, a)→S4 |
| (S0, ()→S22 | (S0, r)→S179 | (S3, l)→S8 |
| (S0, ))→S22 | (S0, s)→S158 | (S3, o)→S82 |

|  |  |  |
| --- | --- | --- |
| (S4, l)→S5 | (S25, \*)→S25 | (S75, u)→S76 |
| (S5, s)→S6 | (S25, &)→S25 | (S75, d)→S78 |
| (S6, e)→S7 | (S25, |)→S25 | (S76, m)→S77 |
| (S8, o)→S9 | (S26, :)→S26 | (S78, l)→S79 |
| (S9, a)→S10 | (S27, цифра)→S1 | (S79, e)→S80 |
| (S10, t)→S81 | (S28, +)→S29 | (S82, r)→S83 |
| (S11, a)→S47 | (S28, -)→S29 | (S84, o)→S85 |
| (S11, h)→S46 | (S29, цифра)→S1 | (S85, t)→S86 |
| (S11, l)→S55 | (S30, n)→S31 | (S86, o)→S87 |
| (S11, o)→S12 | (S30, s)→S33 | (S88, f)→S89 |
| (S12, u)→S13 | (S30, u)→S35 | (S88, n)→S90 |
| (S12, n)→S15 | (S31, d)→S32 | (S90, t)→S91 |
| (S13, t)→S14 | (S33, m)→S34 | (S90, c)→S92 |
| (S15, s)→S16 | (S35, t)→S36 | (S92, l)→S93 |
| (S16, t)→S17 | (S36, o)→S37 | (S93, u)→S94 |
| (S18, e)→S19 | (S38, o)→S39 | (S94, d)→S95 |
| (S18, o)→S66 | (S38, r)→S42 | (S95, e)→S96 |
| (S19, f)→S20 | (S39, o)→S40 | (S97, o)→S98 |
| (S20, a)→S21 | (S40, l)→S41 | (S98, n)→S99 |
| (S21, u)→S59 | (S42, e)→S43 | (S99, g)→S100 |
| (S22, #)→S22 | (S43, a)→S44 | (S101, a)→S102 |
| (S22, :)→S26 | (S44, k)→S45 | (S101, e)→S110 |
| (S22, })→S22 | (S46, a)→S53 | (S101, o)→S112 |
| (S22, {)→S22 | (S47, s)→S48 | (S102, m)→S103 |
| (S22, ()→S22 | (S47, t)→S50 | (S103, e)→S104 |
| (S22, ))→S22 | (S48, e)→S49 | (S104, s)→S105 |
| (S22, ])→S22 | (S50, c)→S51 | (S105, p)→S106 |
| (S22, [)→S22 | (S51, h)→S52 | (S106, a)→S107 |
| (S23, ")→S23 | (S53, r)→S54 | (S107, c)→S108 |
| (S24, ~)→S24 | (S55, a)→S56 | (S108, e)→S109 |
| (S24, ^)→S24 | (S56, s)→S57 | (S110, w)→S111 |
| (S24, %)→S24 | (S57, s)→S58 | (S112, t)→S113 |
| (S24, <)→S25 | (S59, l)→S60 | (S114, U)→S115 |
| (S24, >)→S25 | (S60, t)→S61 | (S115, L)→S116 |
| (S24, +)→S25 | (S62, e)→S63 | (S116, L)→S117 |
| (S24, -)→S25 | (S63, t)→S64 | (S118, r)→S119 |
| (S24, =)→S25 | (S64, e)→S65 | (S120, o)→S121 |
| (S24, &)→S25 | (S66, u)→S67 | (S121, r)→S122 |
| (S24, \*)→S25 | (S67, b)→S68 | (S123, h)→S124 |
| (S24, |)→S25 | (S68, l)→S69 | (S124, i)→S125 |
| (S25, <)→S25 | (S69, e)→S70 | (S125, l)→S126 |
| (S25, >)→S25 | (S71, l)→S72 | (S126, e)→S127 |
| (S25, +)→S25 | (S71,n)→S75 | (S128, o)→S129 |
| (S25, -)→S25 | (S72, s)→S73 | (S128, i)→S132 |
| (S25, =)→S25 | (S73, e)→S74 | (S129, i)→S130 |

|  |  |  |
| --- | --- | --- |
| (S130, d)→S131 | (S158, w)→S169 | (S186, b)→S187 |
| (S132, r)→S133 | (S158, t)→S159 | (S187, l)→S188 |
| (S133, t)→S134 | (S158, h)→S175 | (S188, i)→S189 |
| (S134, u)→S135 | (S159, d)→S160 | (S189, c)→S190 |
| (S135, a)→S136 | (S159, a)→S161 | (S191, i)→S192 |
| (S136, l)→S137 | (S159, r)→S165 | (S191, o)→S197 |
| (S138, h)→S139 | (S161, t)→S162 | (S192, v)→S193 |
| (S138, r)→S145 | (S162, i)→S163 | (S193, a)→S194 |
| (S139, r)→S142 | (S163, c)→S164 | (S194, t)→S195 |
| (S139, i)→S140 | (S165, u)→S166 | (S195, e)→S196 |
| (S140, s)→S141 | (S166, c)→S167 | (S197, t)→S198 |
| (S142, o)→S143 | (S167, c)→S168 | (S198, e)→S199 |
| (S143, w)→S144 | (S169, i)→S170 | (S199, c)→S200 |
| (S145, u)→S147 | (S170, t)→S171 | (S200, t)→S201 |
| (S145, y)→S146 | (S171, c)→S172 | (S201, e)→S202 |
| (S147, e)→S148  (S149, s)→S150 (S149, n)→S154 | (S172, h)→S173  (S175, o)→S176 (S176, r)→S177 | (S202, d)→S203 |
| (S150, i)→S151 (S151, n)→S152 (S152, g)→S153 (S154, i)→S155  (S155, o)→S156 (S156, n)→S157 | (S179, e)→S180 (S180, t)→S181 (S181, u)→S182 (S182, r)→S183  (S185, u)→S186 (S185, r)→S191 |  |

F = {S1, S2, S7, S14, S17, S22, S23, S24, S25, S26, S32, S34, S37, S41 ,S45, S49, S54, S58 S61, S65, S66, S70, S74, S77, S80, S81, S83, S89, S91, S96, S100, S109, S113, S117, S119, S122, S127, S131, S137, S146, S148, S153, S157, S160, S164, S168, S173, S178, S184, S190, S196, S203}.

Граф конечного автомата представлен в приложении 4.

* 1. Построение грамматики по конечному автомату

Алгоритм построения регулярной грамматики G = (T, N, P, S) по автомату A

= (X, S, δ, S0, F):

1) T=X={ A…Z, a…z, 0…9, \_, (, ), {, }, [, ], **,**,**.**, **;**, **:**, \*, %, ~,^,/, **.** , **,** , +, -, |, =, >,

<, !, “, &, #,?,$};

2) N = S⋃F = { S0, S1, S2… S20, S21, S22…S40, S41, S42… S60, S61, S62…S80, S81, S82…S100, S101, S102…S140, S141, S142 … S160, S162, S163, … S180, S181, S182 … S201, S202,S203};

3) S0= {S};

4) F={ S1, S2, S7, S14, S17, S22, S23, S24, S25, S26, S32, S34, S37, S41 ,S45, S49, S54, S58 S61, S65, S66, S70, S74, S77, S80, S81, S83, S89, S91, S96, S100, S109, S113, S117, S119, S122, S127, S131, S137, S146, S148, S153, S157, S160, S164, S168, S173, S178, S184, S190, S196, S203}

1. Отображение δ строится по правилам:
   * каждому правилу подстановки в грамматике G вида А –> аВ ставится в соответствие команда (А,а) –> В;
   * каждому правилу подстановки в грамматике G вида А –> а ставится в соответствие команда (А,а) –> Z.

Регулярная грамматика по конечному автомату представлена в таблице 1.

Таблица 1

|  |  |
| --- | --- |
| Конечный автомат | Регулярная грамматика |
| 1 | 2 |
| (S0, <)→S24 | S0→<S24 |
| (S0, >)→S24 | S0→>S24 |
| (S0, +)→S24 | S0→+S24 |
| (S0, **-**)→S24 | S0→-S24 |
| (S0, ?)→S24 | S0→?S24 |
| (S0, /)→S24 | S0→/S24 |
| (S0, &)→S24 | S0→&S24 |
| (S0, !)→S24 | S0→!S24 |
| (S0, ~)→S24 | S0→~S19 |
| (S0, ^)→S24 | S0→^S24 |
| (S0, %)→S24 | S0→%S24 |
| (S0, |)→S24 | S0→|S24 |
| (S0, \*)→S24 | S0→\*S24 |
| (S0, =)→S24 | S0→=S24 |
| (S0, ")→S23 | S0→"S23 |
| (S0, #)→S22 | S0→#S22 |
| (S0, })→S22 | S0→}S22 |
| (S0, {)→S22 | S0→{S22 |

Продолжение таблицы 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S0, ()→S22 | S0→(S22 |
| (S0, ))→S22 | S0→)S22 |
| (S0, :)→S22 | S0→:S22 |
| (S0, [)→S22 | S0→[S22 |
| (S0, ])→S22 | S0→]S22 |
| (S0, ;)→S22 | S0→;S22 |
| (S0, цифра)→S1 | S0→цифраS1 |
| (S0, буква)→S2 | S0→букваS2 |
| (S0, a)→S30 | S0→aS30 |
| (S0, b)→S38 | S0→bS38 |
| (S0, c)→S11 | S0→cS11 |
| (S0, d)→S18 | S0→dS18 |
| (S0, e)→S71 | S0→eS71 |
| (S0, f)→S3 | S0→fS3 |
| (S0, g)→S84 | S0→gS84 |
| (S0, i)→S88 | S0→iS88 |
| (S0, l)→S97 | S0→lS97 |
| (S0, n)→S101 | S0→nS101 |
| (S0, o)→S118 | S0→oS118 |
| (S0, p)→S185 | S0→pS185 |
| (S0, r)→S179 | S0→rS179 |
| (S0, s)→S158 | S0→sS158 |
| (S0, t)→S138 | S0→tS138 |
| (S0, u)→S149 | S0→uS149 |
| (S0, v)→S128 | S0→vS128 |
| (S0, w)→S123 | S0→2S123 |
| (S0, x)→S120 | S0→xS120 |
| (S0, N)→S114 | S0→NS114 |
| (S1, /t)→S0 | S1→/tS0 |
| (S1, .)→S27 | S1→.S27 |
| (S1, e)→S28 | S1→eS28 |
| (S1, ,)→S27 | S1→,S27 |
| (S1, цифра)→S0 | S1→цифраS0 |
| (S2, -)→S2 | S2→-S2 |
| (S2, .)→S2 | S2→.S2 |
| (S2, ~)→S2 | S2→~S2 |
| (S2, $)→S2 | S2→$S2 |
| (S2, цифра)→S2 | S2→цифраS2 |
| (S2, буква)→S2 | S2→букваS2 |
| (S3, a)→S4 | S3→aS4 |
| (S3, l)→S8 | S3→lS8 |
| (S3, o)→S82 | S3→oS82 |
| (S4, l)→S5 | S4→lS5 |
| (S5, s)→S6 | S5→sS6 |
| (S6, e)→S7 | S6→eS7 |
| (S8, o)→S9 | S8→oS9 |
| (S9, a)→S10 | S9→aS10 |

Продолжение таблицы 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S10, t)→S81 | S10→tS81 |
| (S11, a)→S47 | S11→aS47 |
| (S11, h)→S46 | S11→hS46 |
| (S11, l)→S55 | S11→lS55 |
| (S11, o)→S12 | S11→oS12 |
| (S12, u)→S13 | S12→uS13 |
| (S12, n)→S15 | S12→nS15 |
| (S13, t)→S14 | S13→tS14 |
| (S15, s)→S16 | S15→sS16 |
| (S16, t)→S17 | S16→tS17 |
| (S18, e)→S19 | S18→eS19 |
| (S18, o)→S66 | S18→oS66 |
| (S19, f)→S20 | S19→fS20 |
| (S20, a)→S21 | S20→aS21 |
| (S21, u)→S59 | S21→uS59 |
| (S22, #)→S22 | S22→#S22 |
| (S22, :)→S26 | S22→:S26 |
| (S22, })→S22 | S22→}S22 |
| (S22, {)→S22 | S22→{S22 |
| (S22, ()→S22 | S22→(S22 |
| (S22, ))→S22 | S22→)S22 |
| (S22, ])→S22 | S22→]S22 |
| (S22, [)→S22 | S22→[S22 |
| (S23, ")→S23 | S23→"S23 |
| (S24, ~)→S24 | S24→~S24 |
| (S24, ^)→S24 | S24→^S24 |
| (S24, %)→S24 | S24→%S24 |
| (S24, <)→S25 | S24→<S25 |
| (S24, >)→S25 | S24→>S25 |
| (S24, +)→S25 | S24→+S25 |
| (S24, -)→S25 | S24→-S25 |
| (S24, =)→S25 | S24→=S25 |
| (S24, &)→S25 | S24→&S25 |
| (S24, \*)→S25 | S24→\*S25 |
| (S24, |)→S25 | S24→|S25 |
| (S25, <)→S25 | S25→<S25 |
| (S25, >)→S25 | S25→>S25 |
| (S25, +)→S25 | S25→+S25 |
| (S25, -)→S25 | S25→-S25 |
| (S25, =)→S25 | S25→=S25 |
| (S25, \*)→S25 | S25→\*S25 |
| (S25, &)→S25 | S25→&S25 |
| (S25, |)→S25 | S25→|S25 |
| (S26, :)→S26 | S26→:S26 |
| (S27, цифра)→S1 | S27→цифраS1 |
| (S28, +)→S29 | S28→+S29 |
| (S28, -)→S29 | S28→-S29 |

Продолжение таблицы 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S29, цифра)→S1 | S29→цифраS1 |
| (S30, n)→S31 | S30→nS31 |
| (S30, s)→S33 | S30→sS33 |
| (S30, u)→S35 | S30→uS35 |
| (S31, d)→S32 | S31→dS32 |
| (S33, m)→S34 | S33→mS34 |
| (S35, t)→S36 | S35→tS36 |
| (S36, o)→S37 | S36→oS37 |
| (S38, o)→S39 | S38→oS39 |
| (S38, r)→S42 | S38→rS42 |
| (S39, o)→S40 | S39→oS40 |
| (S40, l)→S41 | S40→lS41 |
| (S42, e)→S43 | S42→eS43 |
| (S43, a)→S44 | S43→aS44 |
| (S44, k)→S45 | S44→kS45 |
| (S46, a)→S53 | S46→aS53 |
| (S47, s)→S48 | S47→sS48 |
| (S47, t)→S50 | S47→tS50 |
| (S48, e)→S49 | S48→eS49 |
| (S50, c)→S51 | S50→cS51 |
| (S51, h)→S52 | S51→hS52 |
| (S53, r)→S54 | S53→rS54 |
| (S55, a)→S56 | S55→aS56 |
| (S56, s)→S57 | S56→sS57 |
| (S57, s)→S58 | S57→sS58 |
| (S59, l)→S60 | S59→lS60 |
| (S60, t)→S61 | S60→tS61 |
| (S62, e)→S63 | S62→eS63 |
| (S63, t)→S64 | S63→tS64 |
| (S64, e)→S65 | S64→eS65 |
| (S66, u)→S67 | S66→uS67 |
| (S67, b)→S68 | S67→bS68 |
| (S68, l)→S69 | S68→lS69 |
| (S69, e)→S70 | S69→eS70 |
| (S71, l)→S72 | S71→lS72 |
| (S71,n)→S75 | S71→nS75 |
| (S72, s)→S73 | S72→sS73 |
| (S73, e)→S74 | S73→eS74 |
| (S75, u)→S76 | S75→uS76 |
| (S75, d)→S78 | S75→dS78 |
| (S76, m)→S77 | S76→mS77 |
| (S78, l)→S79 | S78→lS79 |
| (S79, e)→S80 | S79→eS80 |
| (S82, r)→S83 | S82→rS83 |
| (S84, o)→S85 | S84→oS85 |
| (S85, t)→S86 | S85→tS86 |
| (S86, o)→S87 | S86→oS87 |

Продолжение таблицы 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S88, f)→S89 | S88→fS89 |
| (S88, n)→S90 | S88→nS90 |
| (S90, t)→S91 | S90→tS91 |
| (S90, c)→S92 | S90→cS92 |
| (S92, l)→S93 | S92→lS93 |
| (S93, u)→S94 | S93→uS94 |
| (S94, d)→S95 | S94→dS95 |
| (S95, e)→S96 | S95→eS96 |
| (S97, o)→S98 | S97→oS98 |
| (S98, n)→S99 | S98→nS99 |
| (S99, g)→S100 | S99→gS100 |
| (S101, a)→S102 | S101→aS102 |
| (S101, e)→S110 | S101→eS110 |
| (S101, o)→S112 | S101→oS112 |
| (S102, m)→S103 | S102→mS103 |
| (S103, e)→S104 | S103→eS104 |
| (S104, s)→S105 | S104→sS105 |
| (S105, p)→S106 | S105→pS106 |
| (S106, a)→S107 | S106→aS107 |
| (S107, c)→S108 | S107→cS108 |
| (S108, e)→S109 | S108→eS109 |
| (S110, w)→S111 | S110→wS111 |
| (S112, t)→S113 | S112→tS113 |
| (S114, U)→S115 | S114→US115 |
| (S115, L)→S116 | S115→LS116 |
| (S116, L)→S117 | S116→LS117 |
| (S118, r)→S119 | S118→rS119 |
| (S120, o)→S121 | S120→oS121 |
| (S121, r)→S122 | S121→rS122 |
| (S123, h)→S124 | S123→hS124 |
| (S124, i)→S125 | S124→iS125 |
| (S125, l)→S126 | S125→lS126 |
| (S126, e)→S127 | S126→eS127 |
| (S128, o)→S129 | S128→oS129 |
| (S128, i)→S132 | S128→iS132 |
| (S129, i)→S130 | S129→iS130 |
| (S130, d)→S131 | S130→dS131 |
| (S132, r)→S133 | S132→rS133 |
| (S133, t)→S134 | S133→tS134 |
| (S134, u)→S135 | S134→uS135 |
| (S135, a)→S136 | S135→aS136 |
| (S136, l)→S137 | S136→lS137 |
| (S138, h)→S139 | S138→hS139 |
| (S138, r)→S145 | S138→rS145 |
| (S139, r)→S142 | S139→rS142 |
| (S139, i)→S140 | S139→iS140 |
| (S140, s)→S141 | S140→sS141 |

Продолжение таблицы 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S142, o)→S143 | S142→oS143 |
| (S143, w)→S144 | S143→wS144 |
| (S145, u)→S147 | S145→uS147 |
| (S145, y)→S146 | S145→yS146 |
| (S147, e)→S148 | S147→eS148 |
| (S149, s)→S150 | S149→sS150 |
| (S149, n)→S154 | S149→nS154 |
| (S150, i)→S151 | S150→iS151 |
| (S151, n)→S152 | S151→nS152 |
| (S152, g)→S153 | S152→gS153 |
| (S154, i)→S155 | S154→iS155 |
| (S155, o)→S156 | S155→oS156 |
| (S156, n)→S157 | S156→nS157 |
| (S158, w)→S169 | S158→wS169 |
| (S158, t)→S159 | S158→tS159 |
| (S158, h)→S175 | S158→hS175 |
| (S159, d)→S160 | S159→dS160 |
| (S159, a)→S161 | S159→aS161 |
| (S159, r)→S165 | S159→rS165 |
| (S161, t)→S162 | S161→tS162 |
| (S162, i)→S163 | S162→iS163 |
| (S163, c)→S164 | S163→cS164 |
| (S165, u)→S166 | S165→uS166 |
| (S166, c)→S167 | S166→cS167 |
| (S167, c)→S168 | S167→cS168 |
| (S169, i)→S170 | S169→iS170 |
| (S170, t)→S171 | S170→tS171 |
| (S171, c)→S172 | S171→cS172 |
| (S172, h)→S173 | S172→hS173 |
| (S175, o)→S176 | S175→oS176 |
| (S176, r)→S177 | S176→rS177 |
| (S179, e)→S180 | S179→eS180 |
| (S180, t)→S181 | S180→tS181 |
| (S181, u)→S182 | S181→uS182 |
| (S182, r)→S183 | S182→rS183 |
| (S185, u)→S186 | S185→uS186 |
| (S185, r)→S191 | S185→rS191 |
| (S186, b)→S187 | S186→bS187 |
| (S187, l)→S188 | S187→lS188 |
| (S188, i)→S189 | S188→iS189 |
| (S189, c)→S190 | S189→cS190 |
| (S191, i)→S192 | S191→iS192 |
| (S191, o)→S197 | S191→oS197 |
| (S192, v)→S193 | S192→yS193 |
| (S193, a)→S194 | S193→aS194 |
| (S194, t)→S195 | S194→tS195 |
| (S195, e)→S196 | S195→eS196 |

Продолжение таблицы 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S197, t)→S198 | S197→tS198 |
| (S198, e)→S199 | S198→eS199 |
| (S199, c)→S200 | S199→cS200 |
| (S200, t)→S201 | S200→tS201 |
| (S201, e)→S202 | S201→eS202 |
| (S202, d)→S203 | S202→dS203 |

Таблица переходов конечного автомата – это таблица, которая описывает переходы между состояниями конечного автомата в зависимости от входных символов. Каждая строка таблицы соответствует одному состоянию конечного автомата, а каждый столбец соответствует одному входному символу.

В ячейках таблицы указывается номер состояния, в которое перейдет конечный автомат при получении данного входного символа из текущего состояния. Если перехода не существует, то ячейка остается пустой.

Переходы автомата показаны в таблице 2.

Переходы автомата

Таблица 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал  Состояние | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w |
| S0 | S2,S30 | S2,S38 | S2,S11 | S2,S18 | S2,S71 | S2,S3 | S2,S84 | S2 | S2,S88 | S2 | S2 | S2,S97 | S2 | S2,S101 | S2,S118 | S2,S185 | S2 | S2,S179 | S2,S158 | S2,S138 | S2,S149 | S2,S128 | S2.S123 |
| S1 |  |  |  |  | S28 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 |
| S3 | S4 |  |  |  |  |  |  |  |  |  |  | S8 |  |  | S82 |  |  |  |  |  |  |  |  |
| S4 |  |  |  |  |  |  |  |  |  |  |  | S5 |  |  |  |  |  |  |  |  |  |  |  |
| S5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S6 |  |  |  |  |
| S6 |  |  |  |  | S7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S9 |  |  |  |  |  |  |  |  |
| S9 | S10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S81 |  |  |  |
| S11 | S47 |  |  |  |  |  |  | S46 |  |  |  | S55 |  |  | S12 |  |  |  |  |  |  |  |  |
| S12 |  |  |  |  |  |  |  |  |  |  |  |  |  | S15 |  |  |  |  |  |  | S13 |  |  |
| S13 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S14 |  |  |  |
| S14 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S15 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S16 |  |  |  |  |
| S16 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S17 |  |  |  |
| S17 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S18 |  |  |  |  | S19 |  |  |  |  |  |  |  |  |  | S66 |  |  |  |  |  |  |  |  |
| S19 |  |  |  |  |  | S20 |  |  |  |  |  | S62 |  |  |  |  |  |  |  |  |  |  |  |
| S20 | S21 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S21 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S59 |  |  |
| S22 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S23 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S24 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S25 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S26 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Продолжение таблицы 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал  Состояние | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w |
| S27 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S28 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S29 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S30 |  |  |  |  |  |  |  |  |  |  |  |  |  | S31 |  |  |  |  | S33 |  | S35 |  |  |
| S31 |  |  |  | S32 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S32 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S33 |  |  |  |  |  |  |  |  |  |  |  |  | S34 |  |  |  |  |  |  |  |  |  |  |
| S34 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S35 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S36 |  |  |  |
| S36 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S37 |  |  |  |  |  |  |  |  |
| S37 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S38 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S39 |  |  | S42 |  |  |  |  |  |
| S39 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S40 |  |  |  |  |  |  |  |  |
| S40 |  |  |  |  |  |  |  |  |  |  |  | S41 |  |  |  |  |  |  |  |  |  |  |  |
| S41 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S42 |  |  |  |  | S43 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S43 | S44 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S44 |  |  |  |  |  |  |  |  |  |  | S45 |  |  |  |  |  |  |  |  |  |  |  |  |
| S45 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S46 | S53 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S47 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S48 | S50 |  |  |  |
| S48 |  |  |  |  | S49 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S49 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S50 |  |  | S51 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S51 |  |  |  |  |  |  |  | S52 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S52 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S53 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S54 |  |  |  |  |  |
| S54 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Продолжение таблицы 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал  Состояние | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w |
| S55 | S56 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S56 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S57 |  |  |  |  |
| S57 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S58 |  |  |  |  |
| S58 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S59 |  |  |  |  |  |  |  |  |  |  |  | S60 |  |  |  |  |  |  |  |  |  |  |  |
| S60 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S61 |  |  |  |
| S61 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S62 |  |  |  |  | S63 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S63 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S64 |  |  |  |
| S64 |  |  |  |  | S65 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S65 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S66 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S67 |  |  |
| S67 |  | S68 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S68 |  |  |  |  |  |  |  |  |  |  |  | S69 |  |  |  |  |  |  |  |  |  |  |  |
| S69 |  |  |  |  | S70 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S70 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S71 |  |  |  |  |  |  |  |  |  |  |  | S72 |  | S75 |  |  |  |  |  |  |  |  |  |
| S72 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S73 |  |  |  |  |
| S73 |  |  |  |  | S74 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S74 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S75 |  |  |  | S78 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S76 |  |  |
| S76 |  |  |  |  |  |  |  |  |  |  |  |  | S77 |  |  |  |  |  |  |  |  |  |  |
| S77 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S78 |  |  |  |  |  |  |  |  |  |  |  | S79 |  |  |  |  |  |  |  |  |  |  |  |
| S79 |  |  |  |  | S80 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S80 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S81 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S82 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S83 |  |  |  |  |  |

Продолжение таблицы 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал  Состояние | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w |
| S83 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S84 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S85 |  |  |  |  |  |  |  |  |
| S85 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S86 |  |  |  |
| S86 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S87 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S88 |  |  |  |  |  | S89 |  |  |  |  |  |  |  | S90 |  |  |  |  |  |  |  |  |  |
| S89 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S90 |  |  | S92 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S91 |  |  |  |
| S91 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S92 |  |  |  |  |  |  |  |  |  |  |  | S93 |  |  |  |  |  |  |  |  |  |  |  |
| S93 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S94 |  |  |
| S94 |  |  |  | S95 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S95 |  |  |  |  | S96 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S96 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S97 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S98 |  |  |  |  |  |  |  |  |
| S98 |  |  |  |  |  |  |  |  |  |  |  |  |  | S99 |  |  |  |  |  |  |  |  |  |
| S99 |  |  |  |  |  |  | S100 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S100 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S101 | S102 |  |  |  | S110 |  |  |  |  |  |  |  |  |  | S112 |  |  |  |  |  |  |  |  |
| S102 |  |  |  |  |  |  |  |  |  |  |  |  | S103 |  |  |  |  |  |  |  |  |  |  |
| S103 |  |  |  |  | S104 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S104 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S105 |  |  |  |  |
| S105 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S106 |  |  |  |  |  |  |  |
| S106 | S107 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S107 |  |  | S108 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S108 |  |  |  |  | S019 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S109 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S110 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S111 |

Продолжение таблицы 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал  Состояние | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w |
| S111 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S112 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S113 |  |  |  |
| S113 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S114 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S115 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S116 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S117 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S118 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S119 |  |  |  |  |  |
| S119 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S120 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S121 |  |  |  |  |  |  |  |  |
| S121 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S122 |  |  |  |  |  |
| S122 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S123 |  |  |  |  |  |  |  | S124 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S124 |  |  |  |  |  |  |  |  | S125 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S125 |  |  |  |  |  |  |  |  |  |  |  | S126 |  |  |  |  |  |  |  |  |  |  |  |
| S126 |  |  |  |  |  | S127 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S127 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S128 |  |  |  |  |  |  |  |  | S132 |  |  |  |  |  | S129 |  |  |  |  |  |  |  |  |
| S129 |  |  |  |  |  |  |  |  | S130 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S130 |  |  |  | S131 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S131 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S132 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S133 |  |  |  |  |  |
| S133 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S134 |  |  |  |
| S134 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S135 |  |  |
| S135 | S136 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S136 |  |  |  |  |  |  |  |  |  |  |  | S137 |  |  |  |  |  |  |  |  |  |  |  |
| S137 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S138 |  |  |  |  |  |  |  | S139 |  |  |  |  |  |  |  |  |  | S145 |  |  |  |  |  |

Продолжение таблицы 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал  Состояние | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w |
| S139 |  |  |  |  |  |  |  |  | S140 |  |  |  |  |  |  |  |  | S142 |  |  |  |  |  |
| S140 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S141 |  |  |  |  |
| S141 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S142 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S143 |  |  |  |  |  |  |  |  |
| S143 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S144 |
| S144 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S145 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S147 |  |  |
| S146 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S147 |  |  |  |  | S148 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S148 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S149 |  |  |  |  |  |  |  |  |  |  |  |  |  | S154 |  |  |  |  | S150 |  |  |  |  |
| S150 |  |  |  |  |  |  |  |  | S151 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S151 |  |  |  |  |  |  |  |  |  |  |  |  |  | S152 |  |  |  |  |  |  |  |  |  |
| S152 |  |  |  |  |  |  | S153 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S153 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S154 |  |  |  |  |  |  |  |  | S155 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S155 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S156 |  |  |  |  |  |  |  |  |
| S156 |  |  |  |  |  |  |  |  |  |  |  |  |  | S157 |  |  |  |  |  |  |  |  |  |
| S157 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S158 |  |  |  |  |  |  |  | S175 |  |  |  |  |  |  |  |  |  |  |  | D259 |  |  | S169 |
| S159 | S161 |  |  | S160 |  |  |  |  |  |  |  |  |  |  |  |  |  | S165 |  |  |  |  |  |
| S160 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S161 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S162 |  |  |  |
| S162 |  |  |  |  |  |  |  |  | S163 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S163 |  |  | S164 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S164 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S165 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S166 |  |  |
| S166 |  |  | S167 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Продолжение таблицы 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| х.сигнал  Состояние | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w |
| S167 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S168 |  |  |  |
| S168 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S169 |  |  |  |  |  |  |  |  | S170 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S170 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S171 |  |  |  |
| S171 |  |  | S172 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S172 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S173 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S175 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S176 |  |  |  |  |  |  |  |  |
| S176 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S177 |  |  |  |  |  |
| S177 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S178 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S179 |  |  |  |  | S180 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S180 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S181 |  |  |  |
| S181 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S182 |  |  |
| S182 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S183 |  |  |  |  |  |
| S183 |  |  |  |  |  |  |  |  |  |  |  |  |  | S184 |  |  |  |  |  |  |  |  |  |
| S184 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S185 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S191 |  |  | S186 |  |  |
| S186 |  | S187 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S187 |  |  |  |  |  |  |  |  |  |  |  | S188 |  |  |  |  |  |  |  |  |  |  |  |
| S188 |  |  |  |  |  |  |  |  | S189 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S189 |  |  | S190 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S190 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S191 |  |  |  |  |  |  |  |  | S192 |  |  |  |  |  | S197 |  |  |  |  |  |  |  |  |
| S192 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S193 |  |
| S193 | S194 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S194 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S195 |  |  |  |

Продолжение таблицы 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| х.сигнал  Состояние | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w |
| S195 |  |  |  |  | S196 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S196 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S197 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S198 |  |  |  |
| S198 |  |  |  |  | S199 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S199 |  |  | S200 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S200 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S210 |  |  |  |
| S201 |  |  |  |  | S202 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S202 |  |  |  | S203 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S203 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал Состояние | x | y | z | N | U | L | A-Z | \_ | \t | . | , | 0-9 | + | - | : | |,&,\*,=,>,< | !,^,% | ~ | “ | ? | / | #,}.{.].[,),(,; | $ |
| S0 | S2,S120 | S2 | S2 | S2,S114 | S2 | S2 | S2 | S2 |  |  |  | S1 | S24 | S24 | S22 | S24 | S24 | S24 | S23 | S24 | S24 | S22 |  |
| S1 |  |  |  |  |  |  |  |  | S0 | S27 | S27 | S1 |  |  |  |  |  |  |  |  |  |  |  |
| S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 | S2 |  | S2 |  | S2 |  | S2 |  |  |  | S2 |  |  |  |  | S2 |
| S3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S11 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S12 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S13 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S14 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S15 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S16 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S17 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S18 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S19 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S20 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S21 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S22 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S26 |  |  |  |  |  |  | S22 |  |
| S23 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S1 |  |  |  |  |
| S24 |  |  |  |  |  |  |  |  |  |  |  |  | S25 | S25 |  | S25 | S24 | S24 |  |  |  |  |  |
| S25 |  |  |  |  |  |  |  |  |  |  |  |  | S25 | S25 |  | S25 |  |  |  |  |  |  |  |
| S26 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S26 |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал Состояние | x | y | z | N | U | L | A-Z | \_ | \t | . | , | 0-9 | + | - | : | |,&,\*,=,>,< | !,^,% | ~ | “ | ? | / | #,}.{.].[,),(,; | $ |
| S27 |  |  |  |  |  |  |  |  |  |  |  | S1 |  |  |  |  |  |  |  |  |  |  |  |
| S28 |  |  |  |  |  |  |  |  |  |  |  |  | S29 | S29 |  |  |  |  |  |  |  |  |  |
| S29 |  |  |  |  |  |  |  |  |  |  |  | S1 |  |  |  |  |  |  |  |  |  |  |  |
| S30 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S31 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S32 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S33 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S34 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S35 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S36 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S37 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S38 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S39 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S40 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S41 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S42 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S43 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S44 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S45 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S46 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S47 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S48 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S49 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S50 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S51 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S52 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S53 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал Состояние | x | y | z | N | U | L | A-Z | \_ | \t | . | , | 0-9 | + | - | : | |,&,\*,=,>,< | !,^,% | ~ | “ | ? | / | #,}.{.].[,),(,; | $ |
| S54 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S55 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S56 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S57 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S58 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S59 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S60 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S61 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S62 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S63 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S64 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S65 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S66 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S67 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S68 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S69 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S70 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S71 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S72 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S73 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S74 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S75 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S76 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S77 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S78 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S79 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S80 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал Состояние | x | y | z | N | U | L | A-Z | \_ | \t | . | , | 0-9 | + | - | : | |,&,\*,=,>,< | !,^,% | ~ | “ | ? | / | #,}.{.].[,),(,; | $ |
| S81 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S82 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S83 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S84 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S85 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S86 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S81 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S82 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S83 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S84 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S85 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S86 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S87 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S88 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S89 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S90 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S91 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S92 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S93 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S94 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S95 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S96 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S97 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S98 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S99 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S100 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S101 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал Состояние | x | y | z | N | U | L | A-Z | \_ | \t | . | , | 0-9 | + | - | : | |,&,\*,=,>,< | !,^,% | ~ | “ | ? | / | #,}.{.].[,),(,; | $ |
| S102 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S103 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S104 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S105 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S106 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S107 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S108 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S109 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S110 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S111 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S112 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S113 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S114 |  |  |  |  | S115 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S115 |  |  |  |  |  | S116 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S116 |  |  |  |  |  | S117 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S117 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S118 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S119 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S120 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S121 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S122 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S123 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S124 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S125 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S126 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S127 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S128 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал Состояние | x | y | z | N | U | L | A-Z | \_ | \t | . | , | 0-9 | + | - | : | |,&,\*,=,>,< | !,^,% | ~ | “ | ? | / | #,}.{.].[,),(,; | $ |
| S129 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S130 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S131 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S132 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S133 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S134 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S135 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S136 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S137 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S138 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S139 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S140 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S141 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S142 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S143 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S144 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S145 |  | S146 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S146 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S147 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S148 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S149 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S150 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S151 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S152 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S153 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S154 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S155 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал Состояние | x | y | z | N | U | L | A-Z | \_ | \t | . | , | 0-9 | + | - | : | |,&,\*,=,>,< | !,^,% | ~ | “ | ? | / | #,}.{.].[,),(,; | $ |
| S156 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S157 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S158 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S159 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S160 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S161 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S162 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S163 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S164 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S165 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S166 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S167 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S168 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S169 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S170 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S171 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S172 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S173 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S175 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S176 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S177 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S178 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S179 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S180 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S181 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S182 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S183 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вх.сигнал Состояние | x | y | z | N | U | L | A-Z | \_ | \t | . | , | 0-9 | + | - | : | |,&,\*,=,>,< | !,^,% | ~ | “ | ? | / | #,}.{.].[,),(,; | $ |
| S184 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S185 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S186 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S187 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S188 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S189 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S190 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S191 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S192 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S193 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S194 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S195 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S196 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S197 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S198 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S199 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S200 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S201 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S202 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S203 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

* 1. Построение КС-грамматики

Построение контекстно-свободной грамматики для своего варианта задания:

Фрагмент программы для построения КС-грамматики: if (a > b) {

cout << " A - winner!" << endl<<endl;

cout << "Round 2. A versus C. Final fight!" << endl << endl; if (a > c)

cout << "A - winner winner chicken dinner!" << endl << endl; cout << "Round 3. Loosers fight! B versus C." << endl << endl;

b > c ? cout << "B - the best of the bad" << endl : cout << "C - the best of the bad" << endl;

}

КС – грамматика

G = (T, N, P, S), где

T — набор (алфавит) терминальных символов

N — набор (алфавит) нетерминальных символов

P — набор правил вида: «левая часть» —> «правая часть», где:

«левая часть» — непустая последовательность терминалов и нетерминалов, содержащая хотя бы один нетерминал

«правая часть» — любая последовательность терминалов и нетерминалов

S — стартовый (или начальный) символ грамматики из набора нетерминалов.

T = { if, else, id, (, ), >,>=,=.==,<=, <, {, }, cout, cin, <<, text, const, endl, ; ? ! : +

- \* / <<, >>} S = { S }

N = {S, A, C, D, H, K, J, I, L, G, T, U, Z, O, Q, N, X} P = {

S -> if L {C A D} | if L {CD} | if L {CAD} else C A-> if LKK | if LK else K | if LK

C - > KK D -> H : K

H -> GZK K->JII | JI J - > Q<<N

I -> <<X | <<X;

L -> (G)

G -> T > T

T -> id

U -> << | >>

Z -> ( | ) | { | }| ; | ?| ! | :

O -> | >= | = | == | <= | < |+ | - | \* | / Q -> cout | cin

X -> endl

N -> text | const

}

Правостороннее порождение:

S => if L {C A D} => if L {C A H : K } => if L {C A H : JI } =>

if L {C A H : J<<X; } =>

if L {C A H : J<< endl; } =>

if L {C A H : Q<<N << endl; } =>

if L {C A H : Q<<text << endl; } =>

if L {C A H : cout<<text << endl; } => if L {C A H : cout<<text << endl; } =>

if L {C A GZK : cout<<text << endl; } => if L {C A GZ JI : cout<<text << endl; } =>

if L {C A GZ J <<X : cout<<text << endl; } =>

if L {C A GZ J <<endl : cout<<text << endl; } =>

if L {C A GZ Q<<N <<endl : cout<<text << endl; } => if L {C A GZ Q<<text <<endl : cout<<text << endl; } =>

if L {C A GZ cout<<text <<endl : cout<<text << endl; } => if L {C A G ? cout<<text <<endl : cout<<text << endl; } =>

if L {C A T > T ? cout<<text <<endl : cout<<text << endl; } => if L {C A T > id ? cout<<text <<endl : cout<<text << endl; } => if L {C A id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if LKK id > id ? cout<<text <<endl : cout<<text << endl; } => if L {C if LK JII id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if LK JI <<X; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if LK JI <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } => if L {C if LK J <<X <<endl; id > id ? cout<<text <<endl : cout<<text << endl; }

=>

if L {C if LK J <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl;

} =>

if L {C if LK Q<<N <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if LK Q<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if LK cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text

<< endl; } =>

if L {C if L JII cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if L JI<<X; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if L JI<<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if L J<<X <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if L J<<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text

<<endl : cout<<text << endl; } =>

if L {C if L Q<<N <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text

<<endl : cout<<text << endl; } =>

if L {C if L Q<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if L cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if (G) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if (T > T) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if (T > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L {C if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L { KK if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id

? cout<<text <<endl : cout<<text << endl; } =>

if L { K JI<<X; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L { K JII if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L { K JI<<X; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L { K JI<<endl; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl

<<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L { K J<<X <<endl; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl

<<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L { K J<<endl <<endl; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl

<<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L { K Q<<N <<endl <<endl; if (id > id) cout<<text <<endl <<endl; cout<<text

<<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L { K Q<<text <<endl <<endl; if (id > id) cout<<text <<endl <<endl; cout<<text

<<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L { K cout<<text <<endl <<endl; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } => if L { JII cout<<text <<endl <<endl; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } => if L { JI<<X; cout<<text <<endl <<endl; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } => if L { JI<<endl; cout<<text <<endl <<endl; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } => if L { J<<X <<endl; cout<<text <<endl <<endl; if (id > id) cout<<text <<endl

<<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L { J<<endl <<endl; cout<<text <<endl <<endl; if (id > id) cout<<text <<endl

<<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L { Q<<text <<endl <<endl; cout<<text <<endl <<endl; if (id > id) cout<<text

<<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl : cout<<text << endl; } =>

if L { cout<<text <<endl <<endl; cout<<text <<endl <<endl; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl

: cout<<text << endl; } =>

if (G) { cout<<text <<endl <<endl; cout<<text <<endl <<endl; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl

: cout<<text << endl; } =>

if (T>T) { cout<<text <<endl <<endl; cout<<text <<endl <<endl; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl

: cout<<text << endl; } =>

if (T>id) { cout<<text <<endl <<endl; cout<<text <<endl <<endl; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl

: cout<<text << endl; } =>

if (id >id) { cout<<text <<endl <<endl; cout<<text <<endl <<endl; if (id > id) cout<<text <<endl <<endl; cout<<text <<endl <<endl; id > id ? cout<<text <<endl

: cout<<text << endl; }

Дерево вывода представлено на рисунке 6.

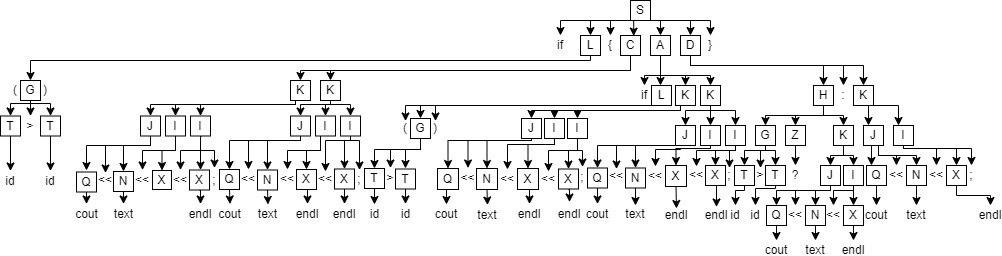


Рис. 6. Дерево вывода

4. Логическое проектирование

* 1. Блок-схемы

С помощью блок-схем можно рассмотреть работу алгоритма программы в целом или в конкретные ее функции. Ниже представлена общая блок-схема алгоритма программы (рис. 7).



Рис. 7. Общая блок-схема алгоритма программы

На рисунках 8-9 представлена блок-схема очищения кода удаления лишних

пробелов, путых строк, табуляций, удаление многострочных и однострочных комментариев.

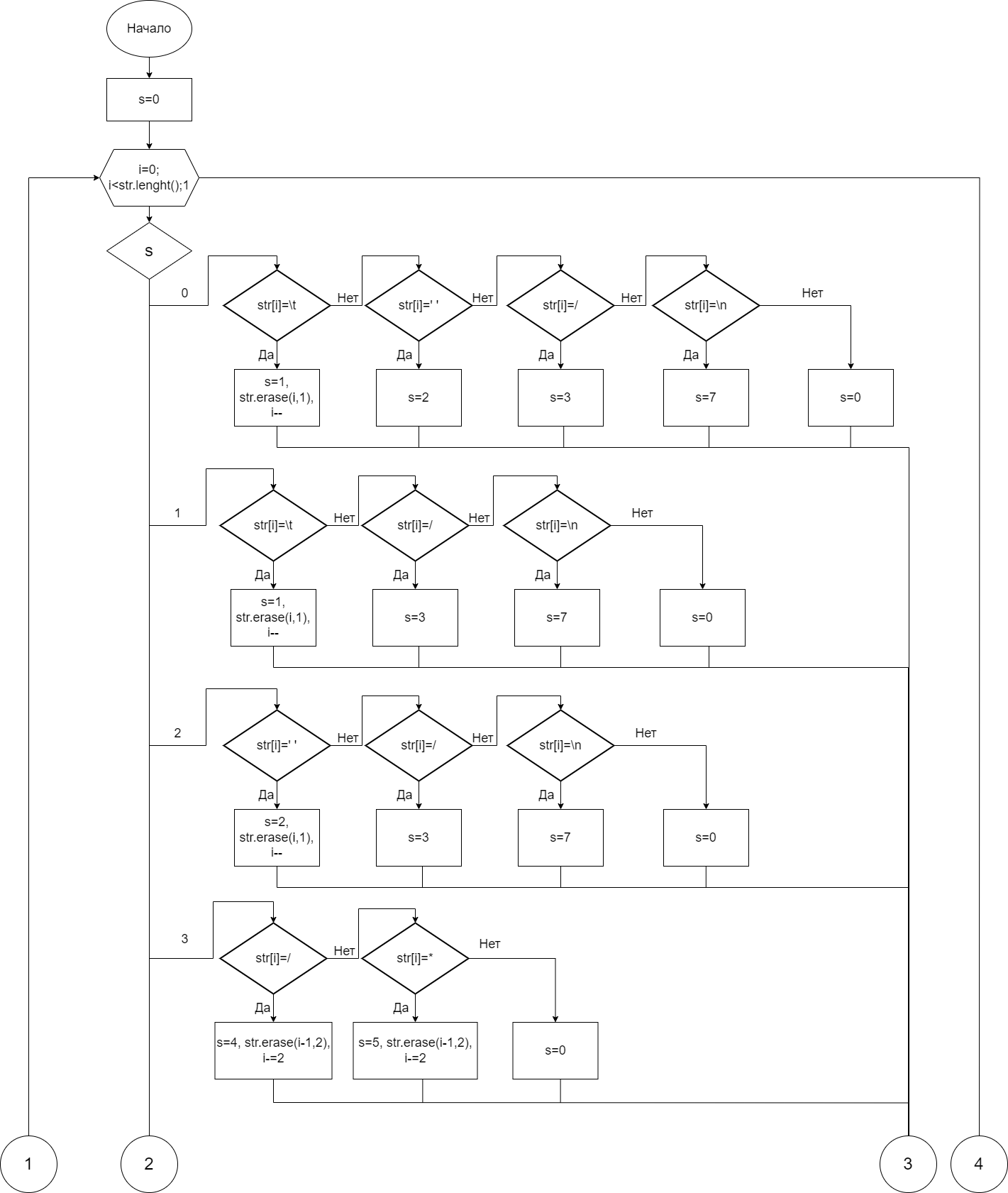


Рис. 8. Блок-схема удаления комментариев часть 1

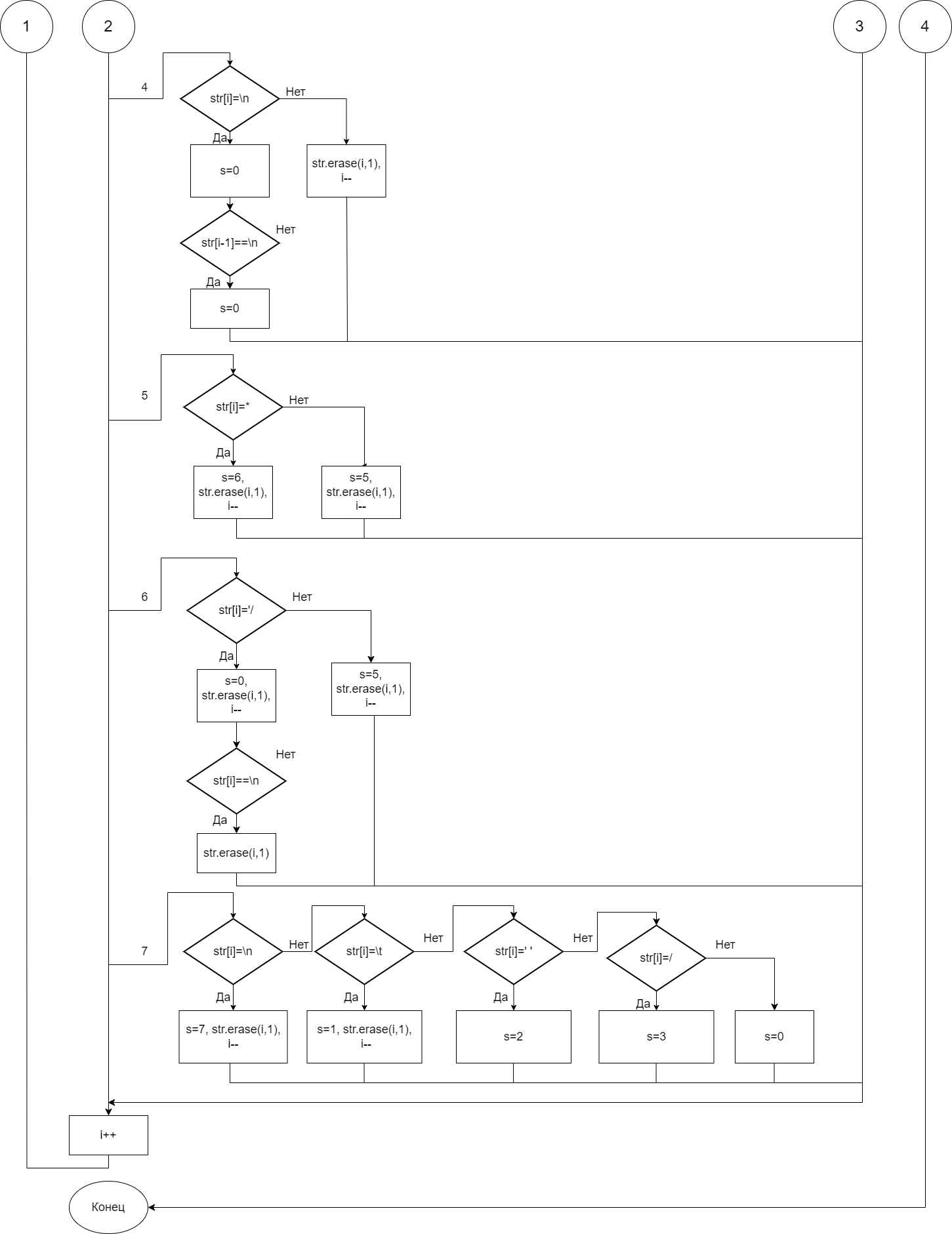


Рис. 9. Блок-схема удаления комментариев часть 2 На рисунках 10-11 представлена блок-схема выявления ошибок.

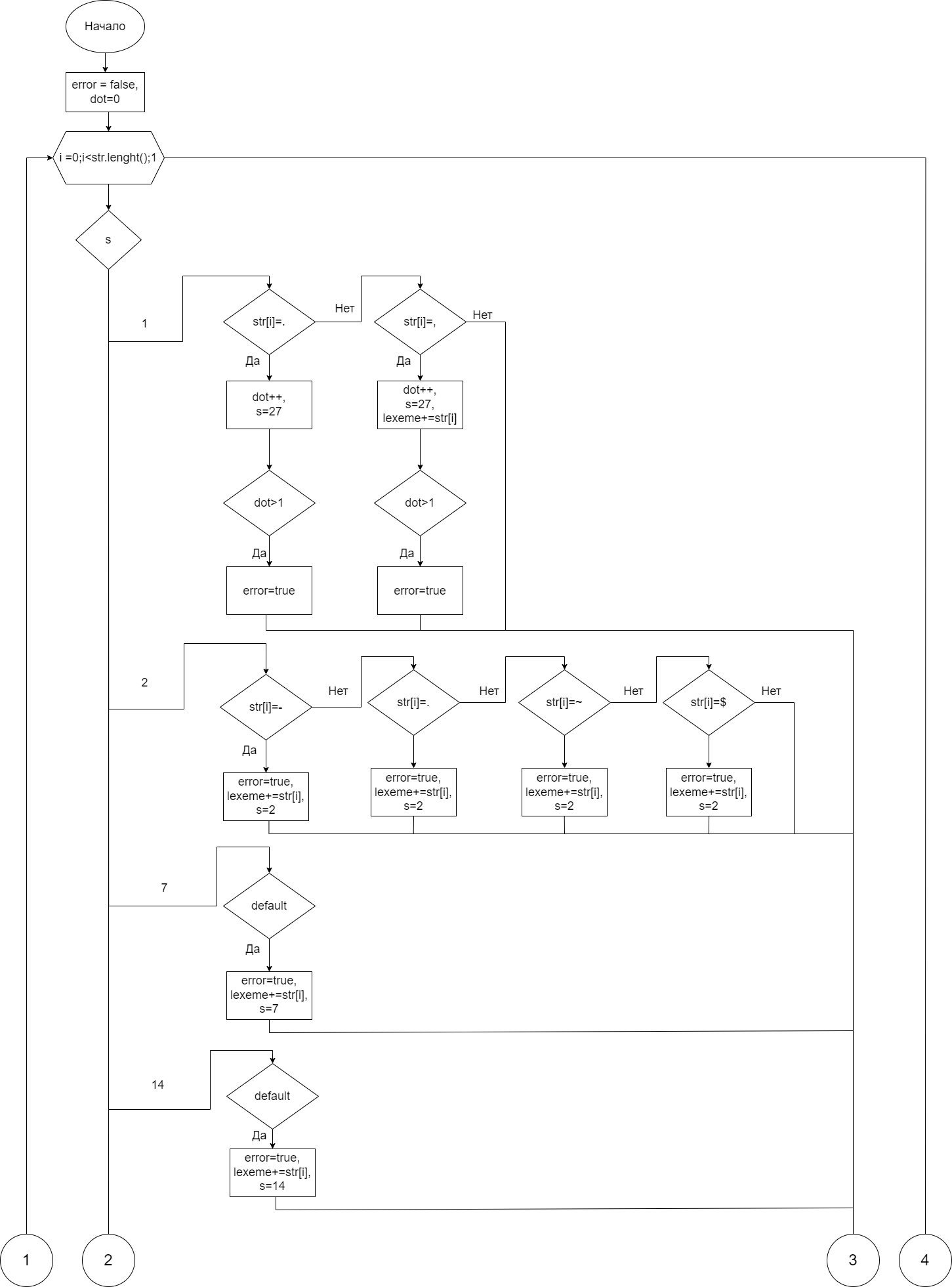


Рис. 10. Блок-схема выявления ошибок часть 1

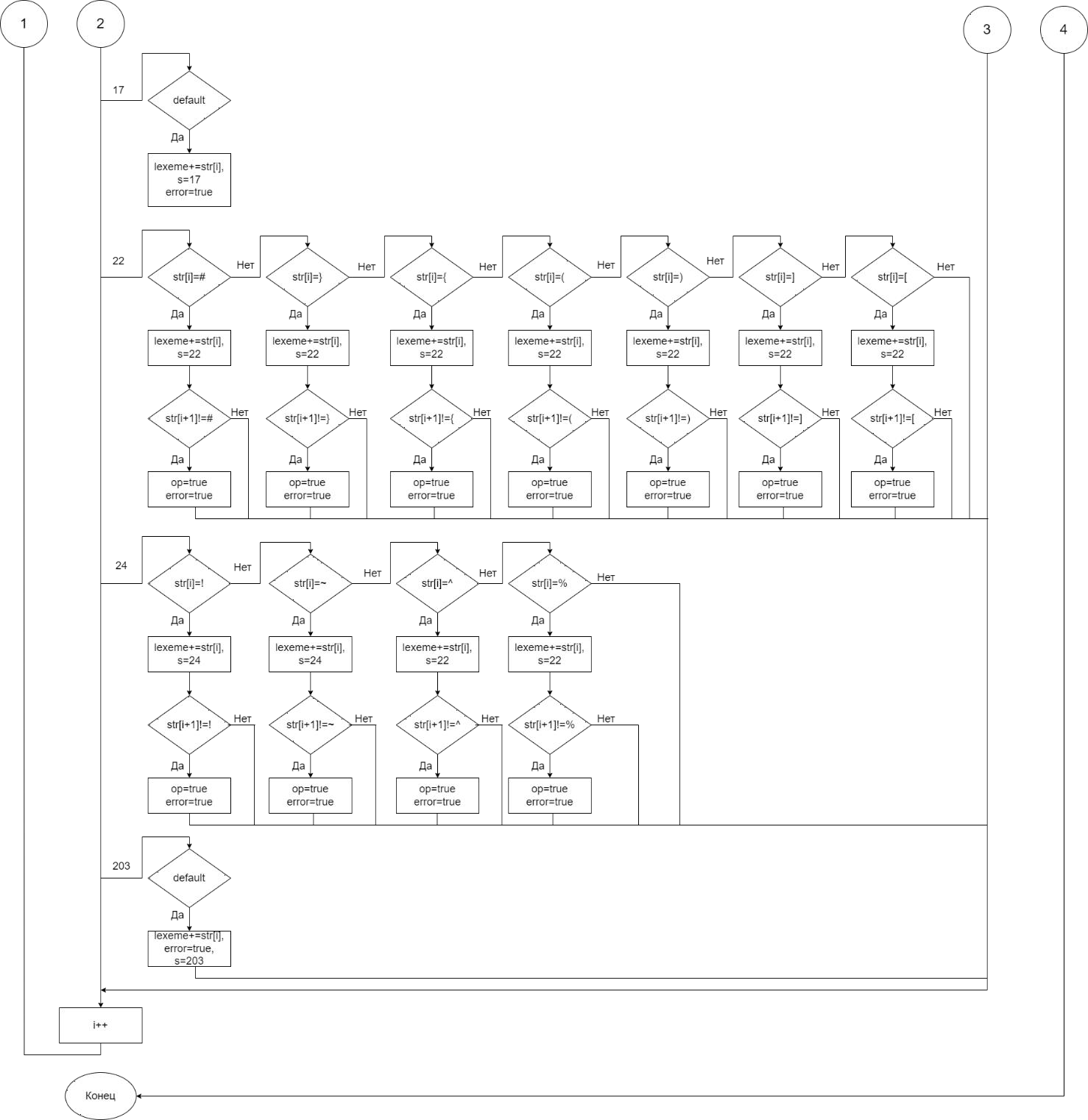


Рис. 11. Болок-схема выявления ошибок часть 2

* 1. Словесное описание работы алгоритма на примере разбора входных последовательностей
     1. Словесное описание алгоритма нахождения комментариев и лишних символов в тексте

Загружается или записывается текст программы. В нем при помощи конечного автомата находятся комментарии, которые начинаются как с символа «//» (однострочные), так и с символа «/\*» (многострочный). Если

встречается однострочный комментарий, то он удаляется пока не встретится символ перехода на новую строку, если же встретился многострочный комментарий, то он удаляется пока не встретится символ завершения многострочного комментария «\*/». Если встречается один пробел, то он остается без изменений, а все последующие за ним пробелы удаляются. Тоже самое делается и с символом переноса строки «\n». Если встречаем символ табуляции «\t», то он удаляеся.

4.2.3 Словесное описание алгоритма вывода ошибок

Определяется тип ошибки в зависимости от того, в каком состоянии в момент нахождения ошибки был конечный автомат и затем выводит ошибку в поле «Вывод ошибок» с её типом и номмером строки.

4.3. Определение сложности полученного алгоритма

Разработанный алгоритм, моделирующий работу лексического анализатора, является линейным, так как в алгоритме используется один цикл с количеством повторений n, а все используемые в нем циклы уменьшают или увеличивают переменную счетчика основного цикла. Следовательно сложность созданного алгоритма равна O(n).

Физическое проектирование

* 1. Выбор структур данных

Физическое проектирование лексического анализатора выполнялось на языке высокого уровня С++. В программе задействованы переменные, представленные в таблице 3.

Таблица 3

Структуры данных

|  |  |  |
| --- | --- | --- |
| Наименование | Обозначение | Тип |
| Входная строка | str | string |
| Текушее состояние | s | int |
| Проверка на наличие лексемы в таблице | error | bool |
| Временная строка | q | string |
| Лексема | lexeme | string |
| Подсчёт символа ‘.’ | dot | int |
| Индекс проверяемого символа | i | int |
| Номер строки, в которой обнаружена ошибка | value | int |
| Проверка на конечную лексему | op | bool |
| Количество символа ‘\n’ в неочищенном коде | n | int |

* 1. Спецификация функций

Функции, использованные в программе, представлены в таблице 4.

Таблица 4

Спецификации функций

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Обозначение | Тип | Параметры |
| Обработка нажатия  кнопки «Выбрать файл» | button1\_Click | System::Void | object sender, EventArgs e |
| Обработчик нажатия  кнопки «Рефакторинг» | button2\_Click | System::Void | object sender, EventArgs e |
| Преобразование типа данных из String^ в  std::string | MarshalString | void | String^ s, string& os |
| Очистка кода | ref | void | - |
| Распознование лексем и  ошибок | recognizer | void | - |

Проектирование интерфейса

Интерфейс программы содержит в себе поле под названием «Исходный код», куда пользователь будет вводить код программы. (рис. 12).

Ниже поля «Исходный код» расположена кнопка «Выбрать файл» (рис. 13): она открывает окно «Проводника», в котором можно выбрать файл с кодом на компьютере.

Правее поля «Исходный код» расположено поле «Обработанный код», в котором выводится очищенный код с пронумерованными строками (рис. 14).

Ниже поля «Обработанный код» находится кнопка «Рефакторинг», которая запускает лексический анализатор, который удаляет комментарии и лишние символы из кода, нумерует строки, распределяет текст по классам лексем, выводит псевдо- и дескрипторный коды, ошибки (рис. 15).

Правее поля «Обработанный код» находится поле «Вывод ошибок», в котором отображаются ошибки, обнаруженные в коде (рис 16).

Ниже кнопок «Выбрать файл» и «Рефакторинг» располагаются поля

«Дескрипторныый код» и «Псевдокод» (рис. 17).

Ниже полей «Дескрипторныый код» и «Псевдокод» располагаются таблицы лексем, в них лексический анализатор заносит лексемы в зависимости от их класса (рис. 18).

Классы лексем:

* константы;
* идентификаторы;
* ключевые слова;
* Знаки;
* Операции.

На рисунке 28 показан интерфейс программы целиком.

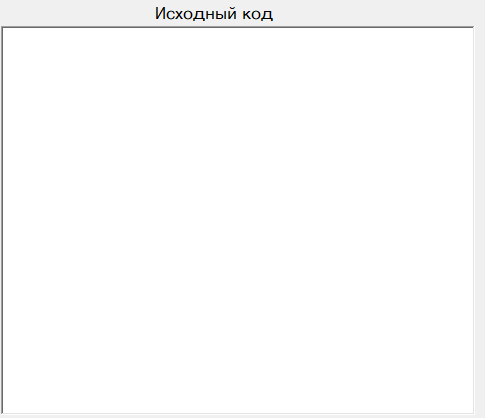


Рис. 12. Поле «Исходный код»



Рис. 13. Кнопка «Выбрать файл»

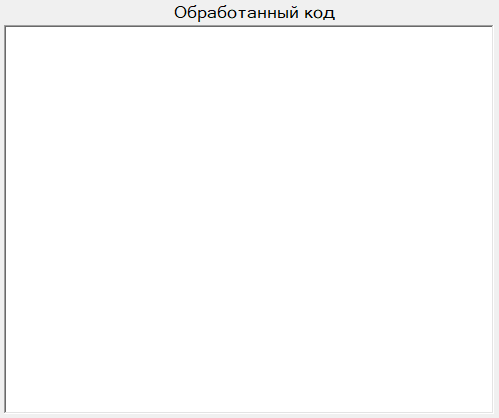


Рис. 14. Поле «Обработанный код»



Рис. 15. Кнопка «Рефакторинг»

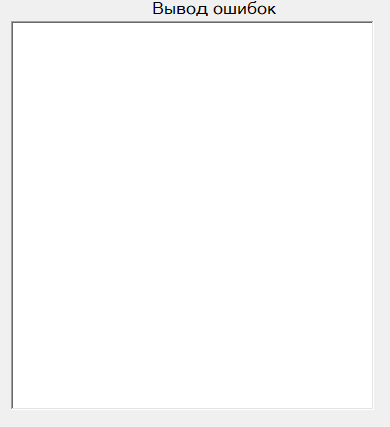


Рис. 16. Поле «Вывод ошибок»



Рис. 17. Поля «Дескрипторный код» и «Псевдокод»

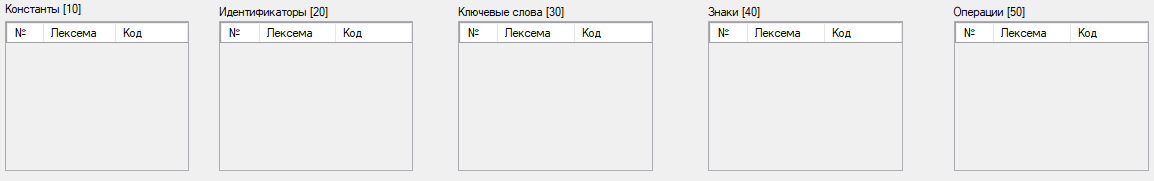


Рис. 18. Таблицы лексем

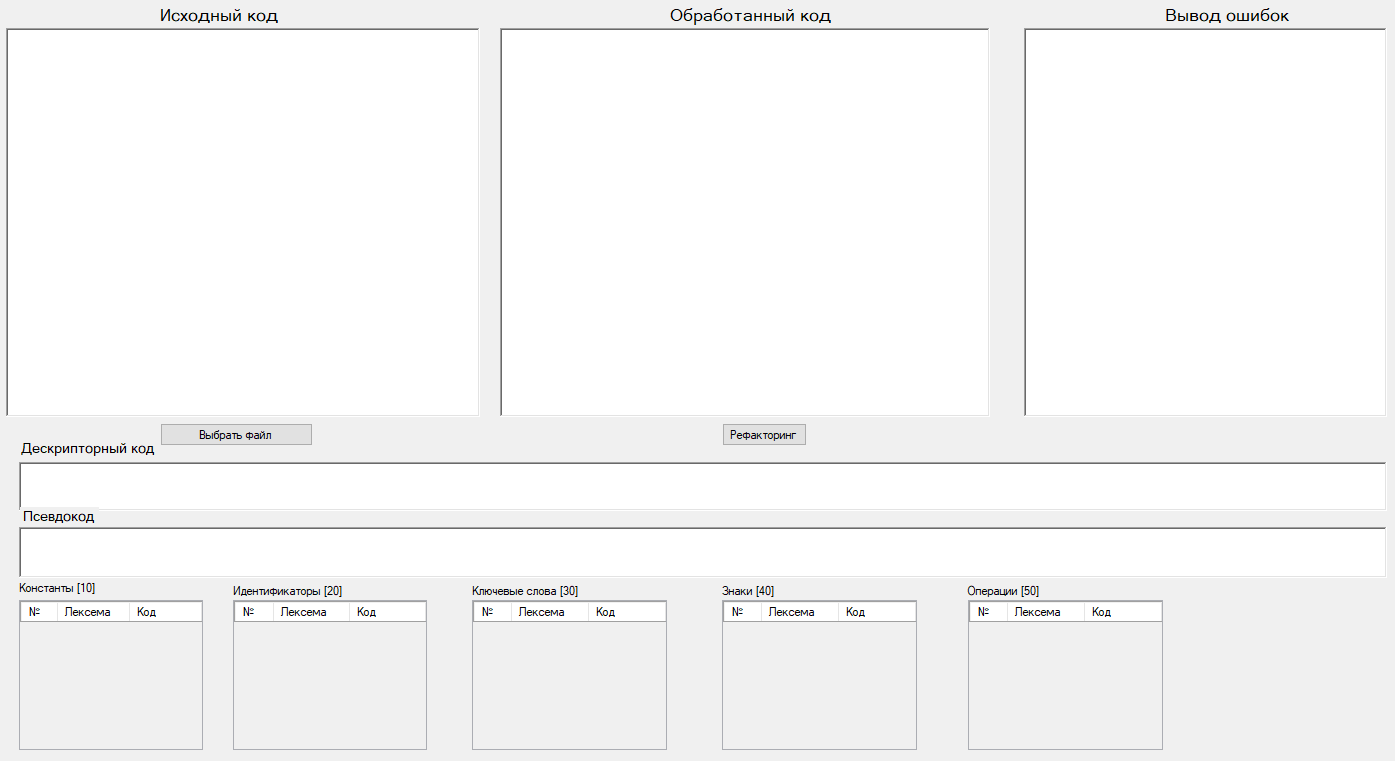


Рис. 19. Интерфейс программы

Тестирование программного обеспечения

Результаты тестирования представлены на рисунках 20 – 27. Тестирование на очищение кода и нумерацию строк:

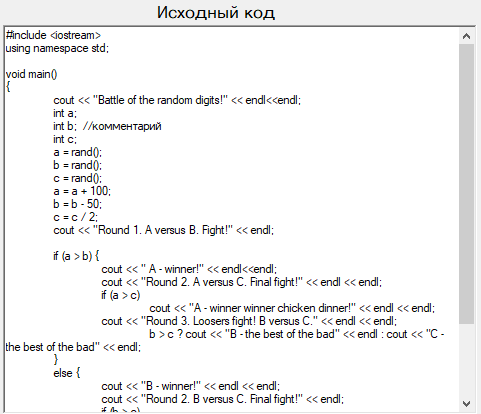


Рис. 20. Исходный код

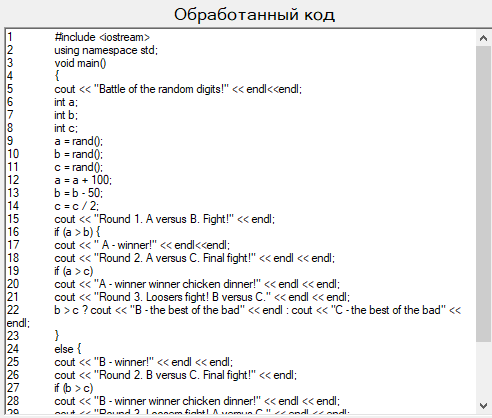


Рис. 21. Обработанный код.

Результат: успешно был очищен код и пронумерованы строки.

Тестирование на определение классов лексем.

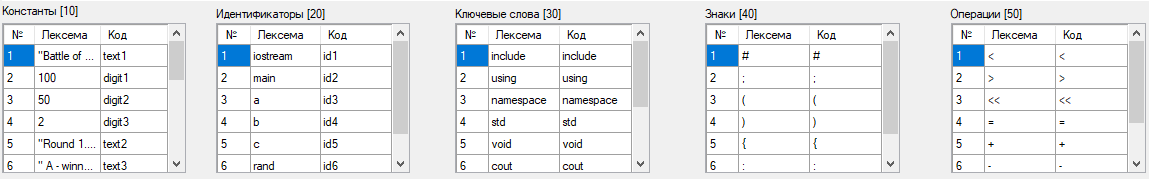


Рис. 22. Классы лексем.

Результат: все лексемы были определены верно.

Тестирование на правильность вывода псевдо- и дескрипторного кодов:

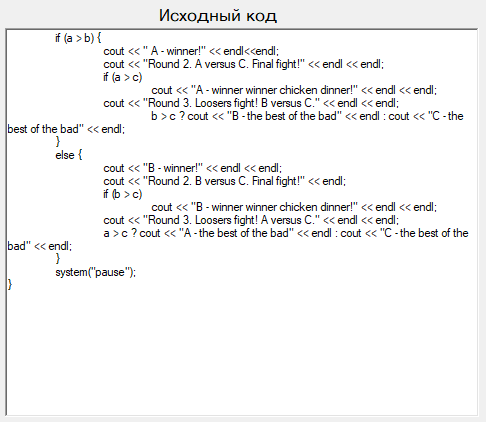


Рис. 23. Фрагмент кода программы

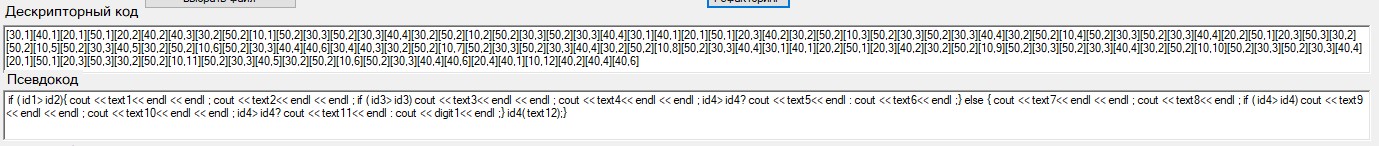


Рис. 24. Псевдо- и дескрипторный коды

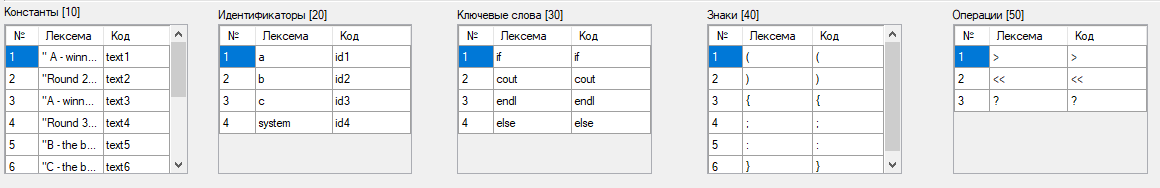


Рис. 25. Получившиеся классы лексем Результат: псевдо- и дескрипторные кода выводятся правильно. Тестирование на нахождение различных ошибок в коде:

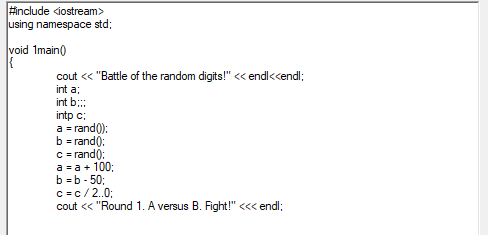


Рис. 26. Код с ошибками

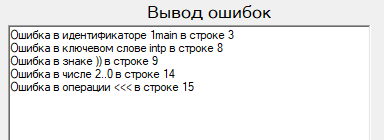


Рис. 27. Сообщение об ошибках Результат: ошибки были найдены и идентифицированы верно.

Результаты тестирования программы представлены в таблице 5.

Результаты тестирования

Таблица 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата  тестирования | Тестируемый  модуль | Кто проводил  тестирование | Описание теста | Результат  тестирования |
| 09.05.2023 | Проект1.sln | Трофимов Н.В. | Ввод текста программы для проверки определения  классов лексем | Успех |
| 13.05.2023 | Проект1.sln | Трофимов Н.В. | Ввод текста программы с комментариями и лишними  символами | Успех |
| 21.05.2023 | Проект1.sln | Трофимов Н.В. | Проверка на правильность вывода дескрипторного  кода и псевдокода | Успех |
| 03.06.2023 | Проект1.sln | Трофимов Н.В. | Проверка на нахождение различных ошибок  в коде | Успех |

Заключение

В ходе выполнения данной курсовой работы был рассмотрен лексический и синтаксический анализ, изучена классификация грамматик Хомского, связь лексического анализа с регулярной грамматикой и конечным автоматом, связь синтаксического анализа с КС-грамматикой.

По результатам курсовой работы была спроектирована и разработана программа лексического анализатора на языке высокого уровня C++, которая удаляет комментарии, двойные пробелы, пустые строки, подсчитывает количество строк, а также проводит лексический анализ входной последовательности символов, т.е. распознаёт лексемы, выводит дескрипторный код и псевдокод, а также выводит сообщения об ошибках из технического задания (приложение 1)(если они были допущены).

Также была оценена сложность алгоритма программы лексического анализатора, построен граф конечного автомата, регулярная грамматика по конечному автомату, КС-грамматика, которая описывает работу с классами и объектами языка C++, левостороннее порождение и дерево вывода.

К программе лексического анализатора написано руководство пользователя (приложение 3).

Список литературы

1. Е.В. Ершов, д-р техн. наук, проф.; Л.Н. Виноградова и др. «Методика и организация самостоятельной работы студентов» Коллектив авторов, 2012 ФГБОУ ВПО «Череповецкий государственный университет», 2012. – 208 с.
2. «Лекции по теории автоматов» [Электронный ресурс] https://sites.google.com/site/formlang/lecturenotes. Дата обращения: 01.06.2023
3. Теория автоматов. Ю.Г. Карпов – СПб.: Питер, 2003. – 208 с.
4. «Информация по теории автоматов» [Электронный ресурс] https://[www.intuit.ru/.](http://www.intuit.ru/) Дата обращения: 01.06.2023
5. А.А. Ожиганов Теория автоматов. Учебное пособие - Санкт-Петербург: НИУ ИТМО, 2013. - 84 с. - экз.
6. В. А. Сокол Введение в теорию формальных языков. Учебное пособие – Ярославль: ЯрГУ, 2014. – 208 с.
7. Е.И. Чигорина, М.А. Шамашов Теория конечных автоматов и формальных языков. Учебное пособие - Самара: Изд-во Самар. гос. аэрокосм, ун-та, 2007. 96 с.: ил.
8. «Справочник по теории автоматов» [Электронный ресурс] - https://neerc.ifmo.ru/wiki/. Дата обращения: 01.06.2023
9. Ганичева О.Г. Теория языков программирования и методы трансляции: Учеб. Пособие. – Череповец: ГОУ ВПО ЧГУ, 2011. – 186 с.

Приложение 1 Техническое задание

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий

наименование института (факультета)

Математическое и программное обеспечение ЭВМ

наименование кафедры

Модуль «Информатика»

наименование дисциплины в соответствии с учебным планом

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ , д.т.н., профессор Ершов Е.В.

« » 2023г.

Разработка алгоритмического обеспечения и построение лексического и синтаксического анализаторов

Техническое задание на курсовую работу Листов 7

Руководитель: Ганичева О.Г и Пышницкий К.М.

Ф.И.О. преподавателя

Исполнитель:

студент 1Пиб-01-1оп-21

группа

Трофимов Н.В.

Фамилия, имя, отчество

2023 год

Введение

Данная курсовая работа посвящена разработке программы, представляющей собой лексический анализатор, реализованная как конечный автомат.

1. Основания для разработки

Основанием для разработки является задание на курсовую работу по дисциплине Модуль «Информатика», выданное на кафедре МПО ЭВМ ИИТ ЧГУ.

Дата утверждения: 16 февраль 2023 года

Наименование темы разработки: «Разработка алгоритмического обеспечения и построение лексического и синтаксического анализаторов».

1. Назначение разработки

Основной задачей курсовой работы является освоение на практике материала, полученного в ходе изучения дисциплин «Теория автоматов и формальных языков» и «Структуры и алгоритмы обработки данных», а также разработка программы, моделирующей работу лексического анализатора.

Программа должна решать следующие задачи: удалять лишние пробелы и комментарии, выделять из текста входной программы все лексемы, подсчитывать количество строк в программе.

1. Требования к программе
   1. Требования к функциональным характеристикам Модель лексического анализатора должна:
      1. Считывать файл в форму с возможностью его корректировки;
      2. Удалять лишние пробелы, знаки табуляции, комментарии, излишние переходы на новую строку;
      3. Выполнять подсчет количества строк в программе;
      4. Выводить таблицы классов лексем;
      5. Выводить дескрипторный код и псевдокод;
      6. Выводить ошибки с указаением строки, в которой была допущена ошибка и типа этой ошибки.

Дополнительные требования:

* Работа лексического анализатора должна моделироваться конечным автоматом;
* Сделать описание конечного автомата в виде графа и табличного описания;
* Построить КС-грамматику и дерево вывода;
* Выполнить оценку сложности алгоритма.
  1. Требования к надежности

Для правильной работы программы необходимо:

* количество вводимых пользователем символов кода в текстовом поле программы не должно превышать 65528;
* вводимые лексемы должны удовлетворять возможным лексемам, которые определены с помощью состояний конечного автомата.
  1. Условия эксплуатации

Приложение должно исполняться на исправном ПК с установленными драйверами Microsoft Visual C++.

* 1. Требования к составу и параметрам технических средств Минимальные системные требования:
* Оперативную память не менее 2 ГБайт;
* Архитектуру процессора x86-64;
* Операционную систему Windows;
* Процессор с тактовой частотой не менее 1,2 GHz.
  1. Требования к информационной и программной совместимости

Код программы написан на языке высокого уровня С++ и скомпилирован в Microsoft Studio 2022 на операционной системе Windows 11.

* 1. Требования к маркировке и упаковке

Программа будет распространятся через копирование исходных файлов на СD-дисках.

* 1. Требования к транспортированию и хранению

Файлы, требуемые для корректной работы программы, необходимо расположить на СD-диске либо памяти компьютера.

* 1. Специальные требования Требования не предъявляются

1. Требования к программной документации
   1. Содержание расчётно-пояснительной записки

Программная документация должна содержать расчётно- пояснительную записку с содержанием:

Титульный лист Оглавление Введение

1. Изучение и описание предметной области
2. Основная часть
3. Логическое проектирование
4. Физическое проектирование
5. Проектирование интерфейса
6. Тестирование программного обеспечения Заключение

Список литературы

Приложения 1 Техническое задание Приложения 2 Текст программы Приложение 3 Руководство пользовотеля Приложение 4 Граф конечного автомата

* 1. Требования к содержательной части

Требования к оформлению, установленные ГОСТ, должны быть выполнены на протяжении всей работы без каких-либо изменений в таблице П1.1.

Таблица П1.1

Требования к оформлению

|  |  |
| --- | --- |
| Документ | Печать на отдельных листах формата А4 (210х297 мм); оборотная сторона не  заполняется; листы нумеруются. Печать возможна ч/б. |
| Страницы | Ориентация — книжная; отдельные страницы, при необходимости, альбомная.  Поля: верхнее, нижнее — по 2 см, левое — 3 см, правое — 1 см. |
| Абзацы | Межстрочный интервал — 1, перед и после абзаца — 0. |
| Шрифты | Кегль — 14. В таблицах шрифт 12. Шрифт листинга — 10 (возможно в 2  колонки). |
| Рисунки | Подписывается под ним по центру: «Рис.Х. Название В» приложениях:  «Рис.П1.3. Название» |
| Таблицы | Подписывается: над таблицей, выравнивание по правому: «Таблица Х». В следующей строке по центру Название Надписи в «шапке» (имена столбцов, полей) — по центру. В теле таблицы (записи) текстовые значения —  выравнены по левому краю, числа, даты — по правому. |

1. Стадии и этапы разработки

Стадии и этапы разработки, представлены в таблице П.1.2.

Стадии и этапы разработки

Таблица П.1.2

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапа разработки | Сроки разработки | Результат выполнения | Отметка о выполнении |
| Оформление технического задания | 1.05.2023 | Утверждено техническое задание | Выполнено |
| Разработка общего  алгоритма | 10.05.2023 | Создан алгоритм решения задания | Выполнено |
| Создание алгоритма  удаления пробелов | 20.05.2023 | Создан алгоритм удаления  пробелов | Выполнено |
| Создание алгоритма  создания таблиц лексем | 1.06.2023 | Создан алгоритм создания таблиц лексем | Выполнено |
| Создание алгоритма создания псевдокода и  дескрипторного кода | 2.06.2023 | Создан алгоритм создания псевдокода и дескрипторного кода | Выполнено |
| Создание алгоритма  обнаружения ошибок | 2.06.2023 | Создан алгоритм обнаружения  ошибок | Выполнено |
| Написание программы | 5.06.2023 | Написаная программы | Выполнено |
| Создание КС-  грамматики | 5.06.2023 | Созданная КС-грамматики | Выполнено |
| Создание таблиц  переходов | 10.06.2023 | Созданые таблицы переходов | Выполнено |
| Написание РПЗ | 17.06.2023 | Написанное РПЗ | Выполнено |

1. Порядок контроля и приемки

Порядок контроля и приемки, представлен в таблице П. 1.3.

Порядок контроля и приемки

Таблица П.1.3

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование контрольного этапа выполнения курсовой  работы | Сроки контроля | Результат выполнения | Отметка о приемке результата контрольного этапа |
| 1 | 2 | 3 | 4 |
| Сдача технического  задания | 10.05.2023 | Готовое техническое  задание | Выполнено |
| Демонстрация работы  функции удаления пробелов | 19.05.2023 | Функция удаления пробелов работет | Выполнено |

Продолжение табл. П1.3

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Демонстрация работы  лексического анализатора | 05.06.20  23 | Работа лексического анализатора продемонстрирована | Выполнено |
| Демонстрация  работы программы | 05.06.20  23 | Работа программы продемонстрирована | Выполнено |
| Демонстрация КС-  грамматики | 05.06.20  23 | КС-грамматика продемонстрирована | Выполнено |
| Защита курсовой работы | 23.06.20  23 | Оценка |  |

Приложение 2 Текст программы

#pragma once #include <algorithm> #include <stdlib.h> #include <string>

#include <msclr/marshal\_cppstd.h> #include <iostream>

#include <fstream> #include <string.h> using namespace System;

using namespace msclr::interop; using namespace std;

namespace Проект1 { using namespace System;

using namespace System::ComponentModel; using namespace System::Collections;

using namespace System::Windows::Forms; using namespace System::Data;

using namespace System::Drawing; using namespace System::IO;

void MarshalString(String^ s, string& os) { //функция преобразование типа данных из String^ в std::string using namespace Runtime::InteropServices;

const char\* chars = (const

char\*)(Marshal::StringToHGlobalAnsi(s)).ToPointer(); os = chars; Marshal::FreeHGlobal(IntPtr((void\*)chars));

}

string q; string lexeme; string str;

void ref() { // функция очищения кода int s = 0;

for (int i = 0; i < str.length(); i++) { switch (s)

{

case 0:

switch (str[i]) { case '\t':

s = 1;

str.erase(i, 1); i--;

break; case ' ': s = 2;

break; case '/': s = 3;

break; case '\n': s = 7;

break; default: s = 0;

break;

}

break;

case 1:

switch (str[i]) { case '\t':

s = 1;

str.erase(i, 1); i--;

break; case '/':

s = 3;

break; case '\n':

s = 7;

break; default:

s = 0;

break;

}

break; case 2:

switch (str[i]) { case ' ':

s = 2;

str.erase(i, 1); i--;

break; case '/':

s = 3;

break; case '\n':

s = 7;

break; default:

s = 0;

break;

}

break; case 3:

switch (str[i]) { case '/':

s = 4;

str.erase(i - 1, 2);

i -= 2;

break; case '\*':

s = 5;

str.erase(i - 1, 2);

i -= 2;

break; default:

s = 0;

break;

}

break; case 4:

switch (str[i]) { case '\n':

s = 0;

if (str[i - 1] == '\n') { str.erase(i, 1);

}

break; default: str.erase(i, 1); i--;

break;

}

break; case 5:

switch (str[i]) { case '\*':

s = 6;

str.erase(i, 1); i--;

break; default:

s = 5;

str.erase(i, 1); i--;

break;

}

break; case 6:

switch (str[i]) { case '/':

s = 0;

str.erase(i, 1); i--;

if (str[i] == '\n') {

str.erase(i, 1);

}

break; default:

s = 5;

str.erase(i, 1); i--;

break;

}

break; case 7:

switch (str[i]) { case '\n':

s = 7;

str.erase(i, 1); i--;

break; case '\t':

s = 1;

str.erase(i, 1); i--;

break; case ' ': s = 2;

break; case '/': s = 3;

break; default: s = 0;

break;

}

break;

}

}

}

/// <summary>

/// Сводка для MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: добавьте код конструктора

//

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::Button^ button1; protected:

private: System::Windows::Forms::Button^ button2; private: System::Windows::Forms::RichTextBox^ richTextBox1;

private: System::Windows::Forms::RichTextBox^ richTextBox2;

private: System::Windows::Forms::OpenFileDialog^ openFileDialog1;

private: System::Windows::Forms::DataGridView^ dataGridView1;

private: System::Windows::Forms::DataGridView^ dataGridView2;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column1;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column2;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column3;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column4;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column5;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column6;

private: System::Windows::Forms::RichTextBox^ richTextBox3;

private: System::Windows::Forms::RichTextBox^

richTextBox4;

private: System::Windows::Forms::RichTextBox^ richTextBox5;

private: System::Windows::Forms::DataGridView^ dataGridView3;

private: System::Windows::Forms::DataGridView^ dataGridView4;

private: System::Windows::Forms::DataGridView^ dataGridView5;

private: System::Windows::Forms::Label^ label1; private: System::Windows::Forms::Label^ label2; private: System::Windows::Forms::Label^ label3; private: System::Windows::Forms::Label^ label4; private: System::Windows::Forms::Label^ label5; private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column7;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column8;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column9;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column10;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column11;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column12;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column13;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column14;

private: System::Windows::Forms::DataGridViewTextBoxCol umn^ Column15;

private: System::Windows::Forms::Label^ label6;

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary> System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора

— не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

this->button1 = (gcnew System::Windows::Forms::Button()); this->button2 = (gcnew

System::Windows::Forms::Button()); this->richTextBox1 = (gcnew

System::Windows::Forms::RichTextBox()); this->richTextBox2 = (gcnew System::Windows::Forms::RichTextBox()); this->openFileDialog1 = (gcnew System::Windows::Forms::OpenFileDialog()); this->dataGridView1 = (gcnew System::Windows::Forms::DataGridView()); this->Column1 = (gcnew

System::Windows::Forms::DataGridViewTextBoxCol umn());

this->Column2 = (gcnew System::Windows::Forms::DataGridViewTextBoxCol umn());

this->Column3 = (gcnew System::Windows::Forms::DataGridViewTextBoxCol umn());

this->dataGridView2 = (gcnew System::Windows::Forms::DataGridView()); this->Column4 = (gcnew

System::Windows::Forms::DataGridViewTextBoxCol umn());

this->Column5 = (gcnew System::Windows::Forms::DataGridViewTextBoxCol umn());

this->Column6 = (gcnew System::Windows::Forms::DataGridViewTextBoxCol umn());

this->richTextBox3 = (gcnew System::Windows::Forms::RichTextBox()); this->richTextBox4 = (gcnew System::Windows::Forms::RichTextBox()); this->richTextBox5 = (gcnew System::Windows::Forms::RichTextBox()); this->dataGridView3 = (gcnew System::Windows::Forms::DataGridView()); this->Column7 = (gcnew

System::Windows::Forms::DataGridViewTextBoxCol umn());

this->Column8 = (gcnew System::Windows::Forms::DataGridViewTextBoxCol umn());

this->Column9 = (gcnew System::Windows::Forms::DataGridViewTextBoxCol umn());

this->dataGridView4 = (gcnew System::Windows::Forms::DataGridView()); this->Column10 = (gcnew

System::Windows::Forms::DataGridViewTextBoxCol umn());

this->Column11 = (gcnew System::Windows::Forms::DataGridViewTextBoxCol umn());

this->Column12 = (gcnew System::Windows::Forms::DataGridViewTextBoxCol umn());

this->dataGridView5 = (gcnew System::Windows::Forms::DataGridView()); this->Column13 = (gcnew

System::Windows::Forms::DataGridViewTextBoxCol umn());

this->Column14 = (gcnew

System::Windows::Forms::DataGridViewTextBoxCol umn());

this->Column15 = (gcnew System::Windows::Forms::DataGridViewTextBoxCol umn());

this->label1 = (gcnew System::Windows::Forms::Label()); this->label2 = (gcnew System::Windows::Forms::Label()); this->label3 = (gcnew System::Windows::Forms::Label()); this->label4 = (gcnew System::Windows::Forms::Label()); this->label5 = (gcnew System::Windows::Forms::Label()); this->label6 = (gcnew System::Windows::Forms::Label());

(cli::safe\_cast<System::ComponentModel::ISupportIni tialize^>(this->dataGridView1))->BeginInit(); (cli::safe\_cast<System::ComponentModel::ISupportIni tialize^>(this->dataGridView2))->BeginInit(); (cli::safe\_cast<System::ComponentModel::ISupportIni tialize^>(this->dataGridView3))->BeginInit(); (cli::safe\_cast<System::ComponentModel::ISupportIni tialize^>(this->dataGridView4))->BeginInit(); (cli::safe\_cast<System::ComponentModel::ISupportIni tialize^>(this->dataGridView5))->BeginInit();

this->SuspendLayout();

//

// button1

//

this->button1->Location = System::Drawing::Point(166, 407); this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(153, 23); this->button1->TabIndex = 0;

this->button1->Text = L"Выбрать файл";

this->button1->UseVisualStyleBackColor = true; this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1\_Click);

//

// button2

//

this->button2->Location = System::Drawing::Point(728, 407); this->button2->Name = L"button2";

this->button2->Size = System::Drawing::Size(85, 23); this->button2->TabIndex = 1;

this->button2->Text = L"Рефакторинг";

this->button2->UseVisualStyleBackColor = true; this->button2->Click += gcnew System::EventHandler(this, &MyForm::button2\_Click);

//

// richTextBox1

//

this->richTextBox1->Location = System::Drawing::Point(12, 12);

this->richTextBox1->Name = L"richTextBox1"; this->richTextBox1->Size = System::Drawing::Size(474, 389);

this->richTextBox1->TabIndex = 2;

this->richTextBox1->Text = L"";

//

// richTextBox2

//

this->richTextBox2->Location = System::Drawing::Point(510, 12);

this->richTextBox2->Name = L"richTextBox2"; this->richTextBox2->Size = System::Drawing::Size(490, 389);

this->richTextBox2->TabIndex = 3; this->richTextBox2->Text = L"";

//

// openFileDialog1

//

this->openFileDialog1->FileName = L"openFileDialog1";

//

// dataGridView1

//

this->dataGridView1->AllowUserToAddRows = false; this->dataGridView1->AllowUserToDeleteRows = false;

this->dataGridView1->AutoSizeColumnsMode = System::Windows::Forms::DataGridViewAutoSizeCol umnsMode::Fill;

this->dataGridView1->BackgroundColor = System::Drawing::SystemColors::Control; this->dataGridView1->BorderStyle =

System::Windows::Forms::BorderStyle::Fixed3D; this->dataGridView1-

>ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHead ersHeightSizeMode::AutoSize;

this->dataGridView1->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^

>(3) {

this->Column1,

this->Column2, this->Column3

});

this->dataGridView1->Location = System::Drawing::Point(25, 600);

this->dataGridView1->Name = L"dataGridView1"; this->dataGridView1->ReadOnly = true;

this->dataGridView1->RowHeadersVisible = false; this->dataGridView1->Size = System::Drawing::Size(184, 150);

this->dataGridView1->TabIndex = 4;

this->dataGridView1->CellContentClick += gcnew System::Windows::Forms::DataGridViewCellEventHa ndler(this, &MyForm::dataGridView1\_CellContentClick);

//

// Column1

//

this->Column1->FillWeight = 60.91373F; this->Column1->HeaderText = L"№"; this->Column1->Name = L"Column1"; this->Column1->ReadOnly = true;

//

// Column2

//

this->Column2->FillWeight = 119.5432F;

this->Column2->HeaderText = L"Лексема"; this->Column2->Name = L"Column2";

this->Column2->ReadOnly = true;

//

// Column3

//

this->Column3->FillWeight = 119.5432F; this->Column3->HeaderText = L"Код"; this->Column3->Name = L"Column3"; this->Column3->ReadOnly = true;

//

// dataGridView2

//

this->dataGridView2->AllowUserToAddRows = false; this->dataGridView2->AllowUserToDeleteRows = false;

this->dataGridView2->AutoSizeColumnsMode = System::Windows::Forms::DataGridViewAutoSizeCol umnsMode::Fill;

this->dataGridView2->BackgroundColor = System::Drawing::SystemColors::Control; this->dataGridView2->BorderStyle =

System::Windows::Forms::BorderStyle::Fixed3D; this->dataGridView2-

>ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHead ersHeightSizeMode::AutoSize;

this->dataGridView2->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^

>(3) {

this->Column4,

this->Column5, this->Column6

});

this->dataGridView2->Location = System::Drawing::Point(239, 600);

this->dataGridView2->Name = L"dataGridView2"; this->dataGridView2->ReadOnly = true;

this->dataGridView2->RowHeadersVisible = false; this->dataGridView2->Size = System::Drawing::Size(194, 150);

this->dataGridView2->TabIndex = 5;

//

// Column4

//

this->Column4->FillWeight = 60.91373F; this->Column4->HeaderText = L"№"; this->Column4->Name = L"Column4"; this->Column4->ReadOnly = true;

//

// Column5

//

this->Column5->FillWeight = 119.5432F; this->Column5->HeaderText = L"Лексема"; this->Column5->Name = L"Column5";

this->Column5->ReadOnly = true;

//

// Column6

//

this->Column6->FillWeight = 119.5432F; this->Column6->HeaderText = L"Код"; this->Column6->Name = L"Column6"; this->Column6->ReadOnly = true;

//

// richTextBox3

//

this->richTextBox3->Location = System::Drawing::Point(1030, 12);

this->richTextBox3->Name = L"richTextBox3"; this->richTextBox3->Size = System::Drawing::Size(363, 389);

this->richTextBox3->TabIndex = 6; this->richTextBox3->Text = L"";

//

// richTextBox4

//

this->richTextBox4->Location = System::Drawing::Point(25, 462);

this->richTextBox4->Name = L"richTextBox4"; this->richTextBox4->Size = System::Drawing::Size(1368, 49);

this->richTextBox4->TabIndex = 7; this->richTextBox4->Text = L"";

//

// richTextBox5

//

this->richTextBox5->Location = System::Drawing::Point(25, 527);

this->richTextBox5->Name = L"richTextBox5"; this->richTextBox5->Size = System::Drawing::Size(1368, 51);

this->richTextBox5->TabIndex = 8; this->richTextBox5->Text = L"";

//

// dataGridView3

//

this->dataGridView3->AllowUserToAddRows = false; this->dataGridView3->AllowUserToDeleteRows = false;

this->dataGridView3->AutoSizeColumnsMode = System::Windows::Forms::DataGridViewAutoSizeCol umnsMode::Fill;

this->dataGridView3->BackgroundColor = System::Drawing::SystemColors::Control; this->dataGridView3->BorderStyle =

System::Windows::Forms::BorderStyle::Fixed3D; this->dataGridView3-

>ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHead ersHeightSizeMode::AutoSize;

this->dataGridView3->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^

>(3) {

this->Column7,

this->Column8, this->Column9

});

this->dataGridView3->Location = System::Drawing::Point(478, 600);

this->dataGridView3->Name = L"dataGridView3"; this->dataGridView3->ReadOnly = true;

this->dataGridView3->RowHeadersVisible = false; this->dataGridView3->Size = System::Drawing::Size(195, 150);

this->dataGridView3->TabIndex = 9;

//

// Column7

//

this->Column7->FillWeight = 59.39086F; this->Column7->HeaderText = L"№"; this->Column7->Name = L"Column7"; this->Column7->ReadOnly = true;

//

// Column8

//

this->Column8->FillWeight = 120.3046F; this->Column8->HeaderText = L"Лексема"; this->Column8->Name = L"Column8";

this->Column8->ReadOnly = true;

//

// Column9

//

this->Column9->FillWeight = 120.3046F; this->Column9->HeaderText = L"Код"; this->Column9->Name = L"Column9"; this->Column9->ReadOnly = true;

//

// dataGridView4

//

this->dataGridView4->AllowUserToAddRows = false; this->dataGridView4->AllowUserToDeleteRows = false;

this->dataGridView4->AutoSizeColumnsMode = System::Windows::Forms::DataGridViewAutoSizeCol umnsMode::Fill;

this->dataGridView4->BackgroundColor = System::Drawing::SystemColors::Control; this->dataGridView4->BorderStyle =

System::Windows::Forms::BorderStyle::Fixed3D; this->dataGridView4-

>ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHead ersHeightSizeMode::AutoSize;

this->dataGridView4->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^

>(3) {

this->Column10,

this->Column11, this->Column12

});

this->dataGridView4->Location = System::Drawing::Point(728, 600);

this->dataGridView4->Name = L"dataGridView4"; this->dataGridView4->ReadOnly = true;

this->dataGridView4->RowHeadersVisible = false; this->dataGridView4->Size = System::Drawing::Size(195, 150);

this->dataGridView4->TabIndex = 10;

//

// Column10

//

this->Column10->FillWeight = 59.39086F; this->Column10->HeaderText = L"№"; this->Column10->Name = L"Column10"; this->Column10->ReadOnly = true;

//

// Column11

//

this->Column11->FillWeight = 120.3046F;

this->Column11->HeaderText = L"Лексема"; this->Column11->Name = L"Column11"; this->Column11->ReadOnly = true;

//

// Column12

//

this->Column12->FillWeight = 120.3046F; this->Column12->HeaderText = L"Код"; this->Column12->Name = L"Column12"; this->Column12->ReadOnly = true;

//

// dataGridView5

//

this->dataGridView5->AllowUserToAddRows = false; this->dataGridView5->AllowUserToDeleteRows = false;

this->dataGridView5->AutoSizeColumnsMode = System::Windows::Forms::DataGridViewAutoSizeCol umnsMode::Fill;

this->dataGridView5->BackgroundColor = System::Drawing::SystemColors::Control; this->dataGridView5->BorderStyle =

System::Windows::Forms::BorderStyle::Fixed3D; this->dataGridView5-

>ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHead ersHeightSizeMode::AutoSize;

this->dataGridView5->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^

>(3) {

this->Column13,

this->Column14, this->Column15

});

this->dataGridView5->Location = System::Drawing::Point(974, 600);

this->dataGridView5->Name = L"dataGridView5"; this->dataGridView5->ReadOnly = true;

this->dataGridView5->RowHeadersVisible = false; this->dataGridView5->Size = System::Drawing::Size(195, 150);

this->dataGridView5->TabIndex = 11;

//

// Column13

//

this->Column13->FillWeight = 59.39086F; this->Column13->HeaderText = L"№"; this->Column13->Name = L"Column13"; this->Column13->ReadOnly = true;

//

// Column14

//

this->Column14->FillWeight = 120.3046F; this->Column14->HeaderText = L"Лексема"; this->Column14->Name = L"Column14"; this->Column14->ReadOnly = true;

//

// Column15

//

this->Column15->FillWeight = 120.3046F; this->Column15->HeaderText = L"Код"; this->Column15->Name = L"Column15"; this->Column15->ReadOnly = true;

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(236, 584);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(116, 13); this->label1->TabIndex = 12;

this->label1->Text = L"Идентификаторы [20]";

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(22, 581);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(83, 13); this->label2->TabIndex = 13;

this->label2->Text = L"Константы [10]";

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(475, 584);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(113, 13); this->label3->TabIndex = 14;

this->label3->Text = L"Ключевые слова [30]";

//

// label4

//

this->label4->AutoSize = true;

this->label4->Location = System::Drawing::Point(725, 584);

this->label4->Name = L"label4";

this->label4->Size = System::Drawing::Size(0, 13); this->label4->TabIndex = 15;

//

// label5

//

this->label5->AutoSize = true;

this->label5->Location = System::Drawing::Point(971, 584);

this->label5->Name = L"label5";

this->label5->Size = System::Drawing::Size(78, 13); this->label5->TabIndex = 16;

this->label5->Text = L"Операции [50]";

//

// label6

//

this->label6->AutoSize = true;

this->label6->Location = System::Drawing::Point(725, 584);

this->label6->Name = L"label6";

this->label6->Size = System::Drawing::Size(59, 13); this->label6->TabIndex = 17;

this->label6->Text = L"Знаки [40]";

//

// MyForm

//

this->AutoScaleDimensions =

System::Drawing::SizeF(6, 13); this->AutoScaleMode =

System::Windows::Forms::AutoScaleMode::Font; this->ClientSize = System::Drawing::Size(1444, 762); this->Controls->Add(this->label6);

this->Controls->Add(this->label5); this->Controls->Add(this->label4); this->Controls->Add(this->label3); this->Controls->Add(this->label2); this->Controls->Add(this->label1);

this->Controls->Add(this->dataGridView5); this->Controls->Add(this->dataGridView4); this->Controls->Add(this->dataGridView3); this->Controls->Add(this->richTextBox5); this->Controls->Add(this->richTextBox4); this->Controls->Add(this->richTextBox3); this->Controls->Add(this->dataGridView2); this->Controls->Add(this->dataGridView1); this->Controls->Add(this->richTextBox2); this->Controls->Add(this->richTextBox1); this->Controls->Add(this->button2);

this->Controls->Add(this->button1); this->Name = L"MyForm";

this->Text = L"MyForm"; (cli::safe\_cast<System::ComponentModel::ISupportIni tialize^>(this->dataGridView1))->EndInit(); (cli::safe\_cast<System::ComponentModel::ISupportIni tialize^>(this->dataGridView2))->EndInit(); (cli::safe\_cast<System::ComponentModel::ISupportIni tialize^>(this->dataGridView3))->EndInit(); (cli::safe\_cast<System::ComponentModel::ISupportIni tialize^>(this->dataGridView4))->EndInit(); (cli::safe\_cast<System::ComponentModel::ISupportIni tialize^>(this->dataGridView5))->EndInit();

this->ResumeLayout(false); this->PerformLayout();

}

int n;

int same(DataGridView ^str1,string str2) { String ^Str2 = gcnew String(str2.c\_str()); for (int i = 0; i < str1->RowCount; i++) {

if(str1->Rows[i]->Cells[1]->FormattedValue-

>ToString()==Str2){ n = i+1;

return 0;

}

}

return 1;

}

int dot = 0; bool op = true;

bool error=false; void recognizer() { int value = 1;

int s = 0;

for (int i = 0; i < str.length(); i++) { switch (s)

{

case 0:

switch (str[i]) { case '<':

lexeme += str[i];

if (str[i + 1] == '<') { op = false;

}

s = 24;

break; case '>':

lexeme += str[i];

if (str[i + 1] == '<') { op = false;

}

s = 24;

break; case '+':

lexeme += str[i];

if (str[i + 1] == '+') { op = false;

}

s = 24;

break; case '-':

lexeme += str[i];

if (str[i + 1] == '-'|| str[i + 1] == '>') { op = false;

}

s = 24;

break; case '?':

lexeme += str[i]; s = 24;

break; case '/':

lexeme += str[i]; s = 24;

break; case '&':

lexeme += str[i];

if (str[i + 1] == '&') { op = false;

}

s = 24;

break; case '!':

lexeme += str[i];

if (str[i + 1] == '='|| str[i + 1] == '!') { op = false;

}

s = 24;

break; case '~':

lexeme += str[i];

if (str[i + 1] == '~') { op = false;

}

s = 24;

break; case '^':

lexeme += str[i];

if (str[i + 1] == '^') { op = false;

}

s = 24;

break; case '%':

lexeme += str[i];

if (str[i + 1] == '%') { op = false;

}

s = 24;

break; case '|':

lexeme += str[i];

if (str[i + 1] == '|') { op = false;

}

s = 24;

break; case '\*':

lexeme += str[i];

if (str[i + 1] == '\*') { op = false;

}

s = 24;

break; case '=':

lexeme += str[i];

if (str[i + 1] == '=') { op = false;

}

s = 24;

break; case '"':

lexeme += str[i]; s = 23;

break; case '#':

lexeme += str[i];

if (str[i + 1] == '#') { op = false;

}

s = 22;

break; case ':':

lexeme += str[i];

if (str[i + 1] == ':') { op = false;

}

s = 22;

break; case '}':

lexeme += str[i];

if (str[i + 1] == '}') { op = false;

}

s = 22;

break; case '{':

lexeme += str[i];

if (str[i + 1] == '{') { op = false;

}

s = 22;

break; case '(':

lexeme += str[i];

if (str[i + 1] == '(') { op = false;

}

s = 22;

break; case ')':

lexeme += str[i];

if (str[i + 1] == ')') { op = false;

}

s = 22;

break; case ']':

lexeme += str[i];

if (str[i + 1] == ']') { op = false;

}

s = 22;

break; case '[':

lexeme += str[i];

if (str[i + 1] == '[') { op = false;

}

s = 22;

break; case ';':

lexeme += str[i]; s = 22;

break; case '0':

lexeme += str[i]; s = 1;

break; case '1':

lexeme += str[i]; s = 1;

break; case '2':

lexeme += str[i]; s = 1;

break; case '3':

lexeme += str[i]; s = 1;

break; case '4':

lexeme += str[i]; s = 1;

break; case '5':

lexeme += str[i]; s = 1;

break; case '6':

lexeme += str[i]; s = 1;

break; case '7':

lexeme += str[i]; s = 1;

break; case '8':

lexeme += str[i]; s = 1;

break; case '9':

lexeme += str[i]; s = 1;

break; case 'a':

lexeme += str[i];

if (str[i + 1] != 'n'&& str[i + 1] != 's'&& str[i + 1] != 'u')

{

s = 2;

}

else

s = 30;

break; case 'b':

lexeme += str[i];

if (str[i + 1] != 'o' && str[i + 1] != 'r') { s = 2;

}

else

s = 38;

break; case 'c':

lexeme += str[i];

if (str[i + 1] != 'a' && str[i + 1] != 'h' && str[i + 1] != 'l' && str[i + 1] != 'o') {

s = 2;

}

else

s = 11;

break; case 'd':

lexeme += str[i];

if (str[i + 1] != 'e' && str[i + 1] != 'o') { s = 2;

}

else

s = 18;

break; case 'e':

lexeme += str[i];

if (str[i + 1] != 'l' && str[i + 1] != 'n') { s = 2;

}

else

s = 71;

break; case 'f':

lexeme += str[i];

if (str[i + 1] != 'a' && str[i + 1] != 'l' && str[i + 1] != 'o') {

s = 2;

}

else

s = 3;

break; case 'g':

lexeme += str[i];

if (str[i + 1] != 'o') { s = 2;

}

else

s = 84;

break; case 'h':

lexeme += str[i]; s = 2;

break; case 'i':

lexeme += str[i];

if (str[i + 1] != 'n' && str[i + 1] != 'f') { s = 2;

}

else

s = 88;

break; case 'j':

lexeme += str[i]; s = 2;

break; case 'k':

lexeme += str[i]; s = 2;

break; case 'l':

lexeme += str[i];

if (str[i + 1] != 'o') { s = 2;

}

else

s = 97;

break; case 'm':

lexeme += str[i]; s = 2;

break; case 'n':

lexeme += str[i];

if (str[i + 1] != 'a' && str[i + 1] != 'e' && str[i + 1] != 'o') {

s = 2;

}

else

s = 101;

break; case 'o':

lexeme += str[i];

if (str[i + 1] != 'r') { s = 2;

}

else

s = 118;

break; case 'p':

lexeme += str[i];

if (str[i + 1] != 'u' && str[i + 1] != 'r') { s = 2;

}

else

s = 185;

break; case 'q':

lexeme += str[i]; s = 2;

break; case 'r':

lexeme += str[i];

if (str[i + 1] != 'e') { s = 2;

}

else

s = 179;

break; case 's':

lexeme += str[i];

if (str[i + 1] != 'w' && str[i + 1] != 't' && str[i + 1] != 'h') {

s = 2;

}

else

s = 158;

break; case 't':

lexeme += str[i];

if (str[i + 1] != 'h' && str[i + 1] != 'r') { s = 2;

}

else

s = 138;

break; case 'u':

lexeme += str[i];

if (str[i + 1] != 'n' && str[i + 1] != 's') { s = 2;

}

else

s = 149;

break; case 'v':

lexeme += str[i];

if (str[i + 1] != 'i' && str[i + 1] != 'o') { s = 2;

}

else

s = 128;

break; case 'w':

lexeme += str[i];

if (str[i + 1] != 'h') { s = 2;

}

else

s = 123;

break; case 'x':

lexeme += str[i];

if (str[i + 1] != 'o') { s = 2;

}

else

s = 120;

break; case 'y':

lexeme += str[i]; s = 2;

break; case 'z':

lexeme += str[i]; s = 2;

break; case 'A':

lexeme += str[i]; s = 2;

break; case 'B':

lexeme += str[i]; s = 2;

break; case 'C':

lexeme += str[i]; s = 2;

break; case 'D':

lexeme += str[i]; s = 2;

break; case 'E':

lexeme += str[i]; s = 2;

break; case 'F':

lexeme += str[i]; s = 2;

break; case 'G':

lexeme += str[i]; s = 2;

break; case 'H':

lexeme += str[i]; s = 2;

break; case 'I':

lexeme += str[i]; s = 2;

break; case 'J':

lexeme += str[i]; s = 2;

break; case 'K':

lexeme += str[i]; s = 2;

break; case 'L':

lexeme += str[i]; s = 2;

break; case 'M':

lexeme += str[i]; s = 2;

break; case 'N':

lexeme += str[i];

if (str[i + 1] != 'U') { s = 2;

}

else

s = 114;

break; case 'O':

lexeme += str[i];

s = 2;

break; case 'P':

lexeme += str[i]; s = 2;

break; case 'Q':

lexeme += str[i]; s = 2;

break; case 'R':

lexeme += str[i]; s = 2;

break; case 'S':

lexeme += str[i]; s = 2;

break; case 'T':

lexeme += str[i]; s = 2;

break; case 'U':

lexeme += str[i]; s = 2;

break; case 'V':

lexeme += str[i]; s = 2;

break; case 'W':

lexeme += str[i]; s = 2;

break; case 'X':

lexeme += str[i]; s = 2;

break; case 'Y':

lexeme += str[i]; s = 2;

case 'Z':

lexeme += str[i]; s = 2;

break; case '\_':

lexeme += str[i]; s = 2;

default:

s = 0;

break;

}

break; case 1:

switch (str[i]) { case '\t':

s = 0;

lexeme = ""; break;

case '.':

lexeme += str[i]; dot++;

if (dot > 1) {

error = true;

}

s = 27;

break; case 'e':

lexeme += str[i]; dot++;

s = 28;

break; case ',': dot++;

if (dot > 1) { error = true;

}

lexeme += str[i]; s = 27;

break; case '0':

lexeme += str[i]; s = 1;

break; case '1':

lexeme += str[i]; s = 1;

break; case '2':

lexeme += str[i]; s = 1;

break; case '3':

lexeme += str[i]; s = 1;

break; case '4':

lexeme += str[i]; s = 1;

break; case '5':

lexeme += str[i]; s = 1;

break; case '6':

lexeme += str[i]; s = 1;

break; case '7':

lexeme += str[i]; s = 1;

break; case '8':

lexeme += str[i]; s = 1;

break; case '9':

lexeme += str[i]; s = 1;

break; default:

lexeme += str[i]; s = 0;

break;

}

break;

case 2:

switch (str[i]) {

case '-':

lexeme += str[i]; error = true;

s = 2;

break; case '.':

lexeme += str[i]; error = true;

s = 2;

break; case '~':

lexeme += str[i]; error = true;

s = 2;

break; case '$':

lexeme += str[i]; error = true;

s = 2;

break; case '0':

lexeme += str[i]; s = 2;

break; case '1':

lexeme += str[i]; s = 2;

break; case '2':

lexeme += str[i]; s = 2;

break; case '3':

lexeme += str[i]; s = 2;

break; case '4':

lexeme += str[i]; s = 2;

break; case '5':

lexeme += str[i]; s = 2;

break; case '6':

lexeme += str[i]; s = 2;

break; case '7':

lexeme += str[i]; s = 2;

break; case '8':

lexeme += str[i]; s = 2;

break; case '9':

lexeme += str[i]; s = 2;

break; case 'a':

lexeme += str[i]; s = 2;

break; case 'b':

lexeme += str[i]; s = 2;

break; case 'c':

lexeme += str[i]; s = 2;

break; case 'd':

lexeme += str[i]; s = 2;

break; case 'e':

lexeme += str[i]; s = 2;

break; case 'f':

lexeme += str[i]; s = 2;

break; case 'g':

lexeme += str[i]; s = 2;

break; case 'h':

lexeme += str[i]; s = 2;

break; case 'i':

lexeme += str[i]; s = 2;

break; case 'j':

lexeme += str[i]; s = 2;

break; case 'k':

lexeme += str[i]; s = 2;

break; case 'l':

lexeme += str[i]; s = 2;

break; case 'm':

lexeme += str[i]; s = 2;

break; case 'n':

lexeme += str[i]; s = 2;

break; case 'o':

lexeme += str[i]; s = 2;

break; case 'p':

lexeme += str[i]; s = 2;

break;

case 'q':

lexeme += str[i]; s = 2;

break; case 'r':

lexeme += str[i]; s = 2;

break; case 's':

lexeme += str[i]; s = 2;

break; case 't':

lexeme += str[i]; s = 2;

break; case 'u':

lexeme += str[i]; s = 2;

break; case 'v':

lexeme += str[i]; s = 2;

break; case 'w':

lexeme += str[i]; s = 2;

break; case 'x':

lexeme += str[i]; s = 2;

break; case 'y':

lexeme += str[i]; s = 2;

break; case 'z':

lexeme += str[i]; s = 2;

break; case 'A':

lexeme += str[i]; s = 2;

break; case 'B':

lexeme += str[i]; s = 2;

break; case 'C':

lexeme += str[i]; s = 2;

break; case 'D':

lexeme += str[i]; s = 2;

break; case 'E':

lexeme += str[i]; s = 2;

break; case 'F':

lexeme += str[i]; s = 2;

break; case 'G':

lexeme += str[i]; s = 2;

break; case 'H':

lexeme += str[i]; s = 2;

break; case 'I':

lexeme += str[i]; s = 2;

break; case 'J':

lexeme += str[i]; s = 2;

break; case 'K':

lexeme += str[i]; s = 2;

break; case 'L':

lexeme += str[i]; s = 2;

break; case 'M':

lexeme += str[i]; s = 2;

break; case 'N':

lexeme += str[i]; s = 2;

break; case 'O':

lexeme += str[i]; s = 2;

break; case 'P':

lexeme += str[i]; s = 2;

break; case 'Q':

lexeme += str[i]; s = 2;

break; case 'R':

lexeme += str[i]; s = 2;

break; case 'S':

lexeme += str[i]; s = 2;

break; case 'T':

lexeme += str[i]; s = 2;

break; case 'U':

lexeme += str[i]; s = 2;

break; case 'V':

lexeme += str[i];

s = 2;

break; case 'W':

lexeme += str[i]; s = 2;

break; case 'X':

lexeme += str[i]; s = 2;

break; case 'Y':

lexeme += str[i]; s = 2;

case 'Z':

lexeme += str[i]; s = 2;

break; case '\_':

lexeme += str[i]; s = 2;

break; default:

s = 0;

break;

}

break; case 3:

switch (str[i]) { case 'a':

lexeme += str[i]; s = 4;

break; case 'l':

lexeme += str[i]; s = 8;

break; case 'o':

lexeme += str[i]; s = 82;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 4:

switch (str[i]) { case 'l':

lexeme += str[i]; s = 5;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 5:

switch (str[i]) { case 's':

lexeme += str[i]; s = 6;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 6:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 7;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 7:

switch (str[i]) { default:

lexeme += str[i]; s = 7;

error = true; break;

}

break; case 8:

switch (str[i]) { case 'o':

lexeme += str[i]; s = 9;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 9:

switch (str[i]) { case 'a':

lexeme += str[i]; s = 10;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 10:

switch (str[i]) { case 't':

lexeme += str[i]; s = 81;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 11:

switch (str[i]) { case 'a':

lexeme += str[i]; s = 47;

break; case 'h':

lexeme += str[i]; s = 46;

break; case 'l':

lexeme += str[i]; s = 55;

break; case 'o':

lexeme += str[i]; s = 12;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 12:

switch (str[i]) { case 'u':

lexeme += str[i]; s = 13;

break; case 'n':

lexeme += str[i]; s = 15;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 13:

switch (str[i]) { case 't':

lexeme += str[i]; s = 14;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 14:

switch (str[i]) { default:

lexeme += str[i]; s = 14;

error = true; break;

}

break; case 15:

switch (str[i]) { case 's':

lexeme += str[i]; s = 16;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 16:

switch (str[i]) { case 't':

lexeme += str[i]; s = 17;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 17:

switch (str[i]) { default:

lexeme += str[i]; s = 17;

error = true; break;

}

break; case 18:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 19;

break; case 'o':

lexeme += str[i]; s = 66;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 19:

switch (str[i]) { case 'f':

lexeme += str[i]; s = 20;

break; case 'l':

lexeme += str[i]; s = 62;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 20:

switch (str[i]) { case 'a':

lexeme += str[i]; s = 21;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 21:

switch (str[i]) { case 'u':

lexeme += str[i]; s = 59;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 22:

switch (str[i]) { case '#':

lexeme += str[i];

if (str[i + 1] != '#') { op = true;

error = true;

}

s = 22;

break; case ':':

lexeme += str[i];

if (str[i + 1] != ':') { op = true;

}

s = 26;

break; case '}':

lexeme += str[i];

if (str[i + 1] != '}') { op = true;

error = true;

}

s = 22;

break; case '{':

lexeme += str[i];

if (str[i + 1] != '{') { op = true;

error = true;

}

s = 22;

break; case '(':

lexeme += str[i];

if (str[i + 1] != '(') { op = true;

error = true;

}

s = 22;

break; case ')':

lexeme += str[i];

if (str[i + 1] != ')') { op = true;

error = true;

}

s = 22;

break; case ']':

lexeme += str[i];

if (str[i + 1] != ']') { op = true;

error = true;

}

s = 22;

break; case '[':

lexeme += str[i];

if (str[i + 1] != '[') { op = true;

error = true;

}

s = 22;

break; default:

s = 0;

break;

}

break; case 23:

switch (str[i]) {

case '"':

lexeme += str[i]; s = 1;

break; default:

lexeme += str[i]; s = 23;

break;

}

break; case 24:

switch (str[i]) {

case '!':

lexeme += str[i];

if (str[i + 1] != '!') { op = true;

error = true;

}

s = 24;

break; case '~':

lexeme += str[i];

if (str[i + 1] != '~') { op = true;

error = true;

}

s = 24;

break; case '^':

lexeme += str[i];

if (str[i + 1] != '^') { op = true;

error = true;

}

s = 24;

break; case '%':

lexeme += str[i];

if (str[i + 1] != '^') { op = true;

error = true;

}

s = 24;

break; case '<':

lexeme += str[i];

if (str[i + 1] != '<') { op = true;

}

s = 25;

break; case '>':

lexeme += str[i];

if (str[i + 1] != '>') { op = true;

}

s = 25;

break; case '+':

lexeme += str[i];

if (str[i + 1] != '+') { op = true;

}

s = 25;

break; case '-':

lexeme += str[i];

if (str[i + 1] != '-') { op = true;

}

s = 25;

break; case '=':

lexeme += str[i];

if (str[i + 1] != '=') { op = true;

}

s = 25;

break; case '\*':

lexeme += str[i];

if (str[i + 1] != '\*') { op = true;

}

s = 25;

break; case '&':

lexeme += str[i];

if (str[i + 1] != '&') { op = true;

}

s = 25;

break; case '|':

lexeme += str[i];

if (str[i + 1] != '|') { op = true;

}

s = 25;

break; default:

s = 0;

break;

}

break; case 25:

switch (str[i]) { case '<':

lexeme += str[i];

if (str[i + 1] != '<') { op = true;

error = true;

}

s = 25;

break; case '>':

lexeme += str[i];

if (str[i + 1] != '>') { op = true;

error = true;

}

s = 25;

break; case '+':

lexeme += str[i];

if (str[i + 1] != '+') { op = true;

error = true;

}

s = 25;

break; case '-':

lexeme += str[i];

if (str[i + 1] != '-') { op = true;

error = true;

}

s = 25;

break; case '=':

lexeme += str[i];

if (str[i + 1] != '=') { op = true;

error = true;

}

s = 25;

break; case '\*':

lexeme += str[i];

if (str[i + 1] != '\*') { op = true;

error = true;

}

s = 25;

break;

case '&':

lexeme += str[i];

if (str[i + 1] != '&') { op = true;

error = true;

}

s = 25;

break; case '|':

lexeme += str[i];

if (str[i + 1] != '|') { op = true;

error = true;

}

s = 25;

break; default:

s = 0;

break;

}

break; case 26:

switch (str[i]) {

case ':':

lexeme += str[i];

if (str[i + 1] != ':') { op = true;

error = true;

}

s = 26;

break;

default:

s = 0;

break;

}

break; case 27:

switch (str[i]) { case '0':

lexeme += str[i]; s = 1;

break; case '1':

lexeme += str[i]; s = 1;

break; case '2':

lexeme += str[i]; s = 1;

break; case '3':

lexeme += str[i]; s = 1;

break; case '4':

lexeme += str[i]; s = 1;

break; case '5':

lexeme += str[i]; s = 1;

break;

case '6':

lexeme += str[i]; s = 1;

break; case '7':

lexeme += str[i]; s = 1;

break; case '8':

lexeme += str[i]; s = 1;

break; case '9':

lexeme += str[i]; s = 1;

break; default:

lexeme += str[i]; error = true;

s = 27;

break;

}

break; case 28:

switch (str[i]) { case '+':

lexeme += str[i]; s = 29;

break; case '-':

lexeme += str[i]; s = 29;

break; default:

lexeme += str[i]; s = 28;

error = true; break;

}

break; case 29:

switch (str[i]) { case '0':

lexeme += str[i]; s = 1;

break; case '1':

lexeme += str[i]; s = 1;

break; case '2':

lexeme += str[i]; s = 1;

break; case '3':

lexeme += str[i]; s = 1;

break; case '4':

lexeme += str[i]; s = 1;

break; case '5':

lexeme += str[i]; s = 1;

break; case '6':

lexeme += str[i]; s = 1;

break; case '7':

lexeme += str[i]; s = 1;

break; case '8':

lexeme += str[i]; s = 1;

break; case '9':

lexeme += str[i]; s = 1;

break; default:

lexeme += str[i]; s = 29;

error = true; break;

}

break; case 30:

switch (str[i]) { case 'n':

lexeme += str[i]; s = 31;

break; case 's':

lexeme += str[i]; s = 33;

break; case 'u':

lexeme += str[i]; s = 35;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 31:

switch (str[i]) { case 'd':

lexeme += str[i]; s = 32;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 32:

switch (str[i]) { default:

lexeme += str[i]; s = 32;

error = true; break;

}

break; case 33:

switch (str[i]) { case 'm':

lexeme += str[i]; s = 34;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 34:

switch (str[i]) { default:

lexeme += str[i]; s = 34;

error = true; break;

}

break; case 35:

switch (str[i]) { case 't':

lexeme += str[i]; s = 36;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 36:

switch (str[i]) { case 'o':

lexeme += str[i]; s = 37;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 37:

switch (str[i]) { default:

lexeme += str[i]; s = 37;

error = true; break;

}

break; case 38:

switch (str[i]) { case 'o':

lexeme += str[i]; s = 39;

break; case 'r':

lexeme += str[i]; s = 42;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 39:

switch (str[i]) { case 'o':

lexeme += str[i]; s = 40;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 40:

switch (str[i]) { case 'l':

lexeme += str[i]; s = 41;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 41:

switch (str[i]) { default:

lexeme += str[i]; s = 41;

error = true; break;

}

break; case 42:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 43;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 43:

switch (str[i]) { case 'a':

lexeme += str[i]; s = 44;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 44:

switch (str[i]) { case 'k':

lexeme += str[i]; s = 45;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 45:

switch (str[i]) { default:

lexeme += str[i]; s = 45;

error = true; break;

}

break; case 46:

switch (str[i]) { case 'a':

lexeme += str[i]; s = 53;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 47:

switch (str[i]) { case 's':

lexeme += str[i]; s = 48;

break; case 't':

lexeme += str[i]; s = 50;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 48:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 49;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 49:

switch (str[i]) { default:

lexeme += str[i]; s = 49;

error = true; break;

}

break; case 50:

switch (str[i]) { case 'c':

lexeme += str[i]; s = 51;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 51:

switch (str[i]) { case 'h':

lexeme += str[i]; s = 52;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 52:

switch (str[i]) { default:

lexeme += str[i]; s = 52;

error = true; break;

}

break; case 53:

switch (str[i]) { case 'r':

lexeme += str[i]; s = 54;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 54:

switch (str[i]) { default:

lexeme += str[i]; s = 54;

error = true;

break;

}

break; case 55:

switch (str[i]) { case 'a':

lexeme += str[i]; s = 56;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 56:

switch (str[i]) { case 's':

lexeme += str[i]; s = 57;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 57:

switch (str[i]) { case 's':

lexeme += str[i]; s = 58;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 58:

switch (str[i]) { default:

lexeme += str[i]; s = 58;

error = true; break;

}

break; case 59:

switch (str[i]) { case 'l':

lexeme += str[i]; s = 60;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 60:

switch (str[i]) { case 't':

lexeme += str[i]; s = 61;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 61:

switch (str[i]) { default:

lexeme += str[i]; s = 61;

error = true; break;

}

break; case 62:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 63;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 63:

switch (str[i]) { case 't':

lexeme += str[i]; s = 64;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 64:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 65;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 65:

switch (str[i]) { default:

lexeme += str[i]; s = 65;

error = true; break;

}

break;

case 66:

switch (str[i]) { case 'u':

lexeme += str[i]; s = 67;

break; default:

lexeme += str[i]; s = 66;

error = true; break;

}

break; case 67:

switch (str[i]) { case 'b':

lexeme += str[i]; s = 68;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 68:

switch (str[i]) { case 'l':

lexeme += str[i]; s = 69;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 69:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 70;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 70:

switch (str[i]) { default:

lexeme += str[i]; s = 70;

error = true; break;

}

break; case 71:

switch (str[i]) { case 'l':

lexeme += str[i]; s = 72;

break; case 'n':

lexeme += str[i]; s = 75;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 72:

switch (str[i]) { case 's':

lexeme += str[i]; s = 73;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 73:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 74;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 74:

switch (str[i]) { default:

lexeme += str[i]; s = 74;

error = true; break;

}

break; case 75:

switch (str[i]) { case 'u':

lexeme += str[i]; s = 76;

break; case 'd':

lexeme += str[i]; s = 78;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 76:

switch (str[i]) { case 'm':

lexeme += str[i]; s = 77;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 77:

switch (str[i]) { default:

lexeme += str[i]; s = 77;

error = true; break;

}

break; case 78:

switch (str[i]) { case 'l':

lexeme += str[i]; s = 79;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 79:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 80;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 80:

switch (str[i]) { default:

lexeme += str[i]; s = 80;

error = true; break;

}

break; case 81:

switch (str[i]) { default:

lexeme += str[i]; s = 81;

error = true; break;

}

break; case 82:

switch (str[i]) { case 'r':

lexeme += str[i]; s = 83;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 83:

switch (str[i]) { default:

lexeme += str[i]; s = 83;

error = true; break;

}

break; case 84:

switch (str[i]) { case 'o':

lexeme += str[i]; s = 85;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 85:

switch (str[i]) { case 't':

lexeme += str[i]; s = 86;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 86:

switch (str[i]) { case 'o':

lexeme += str[i]; s = 87;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 87:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 87;

break;

}

break;

case 88:

switch (str[i]) { case 'f':

lexeme += str[i]; s = 89;

break; case 'n':

lexeme += str[i]; s = 90;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 89:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 89;

break;

}

break; case 90:

switch (str[i]) { case 't':

lexeme += str[i]; s = 91;

break; case 'c':

lexeme += str[i]; s = 92;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 91:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 91;

break;

}

break; case 92:

switch (str[i]) { case 'l':

lexeme += str[i]; s = 93;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 93:

switch (str[i]) { case 'u':

lexeme += str[i]; s = 94;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 94:

switch (str[i]) { case 'd':

lexeme += str[i]; s = 95;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 95:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 96;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 96:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 96;

break;

}

break; case 97:

switch (str[i]) { case 'o':

lexeme += str[i]; s = 98;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 98:

switch (str[i]) { case 'n':

lexeme += str[i]; s = 99;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 99:

switch (str[i]) { case 'g':

lexeme += str[i]; s = 100;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 100:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 100;

break;

}

break; case 101:

switch (str[i]) { case 'a':

lexeme += str[i]; s = 102;

break; case 'e':

lexeme += str[i]; s = 110;

break; case 'o':

lexeme += str[i]; s = 112;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 102:

switch (str[i]) { case 'm':

lexeme += str[i]; s = 103;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 103:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 104;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 104:

switch (str[i]) { case 's':

lexeme += str[i]; s = 105;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 105:

switch (str[i]) { case 'p':

lexeme += str[i]; s = 106;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 106:

switch (str[i]) { case 'a':

lexeme += str[i]; s = 107;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 107:

switch (str[i]) { case 'c':

lexeme += str[i]; s = 108;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 108:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 109;

break; default:

lexeme += str[i];

s = 2;

break;

}

break; case 109:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 109;

break;

}

break; case 110:

switch (str[i]) { case 'w':

lexeme += str[i]; s = 111;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 111:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 111;

break;

}

break; case 112:

switch (str[i]) { case 't':

lexeme += str[i]; s = 113;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 113:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 113;

break;

}

break; case 114:

switch (str[i]) { case 'U':

lexeme += str[i]; s = 115;

break; default:

lexeme += str[i];

s = 2;

break;

}

break; case 115:

switch (str[i]) { case 'L':

lexeme += str[i]; s = 116;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 116:

switch (str[i]) { case 'L':

lexeme += str[i]; s = 117;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 117:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 117;

break;

}

break; case 118:

switch (str[i]) { case 'r':

lexeme += str[i]; s = 119;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 119:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 119;

break;

}

break; case 120:

switch (str[i]) { case 'o':

lexeme += str[i]; s = 121;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 121:

switch (str[i]) { case 'r':

lexeme += str[i]; s = 122;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 122:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 122;

break;

}

break; case 123:

switch (str[i]) { case 'h':

lexeme += str[i]; s = 124;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 124:

switch (str[i]) { case 'i':

lexeme += str[i]; s = 125;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 125:

switch (str[i]) { case 'l':

lexeme += str[i]; s = 126;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 126:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 127;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 127:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 127;

break;

}

break; case 128:

switch (str[i]) { case 'o':

lexeme += str[i]; s = 129;

break; case 'i':

lexeme += str[i]; s = 132;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 129:

switch (str[i]) { case 'i':

lexeme += str[i]; s = 130;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 130:

switch (str[i]) { case 'd':

lexeme += str[i]; s = 131;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 131:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 131;

break;

}

break; case 132:

switch (str[i]) { case 'r':

lexeme += str[i]; s = 133;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 133:

switch (str[i]) { case 't':

lexeme += str[i]; s = 134;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 134:

switch (str[i]) { case 'u':

lexeme += str[i]; s = 135;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 135:

switch (str[i]) { case 'a':

lexeme += str[i]; s = 136;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 136:

switch (str[i]) { case 'l':

lexeme += str[i]; s = 137;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 137:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 137;

break;

}

break; case 138:

switch (str[i]) { case 'h':

lexeme += str[i]; s = 139;

break; case 'r':

lexeme += str[i]; s = 145;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 139:

switch (str[i]) { case 'r':

lexeme += str[i]; s = 142;

break; case 'i':

lexeme += str[i]; s = 140;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 140:

switch (str[i]) { case 's':

lexeme += str[i]; s = 141;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 141:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 141;

break;

}

break; case 142:

switch (str[i]) { case 'o':

lexeme += str[i]; s = 143;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 143:

switch (str[i]) { case 'w':

lexeme += str[i]; s = 144;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 144:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 144;

break;

}

break; case 145:

switch (str[i]) { case 'u':

lexeme += str[i]; s = 147;

break; case 'y':

lexeme += str[i]; s = 146;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 146:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 146;

break;

}

break; case 147:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 148;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 148:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 148;

break;

}

break; case 149:

switch (str[i]) { case 's':

lexeme += str[i]; s = 150;

break; case 'n':

lexeme += str[i]; s = 154;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 150:

switch (str[i]) { case 'i':

lexeme += str[i]; s = 151;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 151:

switch (str[i]) { case 'n':

lexeme += str[i]; s = 152;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 152:

switch (str[i]) { case 'g':

lexeme += str[i]; s = 153;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 153:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 153;

break;

}

break; case 154:

switch (str[i]) { case 'i':

lexeme += str[i]; s = 155;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 155:

switch (str[i]) { case 'o':

lexeme += str[i]; s = 156;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 156:

switch (str[i]) { case 'n':

lexeme += str[i]; s = 157;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 157:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 157;

break;

}

break;

case 158: switch (str[i]) { case 'w':

lexeme += str[i]; s = 169;

break; case 't':

lexeme += str[i]; s = 159;

break; case 'h':

lexeme += str[i]; s = 175;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 159:

switch (str[i]) { case 'd':

lexeme += str[i]; s = 160;

break; case 'a':

lexeme += str[i]; s = 161;

break; case 'r':

lexeme += str[i]; s = 165;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 160:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 160;

break;

}

break; case 161:

switch (str[i]) { case 't':

lexeme += str[i]; s = 162;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 162:

switch (str[i]) {

case 'i':

lexeme += str[i]; s = 163;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 163:

switch (str[i]) { case 'c':

lexeme += str[i]; s = 164;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 164:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 2;

break;

}

break; case 165:

switch (str[i]) { case 'u':

lexeme += str[i]; s = 166;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 166:

switch (str[i]) { case 'c':

lexeme += str[i]; s = 167;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 167:

switch (str[i]) { case 't':

lexeme += str[i]; s = 168;

break; default:

lexeme += str[i];

s = 2;

break;

}

break; case 168:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 2;

break;

}

break; case 169:

switch (str[i]) { case 'i':

lexeme += str[i]; s = 170;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 170:

switch (str[i]) { case 't':

lexeme += str[i]; s = 171;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 171:

switch (str[i]) { case 'c':

lexeme += str[i]; s = 172;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 172:

switch (str[i]) { case 'h':

lexeme += str[i]; s = 173;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 173:

switch (str[i]) {

default:

lexeme += str[i]; error = true;

s = 2;

break;

}

break; case 175:

switch (str[i]) { case 'o':

lexeme += str[i]; s = 176;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 176:

switch (str[i]) { case 'r':

lexeme += str[i]; s = 177;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 177:

switch (str[i]) { case 't':

lexeme += str[i]; s = 173;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 178:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 178;

break;

}

break; case 179:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 180;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 180:

switch (str[i]) { case 't':

lexeme += str[i]; s = 181;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 181:

switch (str[i]) { case 'u':

lexeme += str[i]; s = 182;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 182:

switch (str[i]) { case 'r':

lexeme += str[i]; s = 183;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 183:

switch (str[i]) { case 'n':

lexeme += str[i]; s = 184;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 184:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 184;

break;

}

break; case 185:

switch (str[i]) { case 'u':

lexeme += str[i]; s = 186;

break; case 'r':

lexeme += str[i]; s = 191;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 186:

switch (str[i]) { case 'b':

lexeme += str[i]; s = 187;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 187:

switch (str[i]) { case 'l':

lexeme += str[i]; s = 188;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 188:

switch (str[i]) { case 'i':

lexeme += str[i]; s = 189;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 189:

switch (str[i]) { case 'c':

lexeme += str[i]; s = 190;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 190:

switch (str[i]) { default:

lexeme += str[i];

error = true; s = 190;

break;

}

break; case 191:

switch (str[i]) { case 'i':

lexeme += str[i]; s = 192;

break; case 'o':

lexeme += str[i]; s = 197;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 192:

switch (str[i]) { case 'v':

lexeme += str[i]; s = 193;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 193:

switch (str[i]) { case 'a':

lexeme += str[i]; s = 194;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 194:

switch (str[i]) { case 't':

lexeme += str[i]; s = 195;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 195:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 196;

break;

default:

lexeme += str[i]; s = 2;

break;

}

break; case 196:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 196;

break;

}

break; case 197:

switch (str[i]) { case 't':

lexeme += str[i]; s = 198;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 198:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 199;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 199:

switch (str[i]) { case 'c':

lexeme += str[i]; s = 200;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 200:

switch (str[i]) { case 't':

lexeme += str[i]; s = 201;

break; default:

lexeme += str[i]; s = 2;

break;

}

break;

case 201:

switch (str[i]) { case 'e':

lexeme += str[i]; s = 202;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 202:

switch (str[i]) { case 'd':

lexeme += str[i]; s = 203;

break; default:

lexeme += str[i]; s = 2;

break;

}

break; case 203:

switch (str[i]) { default:

lexeme += str[i]; error = true;

s = 203;

break;

}

break;

}

if (str[i] == '\n') { value++;

}

if (s == 24 && op && !error) {

if (same(dataGridView5, lexeme)) { dataGridView5->Rows->Add();

dataGridView5->Rows[dataGridView5->RowCount - 1]->Cells[0]->Value = dataGridView5->RowCount; dataGridView5->Rows[dataGridView5->RowCount - 1]->Cells[1]->Value = gcnew String(lexeme.c\_str()); dataGridView5->Rows[dataGridView5->RowCount - 1]->Cells[2]->Value = gcnew String(lexeme.c\_str()); richTextBox4->Text += "[50," + dataGridView5-

>Rows[dataGridView5->RowCount - 1]->Cells[0]-

>Value + "]";

richTextBox5->Text += gcnew String(lexeme.c\_str()); lexeme = "";

s = 0;

}

else {

richTextBox4->Text += "[50," + n + "]"; richTextBox5->Text += gcnew String(lexeme.c\_str()); s = 0;

lexeme = "";

}

}

else if (s == 24 && error && op) { q = lexeme;

richTextBox3->Text += "Ошибка в операции " + gcnew String(q.c\_str()) + " в строке " + value + "\n"; lexeme = "";

error = false; s = 0;

}

else if (s == 25 && op && !error) { if (same(dataGridView5, lexeme)) { dataGridView5->Rows->Add();

dataGridView5->Rows[dataGridView5->RowCount - 1]->Cells[0]->Value = dataGridView5->RowCount; dataGridView5->Rows[dataGridView5->RowCount - 1]->Cells[1]->Value = gcnew String(lexeme.c\_str()); dataGridView5->Rows[dataGridView5->RowCount - 1]->Cells[2]->Value = gcnew String(lexeme.c\_str()); richTextBox4->Text += "[50," + dataGridView5-

>Rows[dataGridView5->RowCount - 1]->Cells[0]-

>Value + "]";

richTextBox5->Text += gcnew String(lexeme.c\_str()); lexeme = "";

s = 0;

}

else {

richTextBox4->Text += "[50," + n + "]"; richTextBox5->Text += gcnew String(lexeme.c\_str()); s = 0;

lexeme = "";

}

}

else if(s==25&&error) { q = lexeme;

richTextBox3->Text += "Ошибка в операции " + gcnew String(q.c\_str()) + " в строке " + value + "\n"; lexeme = "";

error = false; s = 0;

}

if (s == 22 && op && !error) {

if (same(dataGridView4, lexeme)) { dataGridView4->Rows->Add();

dataGridView4->Rows[dataGridView4->RowCount - 1]->Cells[0]->Value = dataGridView4->RowCount; dataGridView4->Rows[dataGridView4->RowCount - 1]->Cells[1]->Value = gcnew String(lexeme.c\_str()); dataGridView4->Rows[dataGridView4->RowCount - 1]->Cells[2]->Value = gcnew String(lexeme.c\_str()); richTextBox4->Text += "[40," + dataGridView4-

>Rows[dataGridView4->RowCount - 1]->Cells[0]-

>Value + "]";

richTextBox5->Text += gcnew String(lexeme.c\_str()); lexeme = "";

s = 0;

}

else {

richTextBox4->Text += "[40," + n + "]"; richTextBox5->Text += gcnew String(lexeme.c\_str()); s = 0;

lexeme = "";

}

}

else if (s == 22 && error && op) { q = lexeme;

richTextBox3->Text += "Ошибка в знаке " + gcnew String(q.c\_str()) + " в строке " + value + "\n"; lexeme = "";

error = false; s = 0;

}

else if (s == 26 && op && !error) { if (same(dataGridView4, lexeme)) { dataGridView4->Rows->Add();

dataGridView4->Rows[dataGridView4->RowCount - 1]->Cells[0]->Value = dataGridView4->RowCount; dataGridView4->Rows[dataGridView4->RowCount - 1]->Cells[1]->Value = gcnew String(lexeme.c\_str()); dataGridView4->Rows[dataGridView4->RowCount - 1]->Cells[2]->Value = gcnew String(lexeme.c\_str()); richTextBox4->Text += "[40," + dataGridView4-

>Rows[dataGridView4->RowCount - 1]->Cells[0]-

>Value + "]";

richTextBox5->Text += gcnew String(lexeme.c\_str()); lexeme = "";

s = 0;

}

else {

richTextBox4->Text += "[40," + n + "]"; richTextBox5->Text += gcnew String(lexeme.c\_str()); s = 0;

lexeme = "";

}

}

else if (s == 26 && error && op) { q = lexeme;

richTextBox3->Text += "Ошибка в знаке " + gcnew String(q.c\_str()) + " в строке " + value + "\n"; lexeme = "";

error = false; s = 0;

}

if (str[i + 1] == '-'|| str[i + 1] == '{' || str[i + 1] == '}' || str[i + 1] == '!'|| str[i + 1] == '=' || str[i + 1] == '\n' || str[i

+ 1] == ' ' || str[i + 1] == ';' || str[i + 1] == '"'|| str[i + 1]

== '[' || str[i + 1] == ']' || str[i + 1] == '(' || str[i + 1] == ')'

|| str[i + 1] == '>' || str[i + 1] == '<' || str[i + 1] == NULL || str[i + 1] == '|' || str[i + 1] == '&') {

if (s == 1 && !error) {

if (same(dataGridView1, lexeme)) { if(str[i]=='"'){

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->RowCount - 1]->Cells[0]->Value = dataGridView1->RowCount; dataGridView1->Rows[dataGridView1->RowCount - 1]->Cells[1]->Value = gcnew String(lexeme.c\_str()); dataGridView1->Rows[dataGridView1->RowCount - 1]->Cells[2]->Value = "text";

richTextBox4->Text += "[10," + dataGridView1-

>Rows[dataGridView1->RowCount - 1]->Cells[0]-

>Value + "]";

richTextBox5->Text += " text "; lexeme = "";

s = 0;

dot = 0;

}

else {

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->RowCount - 1]->Cells[0]->Value = dataGridView1->RowCount; dataGridView1->Rows[dataGridView1->RowCount - 1]->Cells[1]->Value = gcnew String(lexeme.c\_str()); dataGridView1->Rows[dataGridView1->RowCount - 1]->Cells[2]->Value = "digit";

richTextBox4->Text += "[10," + dataGridView1-

>Rows[dataGridView1->RowCount - 1]->Cells[0]-

>Value + "]";

richTextBox5->Text += " digit "; lexeme = "";

s = 0;

dot = 0;

}

}

else {richTextBox4->Text += "[10," + n + "]"; richTextBox5->Text += " digit ";

s = 0;

dot = 0; lexeme = ""; }

}

else if (s == 1 && error) { q = lexeme;

richTextBox3->Text += "Ошибка в числе " + gcnew String(q.c\_str()) + " в строке " + value + "\n"; lexeme = "";

dot = 0; error = false; s = 0;

}

else if (s == 27 && error) { q = lexeme;

richTextBox3->Text += "Ошибка в числе " + gcnew String(q.c\_str()) + " в строке " + value + "\n"; lexeme = "";

dot = 0; error = false; s = 0;

}

else if (s == 28 && error) { q = lexeme;

richTextBox3->Text += "Ошибка в числе " + gcnew String(q.c\_str()) + " в строке " + value + "\n"; lexeme = "";

dot = 0; error = false; s = 0;

}

else if (s == 29 && error) { q = lexeme;

richTextBox3->Text += "Ошибка в числе " + gcnew String(q.c\_str()) + " в строке " + value + "\n"; lexeme = "";

dot = 0; error = false; s = 0;

}

else if (s == 2) {

if (error|| lexeme[0]=='0'|| lexeme[0] == '1'||lexeme[0]

== '2' || lexeme[0] == '3' || lexeme[0] == '4' || lexeme[0]

== '5' || lexeme[0] == '6' || lexeme[0] == '7' || lexeme[0]

== '8' || lexeme[0] == '9') {

q = lexeme;

richTextBox3->Text += "Ошибка в идентификаторе " + gcnew String(q.c\_str())+ " в строке " + value

+"\n"; lexeme = ""; error = false ; s = 0;

}else

if (same(dataGridView2, lexeme)) { dataGridView2->Rows->Add();

dataGridView2->Rows[dataGridView2->RowCount - 1]->Cells[0]->Value = dataGridView2->RowCount; dataGridView2->Rows[dataGridView2->RowCount - 1]->Cells[1]->Value = gcnew String(lexeme.c\_str()); dataGridView2->Rows[dataGridView2->RowCount - 1]->Cells[2]->Value = "id";

richTextBox4->Text += "[20," + dataGridView2-

>Rows[dataGridView2->RowCount - 1]->Cells[0]-

>Value + "]";

richTextBox5->Text += " id "; lexeme = "";

s = 0;

}

else {

richTextBox4->Text += "[20," + n + "]"; richTextBox5->Text += " id ";

s = 0;

lexeme = "";

}

}

else if (s == 3|| s == 7 || s == 14 || s == 17 || s == 32||s==34||s == 37 || s == 41 || s == 45 || s == 49 || s ==

52 || s == 54 || s == 58|| s == 61|| s == 65|| s == 66 || s

== 70 || s == 74 || s == 77 || s == 80||s == 81 || s == 83 || s == 87 || s == 89 || s == 91 || s == 96 || s == 100 || s == 109 || s == 111 || s == 113 || s == 117 || s == 119 || s ==

122 || s == 127 || s == 131 || s == 137 || s == 141 || s ==

144 || s == 146 || s == 148 || s == 153 || s == 157 || s ==

160 || s == 164 || s == 168 || s == 173 || s == 178 || s ==

184 || s == 190 || s == 196 || s == 203) { if (!error) {

if (same(dataGridView3, lexeme)) { dataGridView3->Rows->Add();

dataGridView3->Rows[dataGridView3->RowCount - 1]->Cells[0]->Value = dataGridView3->RowCount; dataGridView3->Rows[dataGridView3->RowCount - 1]->Cells[1]->Value = gcnew String(lexeme.c\_str()); dataGridView3->Rows[dataGridView3->RowCount - 1]->Cells[2]->Value = gcnew String(lexeme.c\_str()); richTextBox4->Text += "[30," + dataGridView3-

>Rows[dataGridView3->RowCount - 1]->Cells[0]-

>Value + "]";

richTextBox5->Text += " " + gcnew String(lexeme.c\_str()) + " ";

lexeme = ""; s = 0;

}

else {

richTextBox4->Text += "[30," + n + "]"; richTextBox5->Text += " " + gcnew String(lexeme.c\_str()) + " ";

s = 0;

lexeme = "";

}

}

else {

q = lexeme;

richTextBox3->Text += "Ошибка в ключевом слове "

+ gcnew String(q.c\_str()) + " в строке " + value + "\n"; lexeme = "";

error = false; s = 0;

}

}

}

}

}

#pragma endregion

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

String^ FileName = " ";

if (openFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) { FileName = openFileDialog1->FileName;

}

try {

StreamReader^ file = File::OpenText(FileName); richTextBox1->Text = file->ReadToEnd();

}

catch (Exception^ e)

{

MessageBox::Show(this, "Файл не был открыт", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

richTextBox2->Clear(); // Очищает richTextBox2 richTextBox3->Clear();

richTextBox4->Clear(); richTextBox5->Clear();

String^ code = richTextBox1->Text; MarshalString(code, str); //Преобразование code в str ref();// функция очищения кода

str.insert(0, "1\t");//Подсчёт строк int n = 1;

for (int i = 0; i < str.length(); i++) { if (str[i] == '\n') {

n++;

str.insert(i + 1, to\_string(n)); if (n < 10) {

str.insert(i + 2, "\t");

}

else str.insert(i + 3, "\t");

}

}

richTextBox2->Text += gcnew String(str.c\_str());//Вывод очищенного кода dataGridView1->Rows->Clear(); dataGridView2->Rows->Clear(); dataGridView3->Rows->Clear(); dataGridView4->Rows->Clear(); dataGridView5->Rows->Clear();

recognizer();

}

private: System::Void dataGridView1\_CellContentClick(System::Object^ sender, System::Windows::Forms::DataGridViewCellEventArgs^ e) {

}

};

}

Приложение 3 Руководство пользователя

Общие сведения о программе

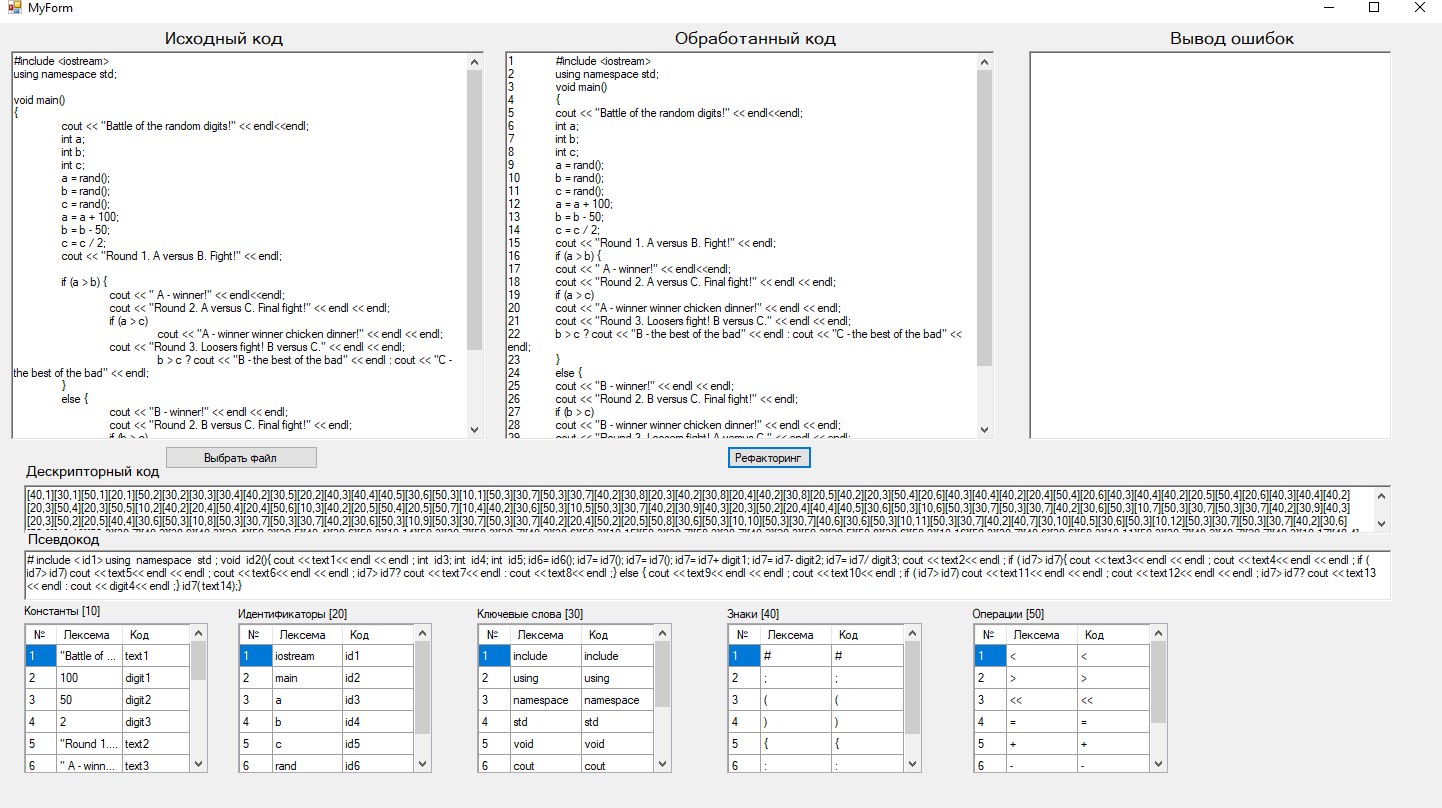
Исполнительный файл программы называется «Проект1.exe». Ввод текста программы производится в поле «Исходный код». Его можно ввести вручную или нажать кнопку «Выбрать файл» и выбрать текстовый файл с кодом. После нажатия на кнопку «Рефакторинг», программа удаляет комментарии и лишние символы из текста программы, выделяет лексемы, выводит дескрипторный код и псевдокод в соответствующие поля, а также, при наличии ошибок, выводит их в поле «Вывод ошибок». (рис. П3.1).

Рис. П3.1. Результат работы программы

* 1. Описание установки

Программа не требует дополнительной установки.

* 1. Описание запуска

Для запуска приложения необходимо запустить файл

«Проект1.exe».

* 1. Описание интерфейса программы

Таблица П3.1

Компоненты интерфейса

|  |  |
| --- | --- |
| Название компонента | Назначение |
| Текстовое поле «Исходный код» | Текстовое поле, предназначенное для  ввода исходного кода. |
| Текстовое поле «Обработанный код» | Текстовое поле, в которое выводится исходный текст после обработки (удаления табуляции, незначащих  пробелов, двойных переносов строк, комментариев разных типов). |
| Текстовое поле «Вывод ошибок» | Поле, в которое выводится сообщение  о возникшей ошибке и ее тип. |
| Кнопка «Рефакторинг» | Кнопка для запуска лексического  анализа. |
| Таблица «Константы», таблица  «Идентификаторы», таблица  «Ключевые слова», таблица «Знаки», таблица «Операции» | Таблицы, предназначенные для вывода информации о распознанной лексеме, где в столбцах указывается класс лексемы, код лексемы, ее наименование и представление  лексемы в псевдокоде. |
| Текстовое поле «Дескрипторный код» | Текстовое поле, в котором выводится код, состоящий из пар (номер класса  лексемы; код лексемы) |
| Текстовое поле «Псевдокод» | Текстовое поле, в котором представлена программа на  псевдокоде. |

Приложение 4 Граф конечного автомата Граф конечного автомата представлен на рисунке П.4.1.

63

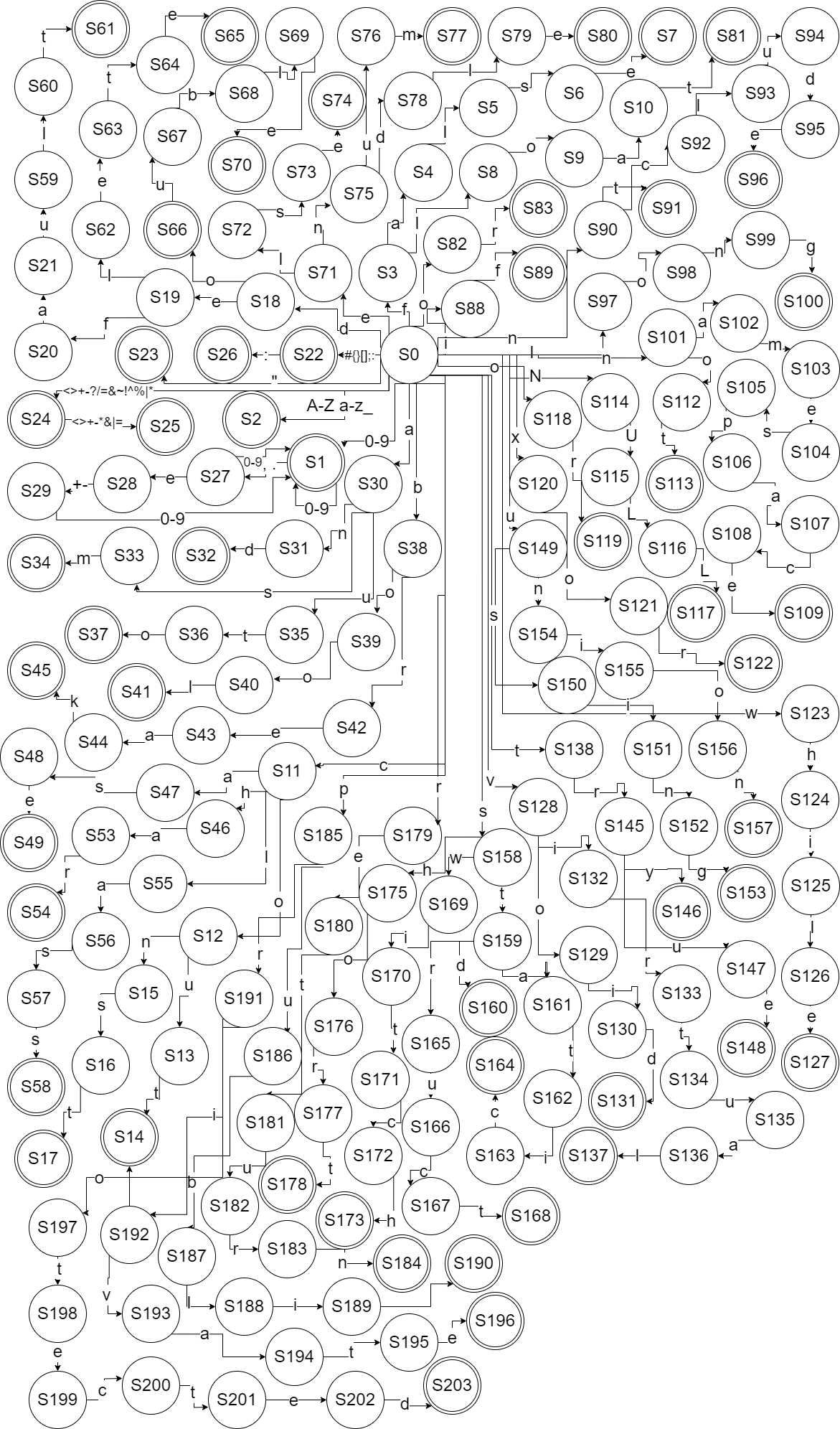


Рис. П4.1. Граф конечного автомата