МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

|  |  |
| --- | --- |
| Институт (факультет) | Информационных технологий |
| Кафедра | Математическое и программное обеспечение ЭВМ |

КУРСОВАЯ РАБОТА

|  |
| --- |
| по дисциплине Теория автоматов и формальных языков программирования |

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

|  |
| --- |
| Выполнил студент группы |
| 1ПИб-02-1оп-22 |
| направления подготовки (специальности) |
| 09.03.04, Программная инженерия |
| шифр, наименование |
|  |
| Маслов Владислав Андреевич |
| фамилия, имя, отчество |

|  |  |
| --- | --- |
| Руководитель |  |
| Ганичева Оксана Георгиевна |  |
| фамилия, имя, отчество |  |
| Доцент |  |
| должность  Пышницкий Константин Михайлович |  |
| фамилия, имя, отчество |  |
| Старший Преподаватель |  |
| должность |  |

|  |
| --- |
| Дата представления работы |
| «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 г. |
|  |
| Заключение о допуске к защите |
|  |
|  |
|  |
|  |
| Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| количество баллов |
| Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |

Аннотация

В данной курсовой работе рассматривается тема разработка алгоритмического обеспечения и построение лексического анализатора компилятора. В работе описывается процесс проектирования и реализации программы. Основной целью работы является разработка приложения, которое будет принимать код некоторой программы и обрабатывать его подобно компилятору. Результат работы может быть использован в образовательных целях.

Оглавление

[Введение 5](#_Toc169126667)

[1. Описание предметной области 6](#_Toc169126668)

[1.1. Лексический анализатор 6](#_Toc169126669)

[1.2 Синтаксический анализатор 9](#_Toc169126670)

[2. Основная часть 11](#_Toc169126671)

[2.1. Описание класса автоматных грамматик в классификации Хомского 11](#_Toc169126672)

[2.2. Связь лексического анализа с автоматными грамматиками и конечными автоматами 15](#_Toc169126673)

[2.3. Описание конечного автомата, лежащего в основе лексического анализатора 17](#_Toc169126674)

[2.4 Построение КС-грамматики, порождения и дерева вывода 41](#_Toc169126675)

[2.4.1 Связь между КС-грамматиками и синтаксическим анализом. Порождение и дерево вывода 41](#_Toc169126676)

[2.4.2 Построение КС-грамматики 43](#_Toc169126677)

[3. Логическое проектирование 52](#_Toc169126678)

[3.1 Разработка блок-схем алгоритма работы программы 52](#_Toc169126679)

[3.2 Оценка сложности алгоритма 58](#_Toc169126680)

[3.3 Словесное описание работы алгоритма на примере разбора тестовых входных последовательностей 59](#_Toc169126681)

[4. Физическое проектирование программного обеспечения 60](#_Toc169126682)

[4.1 Физическое проектирование 60](#_Toc169126683)

[4.1.1 Выбор структур данных 61](#_Toc169126684)

[4.1.2 Спецификация функций 62](#_Toc169126685)

[5. Проектирование интерфейса 63](#_Toc169126686)

[6. Тестирование программного обеспечения 65](#_Toc169126687)

[7. Заключение 70](#_Toc169126688)

[8. Список литературы 71](#_Toc169126689)

[Приложение 1. Техническое задание 72](#_Toc169126690)

[Приложение 2. Текст программы 78](#_Toc169126691)

[Приложение 3. Руководство пользователя 112](#_Toc169126692)

# Введение

В современном мире информационных технологий разработка программного обеспечения играет ключевую роль. Одним из важнейших этапов создания любого программного продукта является процесс компиляции, который преобразует исходный код программы в машинный код, понятный компьютеру.

Работа компилятора делится на следующие этапы:

1. Лексический анализ.

2. Синтаксический анализ.

3. Семантический анализ.

4. Промежуточное представление.

5. Оптимизация кода.

6. Генерация кода.

7. Таблицы символов.

В основе этого процесса лежит лексический анализ – первый этап анализа исходного кода, который отвечает за разбиение текста программы на отдельные лексемы и определение их типа.

Цель данной курсовой работы – рассмотреть создание компилятора лексического анализатора, разбитого на несколько этапов. Изучение данной темы позволит улучшить свои познания в работе основных принципов компилятора, его важность в разработке программного обеспечения и потенциал для оптимизации процесса компиляции.

# Описание предметной области

В этой части будут рассмотрены определения, назначения и функции лексического и синтаксического анализов.

# 1.1. Лексический анализатор

Лексический анализ, также известный как сканирование, - это первый этап работы компилятора, который отвечает за разбиение исходного кода программы на отдельные лексемы и определение их типа. На вход лексического анализатора поступает текст исходной программы. Результатом работы лексического анализатора является перечень всех найденных в тексте исходной программы лексем. Эти данные можно представить в виде таблицы, называемой таблицей лексем. Каждой лексеме в таблице лексем будет соответствовать некий уникальный код, зависящий от типа лексемы, и дополнительной информации о ней.

Лексемы – это минимальные значимые единицы языка программирования, такие как идентификаторы, ключевые слова, операторы, числа, строки и т.п. Например, при анализе строки "int \_z15a2 += 1;" можно получить соответствующие пары данных, которые приведены в таблице 1.

Таблица 1

Классы лексем

|  |  |
| --- | --- |
| Класс | Лексема |
| Ключевое слово | int |
| Идентификатор | \_z15a2 |
| Оператор | += |
| Числовая константа | 1 |
| Разделитель | ; |

Далее более подробно разберем процесс работы лексического анализатора.

Процесс лексического анализа:

1.Сканирование исходного кода: Лексический анализатор последовательно считывает символы исходного кода программы.

2.Формирование лексем: Анализатор группирует символы в лексемы согласно правилам языка программирования.

3.Определение типа лексемы: После формирования лексемы анализатор определяет её тип. Например, "if" будет определен как ключевое слово, "123" - как целое число, а "abc123" - как идентификатор.

4. Передача информации: Лексический анализатор передает информацию о каждой распознанной лексеме следующему этапу компиляции - синтаксическому анализу.

С теоретической точки зрения лексический анализатор не является обязательной, необходимой частью компилятора. Так как, его функции могут выполняться на этапе синтаксического разбора. Использование же лексического анализатора важно из-за следующих причин:

1) Применение лексического анализатора сокращает объём информации поступающий на обработку следующему этапу компиляции.

2) Качественный лексический анализатор позволяет ускорить процесс компиляции за счет быстрого распознавания лексем.

3) Лексический анализатор улучшает гибкость компиляции. Лексический анализатор отделяет сложный синтаксический анализатор от работы с самим текстом исходной программы. Структура синтаксического анализатора значительно сложнее и может быть разной в зависимости от вычислительной системы. Значит такая конструкция компилятора позволяет для перехода на другую вычислительную систему поменять только лексический анализатор.

4) Обнаружение ошибок на этапе лексического анализа помогает избежать ошибок на последующих этапах и повысить качество итогового кода.

Основные задачи лексического анализатора:

1. Удаление пробелов и комментариев: Лексический анализатор игнорирует пробелы и комментарии, которые не несут смысловой нагрузки.
2. Выделение лексем и создание их классификации.
3. Обработка ошибок: Анализатор должен уметь обнаруживать и сообщать об ошибках в исходном коде, например, о недопустимых символах или неправильно сформированных лексемах.

Построение эффективного лексического анализатора основывается на ряде принципов, которые помогают обеспечить корректное и быстрое распознавание лексем в исходном коде программы.

Этап первый выбор модели и языка программирования.

Лексический анализатор может быть реализован на различных языках программирования, который поддерживает возможности ввода-вывода, позволяющие считывать входные данные;

Конечные автоматы. Это распространенный подход к построению лексических анализаторов. КА представляют собой абстрактные модели, которые могут находиться в конечном числе состояний и переходить между ними в зависимости от входных символов. Для каждой лексемы или группы лексем строится отдельный КА.

Регулярные выражения. Они используются для описания шаблонов лексем. Регулярные выражения позволяют компактно задать правила формирования лексем с помощью специальных символов и метасимволов. В этом случае лексический анализатор строится на основе преобразования регулярных выражений в конечные автоматы.

Этап второй определение структуры лексем и их классификация:

Анализ алфавита языка: необходимо определить набор допустимых символов, из которых могут состоять лексемы.

Выделение классов лексем и описание структуры каждой категории лексем: Лексемы группируются по типам, например, идентификаторы, ключевые слова, операторы, числа, строки и т.п., а также для каждой категории определяются правила формирования лексем, например, с помощью регулярных выражений.

Этап третий обработка ошибок.

Лексический анализатор должен уметь обнаруживать ошибки в исходном коде, например, недопустимые символы или неправильно сформированные лексемы. После чего анализатор должен предоставлять информативные сообщения об ошибках, указывающие на место и тип ошибки.

Построение лексического анализатора – это сложная задача, требующая глубокого понимания теории языков программирования и методов их анализа. Однако, при правильном выборе модели и использовании эффективных алгоритмов можно создать качественный анализатор, который станет основой для успешной работы компилятора.

# 1.2 Синтаксический анализатор

Синтаксический анализ, также известный как разбор, является центральным этапом работы компилятора, играющим ключевую роль в процессе преобразования исходного кода в машинный код. Без выполнения этой фазы компиляция не имеет смысла, поскольку невозможно определить, соответствует ли исходная программа правилам языка программирования.

Лексический анализ, предшествующий синтаксическому анализу, хоть и является важным этапом, не является обязательным. Он выполняет задачу разбиения исходного кода на лексемы (слова) и присваивает каждому слову его тип. Эта информация используется в дальнейших этапах компиляции. Однако, если лексический анализатор выполняет свои функции, то синтаксический анализатор может сосредоточиться на более сложных задачах, таких как проверка грамматических правил и построение дерева разбора.

Синтаксический анализ начинается с получения таблицы лексем от лексического анализатора. Затем, анализатор изучает тип каждой лексемы, проверяя, соответствует ли последовательность лексем грамматическим правилам языка программирования. Информация о других характеристиках лексем используется на более поздних этапах компиляции, таких как семантический анализ, подготовка и генерация кода для итоговой программы.

Основная задача синтаксического анализатора - построить дерево разбора, представляющее синтаксическую структуру программы. Дерево разбора — это иерархическая структура данных, отражающая взаимосвязи элементов программы. Например, ветви дерева могут представлять операторы, а листья — переменные или константы. Дерево разбора является ключевым элементом для последующих этапов компиляции, таких как семантический анализ и генерация кода.

Помимо построения дерева разбора, синтаксический анализатор также определяет, соответствует ли структура программы правилам языка. Это позволяет выявить синтаксические ошибки, такие как пропущенные скобки, неправильный порядок операторов или несогласованные типы данных. Например, если в программе пропущена скобка, синтаксический анализатор обнаружит эту ошибку и сообщит программисту.

Процесс синтаксического анализа можно разделить на несколько этапов:

1. Получение лексем: Данные, полученные от лексического анализатора, передаются на вход синтаксическому анализатору, который анализирует их согласно грамматике входного языка.

2. Проверка грамматических правил: Синтаксический анализатор использует формальную грамматику языка программирования для проверки соответствия последовательности лексем заданным правилам. 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 (контекстно-свободные грамматики), 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.

3. Построение дерева разбора: По мере проверки грамматических правил, анализатор строит дерево разбора, которое отражает синтаксическую структуру программы.

4. Обработка ошибок: Синтаксический анализатор должен уметь обнаруживать и сообщать о синтаксических ошибках в исходном коде, например, о пропущенных скобках или несогласованных типах данных.

Синтаксический анализ является важной частью процесса компиляции, которая обеспечивает проверку корректности программы и построение структуры данных, необходимой для последующих этапов анализа и генерации кода.

Синтаксический анализ является важной частью процесса компиляции, которая обеспечивает проверку корректности программы и построение структуры данных, необходимой для последующих этапов анализа и генерации кода.

# Основная часть

# 2.1. Описание класса автоматных грамматик в классификации Хомского

В области исследования формальных языков существует четыре основных типа грамматических систем, каждый из которых определяет свой уникальный класс языков. Эти типы грамматик определяются через систематическое усиление ограничений, положенных на составление правил грамматики. 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.

Грамматики типа 0 называют грамматиками общего вида. Они не имеют никаких ограничений на правила порождения. Эти грамматики порождают естественные языки.

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, являются самыми сложными по структуре.

Практического применения грамматики, относящиеся только к типу 0, не имеют.

Тип 1: контекстно-зависимые (КЗ) и не укорачивающие грамматики

Правила структуры контекстно-зависимых грамматик предполагают, что в предложениях определенного языка один и тот же нетерминальный символ может быть заменен различными символьными цепочками в зависимости от контекста. Именно поэтому такие грамматики именуются контекстно-зависимыми. В отличие от них, у неукорачивающих грамматик структура правил такова, что любая цепочка символов при построении предложений заданного языка может быть заменена цепочкой символов равной длины или длиннее. Именно поэтому они и называются "неукорачивающими". Было доказано, что эти два класса грамматик эквивалентны. В разработке компиляторов такие грамматики не используются, поскольку языки программирования, с которыми работают компиляторы, обладают более простой структурой и могут быть описаны с помощью грамматик другого типа.

Грамматики типа 1 значительно удобнее на практике, чем грамматики типа 0, поскольку в левой части правила заменяется всегда один нетерминальный символ, который можно связать с некоторым синтаксическим понятием, в то время как в грамматике типа 0 можно заменять сразу несколько символов, в том числе и терминальных.

Грамматика типа 2: контекстно-свободные (КС) грамматики

Грамматики типа 2 называют контекстно-свободными и бесконтекстными грамматиками (КС-грамматики или Б-грамматики).

Всегда присутствует, по меньшей мере, один символ в правой части. Отличие заключается в том, что левая часть правил должна содержать только один нетерминальный символ. Такие грамматики иногда называются не укорачивающими контекстно-свободными (НКС) грамматиками, так как в правой части правил всегда должен быть как минимум один символ. Существует также почти эквивалентный им класс грамматик – укорачивающие контекстно-свободные (УКС) грамматики. Разница между этими двумя классами грамматик заключается лишь в том, что в УКС -грамматиках в правой части правил может присутствовать пустая цепочка, а в НКС -грамматиках – нет.

КС-грамматики широко используются при описании синтаксических конструкций языков программирования. Синтаксис большинства известных языков программирования основан именно на КС-грамматиках.

Грамматика 3 типа: регулярные грамматики

В правой части правил такого рода грамматики допускается только один нетерминальный символ, который должен быть расположен с одной и той же стороны от цепочки терминалов в каждом правиле грамматики. Требования к левой части правил совпадают с предыдущим типом.

Существуют два эквивалентных класса грамматик, относящихся к регулярному типу: леволинейные и праволинейные. Оба этих класса грамматик эквивалентны и относятся к регулярным грамматикам. Леволинейные грамматики G = T, N, P, S ; V = N ∪ 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 ∪ T также могут иметь правила двух видов: A → γB или A → γ, где A, B∈N; γ∈ V T\*. Оба эти класса грамматик эквивалентны и относятся к регулярным грамматикам. Регулярные грамматики применяются для описания базовых элементов языков программирования, таких как идентификаторы, константы, строки, комментарии и прочее. Эти грамматики просты в использовании и удобны, поэтому на их основе создаются функции лексического анализа входного языка в компиляторах.

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.

Грамматические категории взаимосвязаны следующим образом:

Все регулярные грамматики принадлежат к категории КС;

Регулярные грамматики также входят в категорию УКС;

КС-грамматики являются частью УКС;

КС-грамматика может быть классифицирована как КЗ;

Ни одна КС-грамматика не укорачивается;

КЗ-грамматика принадлежит к типу 0;

Грамматики, которые не укорачиваются, также относятся к типу 0;

УКС-грамматики также входят в категорию типа 0.

Категории грамматик имеют уникальные взаимосвязи. Изучение категорий 2 и 3 показывает, что регулярные грамматики соответствуют КС, а не наоборот. При этом каждая грамматика может быть описана как тип 0, так как этот тип не налагает ограничений на правила. Однако существуют КС-грамматики (тип 2), которые не являются ни контекстно-зависимыми, ни укорачивающимися (тип 1). В целом, сложность грамматики обратно пропорциональна максимально возможному типу, к которому она может быть отнесена. Грамматики типа 0 представляют собой наиболее сложные, в то время как те, что относятся к типу 3, являются наименее сложными.

# 2.2. Связь лексического анализа с автоматными грамматиками и конечными автоматами

Как уже упоминалось ранее, лексический анализатор занимается обработкой элементов, таких как константы, идентификаторы, ключевые слова и т.д. Автоматные грамматики, также известные как контекстно-зависимые грамматики, используются для описания более сложных языков. Они ограничивают применение правил контекстом, в котором находится нетерминальный символ.

Часто для проведения лексического анализа используют конечные автоматы, которые представляют собой машины состояний, переходящие от одного состояния к другому в зависимости от входного символа. Регулярный язык, состоящий из констант и идентификаторов, обычно описывается с использованием регулярных грамматик (праволинейных или леволинейных), которые применяются для описания регулярных языков — множества цепочек символов, выбранных из определенного словаря.

Конечные автоматы могут быть использованы для распознавания лексем в исходном коде. Каждое состояние автомата соответствует части лексемы, а переходы между состояниями определяются входными символами. Когда автомат достигает конечного состояния, он распознает лексему.

Автомат может также использовать свои состояния для определения типа распознанной лексемы. Например, автомат может переходить в одно состояние для распознавания целых чисел, и в другое состояние для распознавания идентификаторов.

Автомат может генерировать лексемы, выполняя некоторую обработку входных данных и создавая новые лексемы на выходе.

Регулярные языки исследуются как один из наиболее изученных классов языков, для которых разработана значительная математическая теория. Они широко используются в различных областях, включая разработку лексических анализаторов для компиляторов языков программирования.

Теорема, доказанная Стивеном Коулом Клини, устанавливает связь между регулярными языками и конечными автоматами. Согласно этой теореме, классы регулярных множеств и автоматных языков эквивалентны: каждый автоматный язык может быть описан с использованием регулярного выражения, и наоборот, каждое регулярное множество может быть распознано конечным автоматом. Эта теорема имеет ключевое значение для теории автоматов.

Следовательно, регулярные выражения и регулярные грамматики тесно взаимосвязаны, поскольку:

1. Любой язык, определенный с помощью регулярного выражения, также может быть описан регулярной грамматикой;

2. Любой язык, заданный с помощью регулярной грамматики, может быть выражен с помощью регулярного выражения.

Регулярные грамматики и конечные автоматы считаются эквивалентными, поскольку для каждой регулярной грамматики можно построить соответствующий конечный автомат, и наоборот. Это свойство имеет большое значение при определении лексических структур языков программирования. Создав автомат на основе грамматики, мы можем распознавать лексические структуры данного языка. Обратное утверждение также важно, так как оно позволяет определить грамматику, порождающую строки языка, которые принимаются заданным автоматом.

При изучении лексики особенности реализации конечного автомата необходимо учитывать. Каждое слово соответствует своему собственному циклу, состоящему из последовательности состояний автомата. Для этого в автомате присутствуют специальные завершающие состояния:

1. Когда автомат достигает завершающего состояния, это означает, что слово было обнаружено. Каждому завершающему состоянию соответствует свое определенное слово, поэтому дополнительные выходные сигналы не требуются;

2. Значение накопленного слова представляет часть входной строки, обработанной автоматом с начального до конечного состояния;

3. После достижения конечного состояния автомат автоматически возвращается в начальное состояние.

Таким образом, описание лексического анализатора с использованием регулярного выражения или автоматной грамматики позволяет создать конечный автомат, который распознает соответствующий язык.

# 2.3. Описание конечного автомата, лежащего в основе лексического анализатора

Автомат - это устройство, которое принимает последовательность символов на входе и преобразует их в другую последовательность символов на выходе. При этом результат на выходе зависит не только от текущего символа, но и от предыдущих символов на входе.

Представим, что эти символы записаны на ленте, где каждая ячейка содержит одну букву. Чтение ленты происходит пошагово слева направо. Автомат начинает чтение с определенного начального состояния. При чтении каждого символа автомат сдвигается на следующий символ и переходит в новое состояние, зависящее от текущего символа и текущего состояния. Для обработки цепочки символов длиной l автомат требуется l шагов. Результат обработки цепочки определяется состоянием автомата после завершения чтения.

Программное устройство, известный как языковой распознаватель, анализирует введенную строку текста и выявляет, соответствует ли она языку, описанному через регулярное выражение. Для разработки такого распознавателя на базе регулярного выражения формируется конечный автомат, представляющий собой структуру, состоящую из переходов. Этот автомат может быть детерминированным, то есть для каждого символа входной строки существует один и только один переход, или недетерминированным, когда из одного состояния возможны несколько переходов при одинаковом входном символе.

Оба вида конечных автоматов – детерминированные и недетерминированные – способны распознавать точно регулярные множества, что позволяет им идентифицировать все, что может быть описано с помощью регулярных выражений. Тем не менее, детерминированные конечные автоматы, которые характеризуются более высокой скоростью распознавания, обычно требуют больше памяти по сравнению с недетерминированными автоматами, аналогичными по функциональности.

В данной курсовой работе был использован детерминированный конечный автомат (ДКА). В нем каждое состояние имеет только один переход для каждого входного символа. Если ДКА представлен в виде таблицы переходов, то каждая запись соответствует уникальному состоянию. Это означает, что проверка принадлежности строки к языку становится простой, так как существует только один путь от начального состояния до конечного, определенного данной строкой.

Можно сформулировать определение конечного автомата следующим образом: конечный автомат - это набор

A = (V,Q,δ,q0,F), где:

V – конечный входной алфавит,

Q – конечное множества состояний;

δ - функция переходов;

q0 – известного как стартовое (начальное);

F – множество конечных состояний.

Ниже представлено описание конечного автомата, построенного в данной работе.

V = {A..Z, a..z, 0..9, (, ), [, ], {, }, «;», «.», «,», « », :, %, !, ?, \_, &, |, \, \*, +, -, =, <, >}

Q={S0..S215, S300, S301, S350, S351, S352, S353, S354, S355, S356, S357, S400, S401, S402, S403, S404, S405, S406 }

F = {S0, S300, S301, S353, S355, S357}

δ:(S,x) →S’

Функции переходов δ, представлены ниже.

(S0, Пробел)→S0

(S0, \n)→S0

(S0, \r)→S0

(S0, 1…9)→S1

(S0, 0)→S2

(S0, a)→S7

(S0, b)→S12

(S0, c)→S23

(S0, d)→S218

(S0, e)→S50

(S0, f)→S59

(S0, i)→S67

(S0, n)→S73

(S0, o)→S91

(S0, p)→S112

(S0, r)→S128

(S0, g)→S133

(S0, s)→S136

(S0, t)→S155

(S0, u)→S177

(S0, v)→S181

(S0, w)→S194

(S0, m)→S198

(S0, x)→S204

(S0, ., ,, {, }, [, ], (, ), ;, #)→S352

(S0, :)→S206

(S0, <)→S207

(S0, >)→S208

(S0, |)→S209

(S0, &)→S210

(S0, =)→S211

(S0, !)→S212

(S0, +)→S213

(S0, -)→S214

(S0, ?)→S357

(S0, %)→S215

(S0, \*)→S216

(S0, /)→S217

(S1, число)→S1

(S1, ., ,)→S3

(S2, .)→S3

(S3, число)→S4

(S4, число)→S4

(S4, e, E)→S5

(S5, число)→S5

(S5, +)→S6

(S5, -)→S6

(S6, число)→S6

(S7, s)→S8

(S7, n)→S9

(S7, u)→S10

(S7, буква/цифра/\_)→S350

(S8, m)→S300

(S9, d)→S300

(S10, t)→S11

(S11, o)→S300

(S12, i)→S13

(S12, o)→S18

(S12, r)→S20

(S13, t)→S14

(S14, a)→S15

(S14, o)→S17

(S15, n)→S16

(S16, d)→S300

(S17, r)→S300

(S18, o)→S19

(S19, l)→S300

(S20, e)→S21

(S21, a)→S22

(S22, k)→S300

(S23, a)→S24

(S23, h)→S28

(S24, s)→S25

(S24, t)→S26

(S24, l)→S30

(S24, o)→S33

(S25, e)→S300

(S26, c)→S27

(S27, h)→S300

(S28, a)→S29

(S29, r)→S300

(S30, a)→S31

(S31, s)→S32

(S32, s)→S300

(S33, n)→S34

(S34, s)→S35

(S34, t)→S36

(S35, t)→S300

(S36, i)→S38

(S38, n)→S39

(S39, u)→S40

(S40, e)→S300

(S64, a)→S65

(S65, t)→S300

(S66, r)→S300

(S67, f)→S300

(S67, n)→S68

(S68, t)→S300

(S68, l)→S99

(S69, o)→S70

(S70, o)→S71

(S71, n)→S72

(S72, g)→S300

(S73, a)→S74

(S73, e)→S81

(S73, o)→S82

(S74, m)→S75

(S75, e)→S76

(S76, s)→S77

(S77, p)→S78

(S78, a)→S79

(S79, c)→S80

(S80, e)→S300

(S81, w)→S300

(S82, t)→S83

(S83, e)→S83

(S83, \_)→S83

(S84, x)→S85

(S85, p)→S86

(S86, e)→S87

(S87, c)→S88

(S88, t)→S300

(S89, e)→S90

(S90, q)→S300

(S91, p)→S92

(S91, r)→S300

(S92, e)→S93

(S93, r)→S94

(S94, a)→S95

(S95, t)→S96

(S96, o)→S97

(S98, r)→S300

(S99, i)→S100

(S100, n)→S101

(S101, e)→S300

(S102, i)→S103

(S103, e)→S104

(S104, n)→S105

(S105, d)→S300

(S106, u)→S107

(S107, l)→S108

(S108, l)→S109

(S109, p)→S110

(S110, t)→S111

(S111, r)→S300

(S112, r)→S113

(S112, u)→S124

(S113, i)→S114

(S113, o)→S118

(S114, v)→S115

(S115, a)→S116

(S116, t)→S117

(S117, e)→S300

(S117, e)→S300

(S118, t)→S119

(S119, e)→S120

(S120, c)→S121

(S121, t)→S122

(S122, e)→S123

(S123, d)→S300

(S124, b)→S125

(S125, l)→S126

(S126, i)→S127

(S127, c)→S300

(S128, e)→S129

(S129, t)→S130

(S130, u)→S131

(S131, r)→S132

(S132, n)→S133

(S133, o)→S134

(S134, t)→S135

(S135, o)→S300

(S136, h)→S137

(S136, i)→S140

(S136, t)→S144

(S136, w)→S151

(S137, o)→S138

(S138, r)→S139

(S139, t)→S300

(S140, z)→S141

(S141, e)→S142

(S142, o)→S143

(S143, f)→S300

(S144, a)→S145

(S144, r)→S148

(S145, t)→S146

(S146, i)→S147

(S147, c)→S300

(S148, u)→S149

(S149, c)→S150

(S150, t)→S300

(S151, i)→S152

(S152, t)→S153

(S153, c)→S154

(S154, h)→S300

(S155, e)→S156

(S155, h)→S162

(S155, r)→S166

(S155, y)→S168

(S156, m)→S157

(S157, p)→S158

(S158, l)→S159

(S159, a)→S160

(S160, t)→S161

(S161, e)→S300

(S162, i)→S163

(S162, r)→S164

(S163, s)→S300

(S164, o)→S165

(S165, w)→S300

(S166, u)→S167

(S166, y)→S300

(S167, e)→S300

(S168, p)→S169

(S169, e)→S170

(S170, n)→S171

(S170, i)→S174

(S170, d)→S175

(S171, a)→S172

(S172, m)→S173

(S173, e)→S300

(S174, d)→S300

(S175, e)→S176

(S176, f)→S300

(S177, s)→S178

(S178, i)→S179

(S179, n)→S180

(S180, g)→S300

(S181, i)→S182

(S181, o)→S187

(S182, r)→S183

(S183, t)→S184

(S184, u)→S185

(S185, a)→S186

(S186, l)→S300

(S187, i)→S188

(S187, l)→S189

(S188, d)→S300

(S189, a)→S190

(S190, t)→S191

(S191, i)→S192

(S192, l)→S193

(S193, e)→S300

(S194, h)→S195

(S195, i)→S196

(S196, l)→S197

(S197, e)→S300

(S198, u)→S199

(S199, t)→S200

(S200, a)→S201

(S201, b)→S202

(S202, l)→S203

(S203, e)→S300

(S204, o)→S205

(S205, r)→S300

(S218, e)→S219

(S218, o)→S224

(S219, l)→S221

(S221, e)→S222

(S222, t)→S223

(S223, e)→S300

(S350, буква/цифра/\_)→S350

(S350, спец.символы, разделители)→S0

(S206, :)→S352

(S207, =)→S354

(S207, <)→S354

(S208, =)→S354

(S208, >)→S354

(S209, =)→S356

(S207, |)→S354

(S210, =)→S356

(S210, &)→S354

(S211, =)→S354

(S213, -,=)→S356

(S214, +,=)→S356

(S215, =)→S356

Граф конечного автомата представлен на рисунке 1.

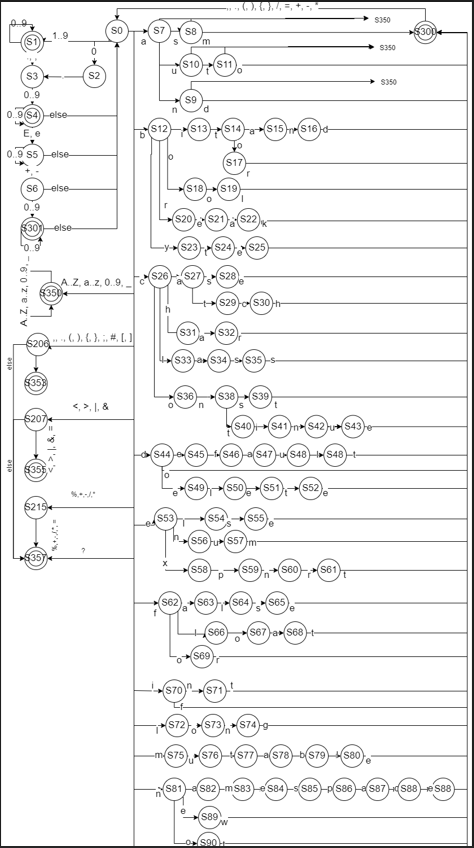
.

Рис. 1. Граф конечного автомата

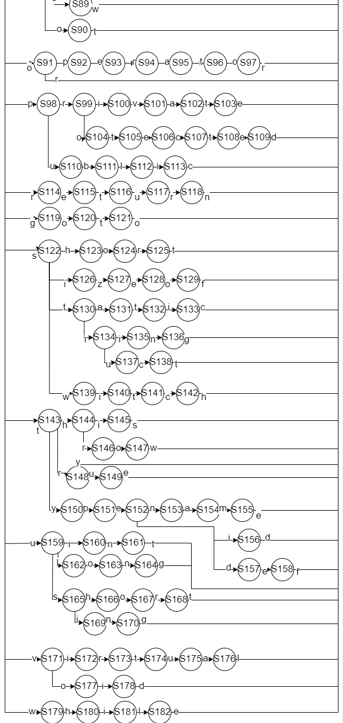


Рис. 2. Граф конечного автомата

Построение грамматики по конечному автомату

В общем регулярная грамматика представляет собой четверку: G = {N, T, S, P}, где:

N – множество нетерминалов;

T – множество терминалов;

S – начальный нетерминал;

P – непустое конечное множество правил грамматики.

Если задан конечный автомат A=(V,Q,δ,q\_0,F), то можно построить в соответствие регулярную грамматику G = {N, T, S, P}, у которой: T=V, N=Q, S={q0}. Множество правил подстановки Р строится таким образом: каждой команде автомата (qi, a) ∈qk ставится в соответствие правило подстановки qi ∈а qk , если qk ∈Q, либо qi∈а , если qk∈Z, где Z – заключительное состояние.

Описание регулярной грамматики выглядит следующим образом:

N = { S0..S215, S300, S301, S350, S351, S352, S353, S354, S355, S356, S357, S400, S401, S402, S403, S404, S405, S406 }

T = {A..Z, a..z, 0..9, (, ), [, ], {, }, «;», «.», «,», « », :, %, !, ?, \_, &, |, \, \*, +, -, =, <, >}

S = {S0}

Регулярная грамматика представлена ниже

S0→ПробелS0

S0→\nS0

S0→\rS0

S0→1…9S0

S0→0S2

S0→aS7

S0→bS12

S0→cS23

S0→dS218

S0→eS50

S0→fS59

S0→iS67

S0→nS73

S0→oS91

S0→pS112

S0→rS128

S0→gS133

S0→sS136

S0→tS155

S0→uS177

S0→vS181

S0→wS194

S0→mS198

S0→xS204

S0→, ., ,, {, }, [, ], (, ), ;, #S352

S0→:S206

S0→<S207

S0→>S208

S0→|S209

S0→&S210

S0→=S211

S0→!S212

S0→+S213

S0→-S214

S0→?S357

S0→%S215

S0→\*S216

S0→/S217

S1→ числоS1

S1→ ., ,S3

S2→ .S3

S3→ числоS4

S4→ числоS4

S3→ e,ES5

S5→ числоS5

S5→ +S6

S5→ -S6

S6→ числоS6

S7→ sS8

S7→ nS9

S7→ uS10

S7→ буква/цифра/\_S350

S8→ mS300

S9→ dS300

S10→ tS11

S11→ oS300

S12→ iS13

S12→ oS18

S12→ rS20

S13→ tS14

S14→ aS15

S14→ oS17

S15→ nS16

S16→ dS300

S17→ rS300

S18→ oS19

S19→ oS300

S20→ eS21

S21→ aS22

S22→ kS300

S23→ aS300

S23→ hS300

S24→ sS25

S24→ tS26

S24→ lS30

S24→ oS33

S25→ eS300

S26→ cS27

S27→ hS300

S28→ aS29

S29→ rS300

S30→ aS31

S31→ sS32

S32→ sS300

S33→ nS34

S34→ sS35

S34→ tS36

S35→ tS300

S36→ iS38

S38→ nS39

S39→ uS40

S40→ eS300

S41→ eS42

S41→ oS300

S42→ fS43

S42→ lS47

S43→ aS44

S44→ uS45

S45→ lS46

S46→ tS300

S47→ eS48

S48→ tS49

S49→ eS300

S50→ lS51

S50→ nS53

S50→ xS55

S51→ sS52

S52→ eS300

S53→ uS54

S54→ mS300

S55→ pS56

S56→ oS57

S57→ rS58

S58→ tS300

S59→ aS60

S59→ lS63

S59→ oS66

S59→ rS102

S60→ lS61

S61→ sS62

S62→ eS300

S63→ oS64

S64→ aS65

S65→ tS300

S66→ rS300

S67→ fS300

S67→ nS68

S68→ tS300

S68→ lS99

S69→ oS70

S70→ oS71

S71→ nS72

S72→ gS300

S73→ aS300

S73→ eS300

S73→ oS300

S74→ mS75

S75→ eS76

S76→ sS77

S77→ fS78

S78→ aS79

S79→ cS80

S80→ eS300

S81→ wS300

S82→ tS83

S83→ eS83

S83→ \_S83

S84→ xS85

S85→ pS86

S86→ eS86

S87→ cS88

S88→ tS300

S89→ eS90

S90→ qS300

S91→ pS92

S91→ rS300

S92→ eS93

S93→ rS94

S94→ aS95

S95→ tS96

S96→ oS97

S98→ oS300

S99→ iS100

S100→ nS101

S101→ eS300

S102→ iS103

S103→ eS104

S104→ nS105

S105→ dS300

S106→ uS107

S107→ lS108

S108→ lS109

S109→ pS110

S110→ tS111

S111→rS300

S112→rS113

S112→uS124

S113→iS114

S113→oS118

S114→vS115

S115→aS116

S116→tS117

S117→eS300

S117→eS300

S118→tS119

S119→eS120

S120→cS121

S121→tS122

S122→eS123

S123→eS300

S124→bS125

S125→lS126

S126→iS126

S127→cS300

S128→eS129

S129→tS130

S130→uS131

S131→rS132

S132→nS133

S133→oS134

S134→tS135

S135→oS300

S136→hS137

S136→iS140

S136→tS144

S136→wS151

S137→oS138

S138→rS139

S139→tS300

S140→zS141

S141→eS142

S142→oS143

S143→fS300

S144→aS145

S144→rS148

S145→tS146

S146→iS147

S147→cS300

S148→uS149

S149→cS150

S150→tS300

S151→iS152

S152→tS153

S153→cS154

S154→hS300

S155→eS156

S155→hS162

S155→rS166

S155→yS168

S156→mS157

S157→pS158

S158→lS159

S159→aS160

S160→tS161

S161→eS300

S162→iS163

S162→rS164

S163→sS300

S164→oS165

S165→wS300

S166→tS167

S166→yS300

S167→eS300

S168→pS169

S169→eS170

S170→nS171

S170→iS174

S170→dS175

S171→aS172

S172→mS173

S173→eS300

S174→dS300

S175→eS176

S176→fS300

S177→sS178

S178→iS178

S179→nS180

S180→gS300

S181→iS182

S181→oS187

S182→rS183

S183→tS184

S184→uS186

S185→aS186

S186→lS300

S187→iS188

S187→lS189

S188→dS300

S189→aS190

S190→tS191

S191→iS192

S192→lS193

S193→eS300

S194→hS195

S195→iS196

S196→lS197

S197→eS300

S198→uS199

S199→tS200

S200→aS201

S201→bS202

S202→lS203

S203→eS300

S204→oS205

S205→rS300

S218→eS219

S218→oS224

S219→lS221

S221→eS222

S222→tS223

S223→eS300

S350→ буква, цифра,\_

S350→спец.символы,

Разделители S0

S206→:S352

S207→ =S354

S207→ <S354

S208→ =S354

S208→ >S354

S209→ =S356

S209→ |S354

S210→ =S356

S210→ &S354

S211→ =S354

S213→ -=S356

S214→ +=S356

S215→ =S356

Для любого конечного автомата существует регулярный язык, строки которого принимаются этим автоматом, то есть конечный автомат есть форма представления регулярных языков.

Обратное утверждение тоже верно: для любого непустого регулярного языка, порождаемого регулярной грамматикой, можно построить соответствующий конечный автомат, возможно недетерминированный, который будет порождать и распознавать этот язык. Ниже приведена сводная таблица, в которой каждой продукции регулярной грамматики соответствует эквивалентное правило конечного автомата. (табл.3).

Таблица 3

Соответствие правил КА и продукции регулярной грамматики

|  |  |
| --- | --- |
| Конечный автомат | Регулярная грамматика |
| 1 | 2 |
| (S0, Пробел)→S0 | S0→ПробелS0 |
| (S0, \n)→S0 | S0→\nS0 |
| (S0, \r)→S0 | S0→\rS0 |
| (S0, 1…9)→S1 | S0→1…9S0 |
| (S0, 0)→S2 | S0→0S2 |
| (S0, a)→S7 | S0→aS7 |
| (S0, b)→S12 | S0→bS12 |
| (S0, c)→S23 | S0→cS23 |
| (S0, d)→S218 | S0→dS218 |
| (S0, e)→S50 | S0→eS50 |
| (S0, f)→S59 | S0→fS59 |
| (S0, i)→S67 | S0→iS67 |
| (S0, n)→S73 | S0→nS73 |
| (S0, o)→S91 | S0→oS91 |
| (S0, p)→S112 | S0→pS112 |
| (S0, r)→S128 | S0→rS128 |
| (S0, g)→S133 | S0→gS133 |
| (S0, s)→S136 | S0→sS136 |
| (S0, t)→S155 | S0→tS155 |
| (S0, u)→S177 | S0→uS177 |
| (S0, v)→S181 | S0→vS181 |
| (S0, w)→S194 | S0→wS194 |
| (S0, m)→S198 | S0→mS198 |
| (S0, x)→S204 | S0→xS204 |
| (S0, ., ,, {, }, [, ], (, ), ;, #)→S352 | S0→, ., ,, {, }, [, ], (, ), ;, #S352 |
| (S0, :)→S206 | S0→:S206 |
| (S0, <)→S207 | S0→<S207 |
| (S0, >)→S208 | S0→>S208 |
| (S0, |)→S209 | S0→|S209 |
| (S0, &)→S210 | S0→&S210 |
| (S0, =)→S211 | S0→=S211 |
| (S0, !)→S212 | S0→!S212 |

Продолжение таблицы 3

|  |  |
| --- | --- |
| 1 | 2 |
| (S0, +)→S213 | S0→+S213 |
| (S0, -)→S214 | S0→-S214 |
| (S0, ?)→S357 | S0→?S357 |
| (S0, %)→S215 | S0→%S215 |
| (S0, \*)→S216 | S0→\*S216 |
| (S0, /)→S217 | S0→/S217 |
| (S1, число)→S1 | S1→ числоS1 |
| (S1, ., ,)→S3 | S1→ ., ,S3 |
| (S2, .)→S3 | S2→ .S3 |
| (S3, число)→S4 | S3→ числоS4 |
| (S4, число)→S4 | S4→ числоS4 |
| (S4, e, E)→S5 | S3→ e,ES5 |
| (S5, число)→S5 | S5→ числоS5 |
| (S5, +)→S6 | S5→ +S6 |
| (S5, -)→S6 | S5→ -S6 |
| (S6, число)→S6 | S6→ числоS6 |
| (S7, s)→S8 | S7→ sS8 |
| (S7, n)→S9 | S7→ nS9 |
| (S7, u)→S10 | S7→ uS10 |
| (S7, буква/цифра/\_)→S350 | S7→ буква/цифра/\_S350 |
| (S8, m)→S300 | S8→ mS300 |
| (S9, d)→S300 | S9→ dS300 |
| (S10, t)→S11 | S10→ tS11 |
| (S11, o)→S300 | S11→ oS300 |
| (S12, i)→S13 | S12→ iS13 |
| (S12, o)→S18 | S12→ oS18 |
| (S12, r)→S20 | S12→ rS20 |
| (S13, t)→S14 | S13→ tS14 |
| (S14, a)→S15 | S14→ aS15 |
| (S14, o)→S17 | S14→ oS17 |

Продолжение таблицы 3

|  |  |
| --- | --- |
| 1 | 2 |
| (S15, n)→S16 | S15→ nS16 |
| (S16, d)→S300 | S16→ dS300 |
| (S17, r)→S300 | S17→ rS300 |
| (S18, o)→S19 | S18→ oS19 |
| (S19, l)→S300 | S19→ oS300 |
| (S20, e)→S21 | S20→ eS21 |
| (S21, a)→S22 | S21→ aS22 |
| (S22, k)→S300 | S22→ kS300 |
| (S23, a)→S24 | S23→ aS300 |
| (S23, h)→S28 | S23→ hS300 |
| (S24, s)→S25 | S24→ sS25 |
| (S24, t)→S26 | S24→ tS26 |
| (S24, l)→S30 | S24→ lS30 |
| (S24, o)→S33 | S24→ oS33 |
| (S25, e)→S300 | S25→ eS300 |
| (S26, c)→S27 | S26→ cS27 |
| (S27, h)→S300 | S27→ hS300 |
| (S28, a)→S29 | S28→ aS29 |
| (S29, r)→S300 | S29→ rS300 |
| (S30, a)→S31 | S30→ aS31 |
| (S31, s)→S32 | S31→ sS32 |
| (S32, s)→S300 | S32→ sS300 |
| (S33, n)→S34 | S33→ nS34 |
| (S34, s)→S35 | S34→ sS35 |
| (S34, t)→S36 | S34→ tS36 |
| (S35, t)→S300 | S35→ tS300 |
| (S36, i)→S38 | S36→ iS38 |
| (S38, n)→S39 | S38→ nS39 |
| (S39, u)→S40 | S39→ uS40 |
| (S40, e)→S300 | S40→ eS300 |

Продолжение таблицы 3

|  |  |
| --- | --- |
| 1 | 2 |
| (S41, e)→S42 | S41→ eS42 |
| (S41, o)→S300 | S41→ oS300 |
| (S42, f)→S43 | S42→ fS43 |
| (S42, l)→S47 | S42→ lS47 |
| (S43, a)→S44 | S43→ aS44 |
| (S44, u)→S45 | S44→ uS45 |
| (S45, l)→S46 | S45→ lS46 |
| (S46, t)→S300 | S46→ tS300 |
| (S47, e)→S48 | S47→ eS48 |
| (S48, t)→S49 | S48→ tS49 |
| (S49, e)→S300 | S49→ eS300 |
| (S50, l)→S51 | S50→ lS51 |
| (S50, n)→S53 | S50→ nS53 |
| (S50, x)→S55 | S50→ xS55 |
| (S51, s)→S52 | S51→ sS52 |
| (S52, e)→S300 | S52→ eS300 |
| (S53, u)→S54 | S53→ uS54 |
| (S54, m)→S300 | S54→ mS300 |
| (S55, p)→S56 | S55→ pS56 |
| (S56, o)→S57 | S56→ oS57 |
| (S57, r)→S58 | S57→ rS58 |
| (S58, t)→S300 | S58→ tS300 |
| (S59, a)→S60 | S59→ aS60 |
| (S59, l)→S63 | S59→ lS63 |
| (S59, o)→S66 | S59→ oS66 |
| (S59, r)→S102 | S59→ rS102 |
| (S60, l)→S61 | S60→ lS61 |
| (S61, s)→S62 | S61→ sS62 |
| (S62, e)→S300 | S62→ eS300 |
| (S63, o)→S64 | S63→ oS64 |

Продолжение таблицы 3

|  |  |
| --- | --- |
| 1 | 2 |
| (S64, a)→S65 | S64→ aS65 |
| (S65, t)→S300 | S65→ tS300 |
| (S66, r)→S300 | S66→ rS300 |
| (S67, f)→S300 | S67→ fS300 |
| (S67, n)→S68 | S67→ nS68 |
| (S68, t)→S300 | S68→ tS300 |
| (S68, l)→S99 | S68→ lS99 |
| (S69, o)→S70 | S69→ oS70 |
| (S70, o)→S71 | S70→ oS71 |
| (S71, n)→S72 | S71→ nS72 |
| (S72, g)→S300 | S72→ gS300 |
| (S73, a)→S74 | S73→ aS300 |
| (S73, e)→S81 | S73→ eS300 |
| (S73, o)→S82 | S73→ oS300 |
| (S74, m)→S75 | S74→ mS75 |
| (S75, e)→S76 | S75→ eS76 |
| (S76, s)→S77 | S76→ sS77 |
| (S77, p)→S78 | S77→ fS78 |
| (S78, a)→S79 | S78→ aS79 |
| (S79, c)→S80 | S79→ cS80 |
| (S80, e)→S300 | S80→ eS300 |
| (S81, w)→S300 | S81→ wS300 |
| (S82, t)→S83 | S82→ tS83 |
| (S83, e)→S83 | S83→ eS83 |
| (S83, \_)→S83 | S83→ \_S83 |
| (S84, x)→S85 | S84→ xS85 |
| (S85, p)→S86 | S85→ pS86 |
| (S86, e)→S87 | S86→ eS86 |
| (S87, c)→S88 | S87→ cS88 |

Продолжение таблицы 3

|  |  |
| --- | --- |
| 1 | 2 |
| (S88, t)→S300 | S88→ tS300 |
| (S89, e)→S90 | S89→ eS90 |
| (S90, q)→S300 | S90→ qS300 |
| (S91, p)→S92 | S91→ pS92 |
| (S91, r)→S300 | S91→ rS300 |
| (S92, e)→S93 | S92→ eS93 |
| (S93, r)→S94 | S93→ rS94 |
| (S94, a)→S95 | S94→ aS95 |
| (S95, t)→S96 | S95→ tS96 |
| (S96, o)→S97 | S96→ oS97 |
| (S98, r)→S300 | S98→ oS300 |
| (S99, i)→S100 | S99→ iS100 |
| (S100, n)→S101 | S100→ nS101 |
| (S101, e)→S300 | S101→ eS300 |
| (S102, i)→S103 | S102→ iS103 |
| (S103, e)→S104 | S103→ eS104 |
| (S104, n)→S105 | S104→ nS105 |
| (S105, d)→S300 | S105→ dS300 |
| (S106, u)→S107 | S106→ uS107 |
| (S107, l)→S108 | S107→ lS108 |
| (S108, l)→S109 | S108→ lS109 |
| (S109, p)→S110 | S109→ pS110 |
| (S110, t)→S111 | S110→ tS111 |
| (S111, r)→S300 | S111→rS300 |
| (S112, r)→S113 | S112→rS113 |
| (S112, u)→S124 | S112→uS124 |
| (S113, i)→S114 | S113→iS114 |
| (S113, o)→S118 | S113→oS118 |
| (S114, v)→S115 | S114→vS115 |
| (S115, a)→S116 | S115→aS116 |
| (S116, t)→S117 | S116→tS117 |

Продолжение таблицы 3

|  |  |
| --- | --- |
| 1 | 2 |
| (S117, e)→S300 | S117→eS300 |
| (S117, e)→S300 | S117→eS300 |
| (S118, t)→S119 | S118→tS119 |
| (S119, e)→S120 | S119→eS120 |
| (S120, c)→S121 | S120→cS121 |
| (S121, t)→S122 | S121→tS122 |
| (S122, e)→S123 | S122→eS123 |
| (S123, d)→S300 | S123→eS300 |
| (S124, b)→S125 | S124→bS125 |
| (S125, l)→S126 | S125→lS126 |
| (S126, i)→S127 | S126→iS126 |
| (S127, c)→S300 | S127→cS300 |
| (S128, e)→S129 | S128→eS129 |
| (S129, t)→S130 | S129→tS130 |
| (S130, u)→S131 | S130→uS131 |
| (S131, r)→S132 | S131→rS132 |
| (S132, n)→S133 | S132→nS133 |
| (S133, o)→S134 | S133→oS134 |
| (S134, t)→S135 | S134→tS135 |
| (S135, o)→S300 | S135→oS300 |
| (S136, h)→S137 | S136→hS137 |
| (S136, i)→S140 | S136→iS140 |
| (S136, t)→S144 | S136→tS144 |
| (S136, w)→S151 | S136→wS151 |
| (S137, o)→S138 | S137→oS138 |
| (S138, r)→S139 | S138→rS139 |
| (S139, t)→S300 | S139→tS300 |
| (S140, z)→S141 | S140→zS141 |
| (S141, e)→S142 | S141→eS142 |
| (S142, o)→S143 | S142→oS143 |
| (S143, f)→S300 | S143→fS300 |

Продолжение таблицы 3

|  |  |
| --- | --- |
| 1 | 2 |
| (S144, a)→S145 | S144→aS145 |
| (S144, r)→S148 | S144→rS148 |
| (S145, t)→S146 | S145→tS146 |
| (S146, i)→S147 | S146→iS147 |
| (S147, c)→S300 | S147→cS300 |
| (S148, u)→S149 | S148→uS149 |
| (S149, c)→S150 | S149→cS150 |
| (S150, t)→S300 | S150→tS300 |
| (S151, i)→S152 | S151→iS152 |
| (S152, t)→S153 | S152→tS153 |
| (S153, c)→S154 | S153→cS154 |
| (S154, h)→S300 | S154→hS300 |
| (S155, e)→S156 | S155→eS156 |
| (S155, h)→S162 | S155→hS162 |
| (S155, r)→S166 | S155→rS166 |
| (S155, y)→S168 | S155→yS168 |
| (S156, m)→S157 | S156→mS157 |
| (S157, p)→S158 | S157→pS158 |
| (S158, l)→S159 | S158→lS159 |
| (S159, a)→S160 | S159→aS160 |
| (S160, t)→S161 | S160→tS161 |
| (S161, e)→S300 | S161→eS300 |
| (S162, i)→S163 | S162→iS163 |
| (S162, r)→S164 | S162→rS164 |
| (S163, s)→S300 | S163→sS300 |
| (S164, o)→S165 | S164→oS165 |
| (S165, w)→S300 | S165→wS300 |
| (S166, u)→S167 | S166→tS167 |
| (S166, y)→S300 | S166→yS300 |
| (S167, e)→S300 | S167→eS300 |
| (S168, p)→S169 | S168→pS169 |

Продолжение таблицы 3

|  |  |
| --- | --- |
| 1 | 2 |
| (S169, e)→S170 | S169→eS170 |
| (S170, n)→S171 | S170→nS171 |
| (S170, i)→S174 | S170→iS174 |
| (S170, d)→S175 | S170→dS175 |
| (S171, a)→S172 | S171→aS172 |
| (S172, m)→S173 | S172→mS173 |
| (S173, e)→S300 | S173→eS300 |
| (S174, d)→S300 | S174→dS300 |
| (S175, e)→S176 | S175→eS176 |
| (S176, f)→S300 | S176→fS300 |
| (S177, s)→S178 | S177→sS178 |
| (S178, i)→S179 | S178→iS178 |
| (S179, n)→S180 | S179→nS180 |
| (S180, g)→S300 | S180→gS300 |
| (S181, i)→S182 | S181→iS182 |
| (S181, o)→S187 | S181→oS187 |
| (S182, r)→S183 | S182→rS183 |
| (S183, t)→S184 | S183→tS184 |
| (S184, u)→S185 | S184→uS186 |
| (S185, a)→S186 | S185→aS186 |
| (S186, l)→S300 | S186→lS300 |
| (S187, i)→S188 | S187→iS188 |
| (S187, l)→S189 | S187→lS189 |
| (S188, d)→S300 | S188→dS300 |
| (S189, a)→S190 | S189→aS190 |
| (S190, t)→S191 | S190→tS191 |
| (S191, i)→S192 | S191→iS192 |
| (S192, l)→S193 | S192→lS193 |
| (S193, e)→S300 | S193→eS300 |
| (S194, h)→S195 | S194→hS195 |
| (S195, i)→S196 | S195→iS196 |

Продолжение таблицы 3

|  |  |
| --- | --- |
| 1 | 2 |
| (S196, l)→S197 | S196→lS197 |
| (S197, e)→S300 | S197→eS300 |
| (S198, u)→S199 | S198→uS199 |
| (S199, t)→S200 | S199→tS200 |
| (S200, a)→S201 | S200→aS201 |
| (S201, b)→S202 | S201→bS202 |
| (S202, l)→S203 | S202→lS203 |
| (S203, e)→S300 | S203→eS300 |
| (S204, o)→S205 | S204→oS205 |
| (S205, r)→S300 | S205→rS300 |
| (S218, e)→S219 | S218→eS219 |
| (S218, o)→S224 | S218→oS224 |
| (S219, l)→S221 | S219→lS221 |
| (S221, e)→S222 | S221→eS222 |
| (S222, t)→S223 | S222→tS223 |
| (S223, e)→S300 | S223→eS300 |
| (S350, буква/цифра/\_)→S350 | S350→ буква, цифра,\_ |
| (S350, спец.символы, разделители)→S0 | S350→ спец.символы, разделителиS0 |
| (S206, :)→S352 | S206→:S352 |
| (S207, =)→S354 | S207→ =S354 |
| (S207, <)→S354 | S207→ <S354 |
| (S208, =)→S354 | S208→ =S354 |
| (S208, >)→S354 | S208→ >S354 |
| (S209, =)→S356 | S209→ =S356 |
| (S207, |)→S354 | S209→ |S354 |
| (S210, =)→S356 | S210→ =S356 |
| (S210, &)→S354 | S210→ &S354 |
| (S211, =)→S354 | S211→ =S354 |
| (S213, -,=)→S356 | S213→ -=S356 |
| (S214, +,=)→S356 | S214→ +=S356 |
| (S215, =)→S356 | S215→ =S356 |

# 2.4 Построение КС-грамматики, порождения и дерева вывода

# 2.4.1 Связь между КС-грамматиками и синтаксическим анализом.

# Порождение и дерево вывода

Особый интерес в структуре грамматик Хомского вызывают контекстно-свободные (КС) грамматики и языки. С использованием КС-грамматик создаются более сложные синтаксические конструкции: описания различных типов и переменных, математические и логические выражения, управляющие операторы, а также полные программы на входном языке. После сканирования исходного кода формируется таблица лексем, которая затем передается на обработку синтаксическому анализатору - еще одному компоненту компилятора. Работа синтаксического анализатора базируется на применении правил КС грамматики, которые определяют структуры исходного языка и необходимы для последующей обработки программы.

Машины с магазинной памятью, или МП-автоматы, являются основой для создания распознавателей КС-языков. Они представляют собой однонаправленные недетерминированные распознаватели с ограниченной линейно памятью. МП-автомат отличается от обычного конечного автомата наличием стека, в который можно добавлять специальные символы, соответствующие терминальным и нетерминальным символам грамматики языка. Переход от одного состояния к другому в МП-автомате зависит не только от текущего входного символа, но также от одного или нескольких символов на вершине стека. Таким образом, состояние МП-автомата определяется тремя параметрами: текущим состоянием, текущим символом входной цепочки и содержимым стека.

МП-автомат считается недетерминированным, если при одних и тех же условиях возможно несколько вариантов перехода. Если же из любого состояния МП-автомата по каждому входному символу существует только один возможный переход в следующее состояние, то такой автомат считается детерминированным (ДМП-автоматом). Детерминированные МП-автоматы определяют класс языков контекстно-свободных, для которых существуют однозначные контекстно-свободные грамматики. Именно этот класс языков лежит в основе синтаксических конструкций всех языков программирования, поскольку каждая синтаксическая конструкция должна иметь только одну возможную интерпретацию.

Существует несколько методов для описания процесса определения языка с помощью грамматики. Один из них - это построение деревьев разбора, другой - это порождение, которое точно описывает процесс построения дерева разбора сверху вниз. В порождении продукция рассматривается как правило замены, где нетерминал в левой части заменяется строкой из правой части продукции. Порождение представляет собой последовательное применение продукций в произвольном порядке для получения цепочки замещений.

Существуют два основных способа построения дерева разбора:

1. Нисходящий анализ: Строит дерево "сверху вниз", от корня к листьям, добавляя поддеревья по мере движения вниз.

2. Восходящий анализ: Строит дерево "снизу вверх", от листьев к корню, объединяя узлы, соответствующие правой части продукции, в новые узлы.

# 2.4.2 Построение КС-грамматики

Согласно варианту курсовой работы необходимо построить КС - грамматику для оператора цикла switch и оператора for для обработки массивов языка, а также построить правостороннее порождение и дерево вывода.

В качестве входной цепочки используется следующая часть код:

switch (a){

case 1:

c = 0;

for (int i = 0; i < n; i++) {

c = c + q; cout << c << endl;

}

break;

case 2:

c = 1;

for (int i = 0; i < n; i++) {

c = c \* q; cout << c << endl;

}

break;

default:

cout << "Error wrong number";

break;

}

Ниже представлено описание КС-грамматики.

G={N,T,S,P}

1. N={A,B,C,D,E,F,I, J,K,L,O,Q,R,M,N,T,V, W};
2. T={switch, for, {, }, (, ), ;, int, double, string, cin, cout, endl, >>,<<, +, ++,

--, - , =, ==, \*, /, &&, ||, >, >=, <=, <, id, !=, [, ], digit };

1. S={A};
2. P = {  
   A→ AA | for(R){A} | switch(I){A} | C; | J; | M; | K;

B → EI | EIZ // Возможность применять несколько сдвигов

C → coutB | cinB // ввод/вывод

D → digit // числа

E →<< | >> // Сдвиги

F →IOI // блок условия в цикле for

I →Id // Идентификаторы

J →WILD | ILD // Блок инициализации

K →IQ | ILILI // Блок модификации переменной

L→ = | + | - | \* | / // Знаки операций

O →== | < | > | <= | >= | != | || | && // Знаки лог сравнения

Q →-- | ++ // Инкремент декремент

R →J; F; K // Содержимое конструкции for

M → N | T | NM // Содержимое switch

N → case D: AV // Содержимое case

T →default: AV // Содержимое default

V →break;

W →int | double | string // тип данных

Z → Eendl; // Конец строки вывода

Порождение представлено ниже.

**A** →switch(**I**){A} → switch(id){**A**} → switch(id){**M**} → switch(id){**N**M} → switch(id){ case **D**: AV M} → switch(id){ case digit: **A**V M} → switch(id){ case digit: **A**AV M} → switch(id){ case digit: **J**AV M} → switch(id){ case digit: **I**LD; AV M} → switch(id){ case digit: id **L**D; AV M} → switch(id){ case digit: id = **D**; AV M} → switch(id){ case digit: id = digit; **A**V M} → switch(id){ case digit: id = digit; for(**R**){A} V M} → switch(id){ case digit: id = digit; for(**J**; F; K){A} V M} → switch(id){ case digit: id = digit; for(**W**ILD; F; K){A} V M} → switch(id){ case digit: id = digit; for(int **I**LD; F; K){A} V M} → switch(id){ case digit: id = digit; for(int id **L**D; F; K){A} V M} → switch(id){ case digit: id = digit; for(int id = **D**; F; K){A} V M} → switch(id){ case digit: id = digit; for(int id = digit; **F**; K){A} V M} → switch(id){ case digit: id = digit; for(int id = digit; **I**OI; K){A} V M} → switch(id){ case digit: id = digit; for(int id = digit; id **O**I; K){A} V M} → switch(id){ case digit: id = digit; for(int id = digit; id < **I**; K){A} V M} → switch(id){ case digit: id = digit; for(int id = digit; id < id; **K**){A} V M} → switch(id){ case digit: id = digit; for(int id = digit; id < id; **I**Q){A} V M} → switch(id){ case digit: id = digit; for(int id = digit; id < id; id **Q**){A} V M} → switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){**A**} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){**A**A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){**K**A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ **I**LILI; A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id **L**ILI; A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = **I**LI; A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id **L**I; A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + **I**; A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; **A**} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; **C**} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout**B** } V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout **E**IZ; } V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << **I**Z; } V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id **Z**; } V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id **E**endl; } V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } **V** M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; **M**}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; **N**M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case **D**: AV M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; **N**M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: **A**V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; **N**M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: **A**AV M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; **N**M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: **J**;AV M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; **N**M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: **I**LD;AV M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; **N**M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id **L**D;AV M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; **N**M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = **D**;AV M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; **N**M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; **A**V M}→switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(**R**){A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(**J**; F; K){A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(**W**ILD; F; K){A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int **I**LD; F; K){A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id **L**D; F; K){A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = **D**; F; K){A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; **F**; K){A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; **I**OI; K){A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id **O**I; K){A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < **I**; K){A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; **K**){A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; **I**Q){A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id **Q**){A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){**A**} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){**A**A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){**K**A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ **I**LILI; A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id **L**ILI; A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = **I**LI; A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id **L**I; A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* **I;** A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; A} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; **A**} V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout**B** } V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout **E**IZ } V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout << **I**Z } V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout << id Z } AV M}<< id **Z** } V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout << id **E**endl; } V M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout << id << endl; } **V** M}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout << id << endl; } break; **M**}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout << id << endl; } break; **T**}→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout << id << endl; } break; default: **A**V }→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout << id << endl; } break; default: **C**V }→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout << id << endl; } break; default: cout**B**; V }→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout << id << endl; } break; default: cout **E**I; V }→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout << id << endl; } break; default: cout << **I;** V }→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout << id << endl; } break; default: cout << id; **V** }→ switch(id){ case digit: id = digit; for(int id = digit; id < id; id++){ id = id + id; cout << id << endl; } break; case digit: id = digit; for(int id = digit; id < id; id++){ id = id \* id; cout << id << endl; } break; default: cout << id; break;}

Дерево представлено на рисунке 3:

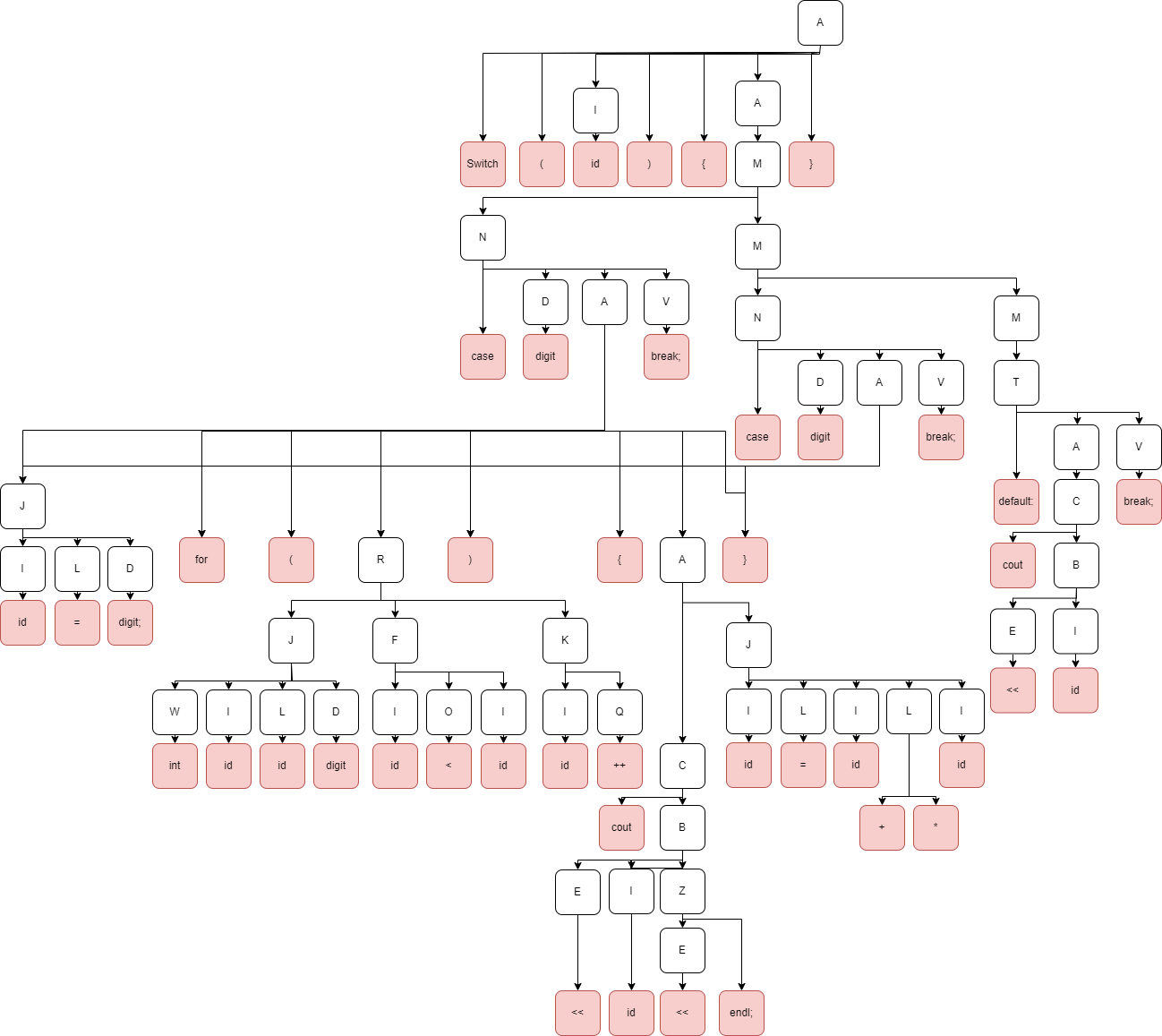


Рис.3.Дерево вывода

# 3. Логическое проектирование

# 3.1 Разработка блок-схем алгоритма работы программы

Блок-схема общего алгоритма работы программы представлена на рисунке 4.

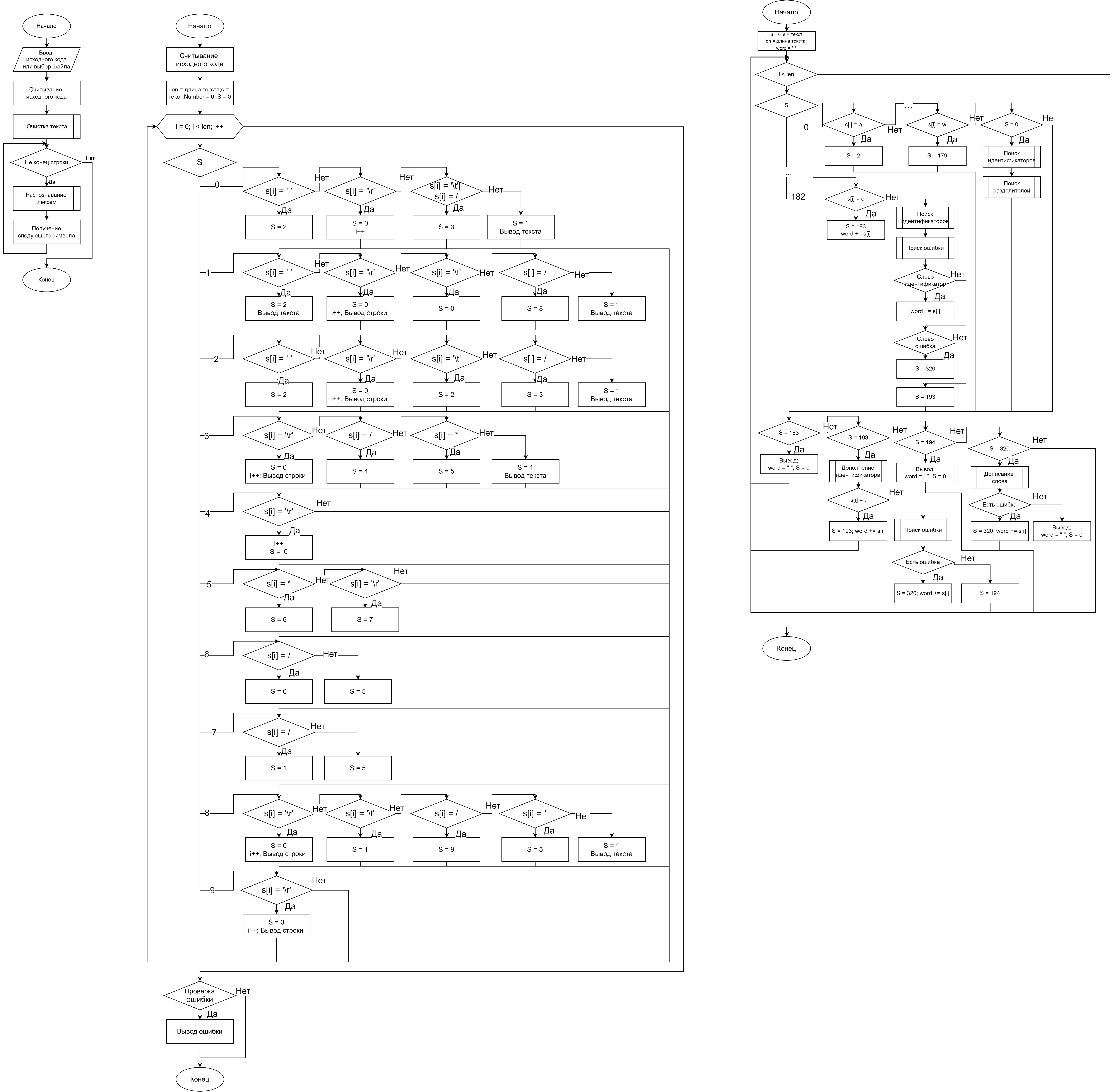


Рис 4. Общая блок-схема программы

Ниже изображены блок-схемы отдельных функциональных частей программы. (рис.5 - 12)

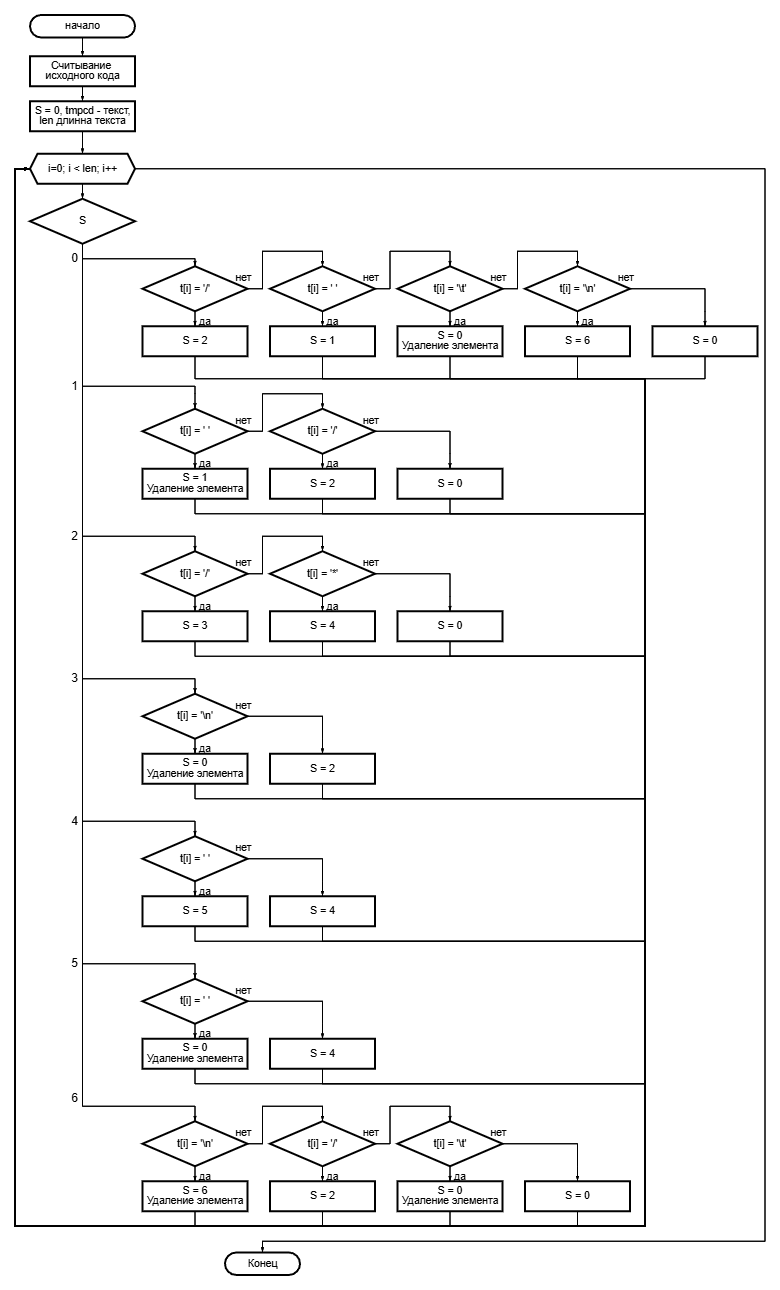


Рис 5. Блок-схема функции предобработки текста

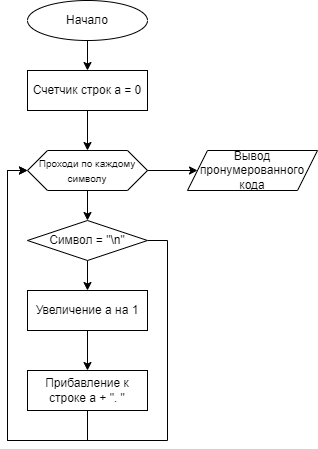


Рис 6. Нумерация строк исходного кода



Рис. 7. Блок-схема вывода псевдокода, дескрипторного кода



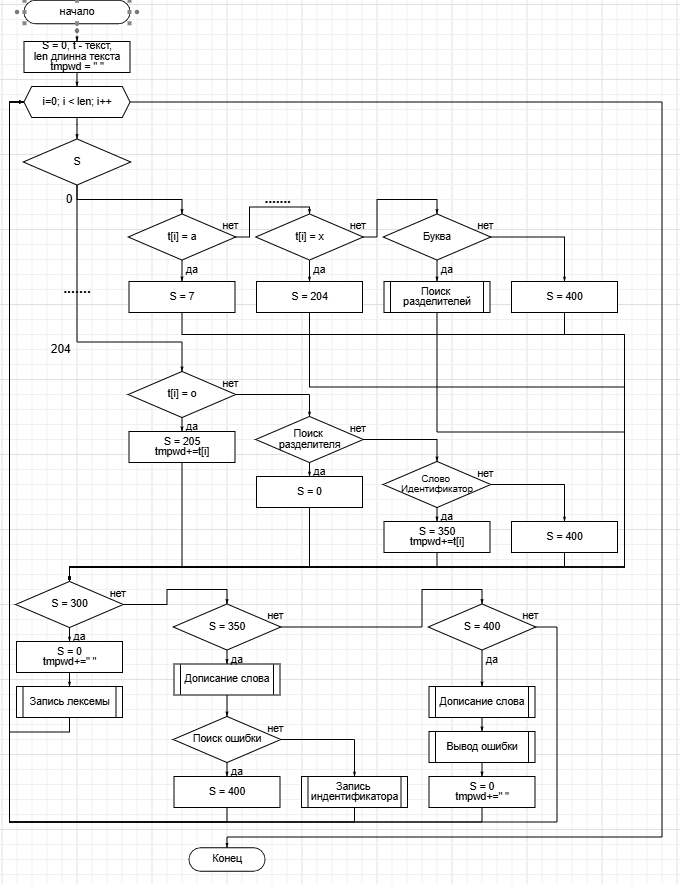


Рис 8. Блок-схема поиска ключевых слов и идентификаторов

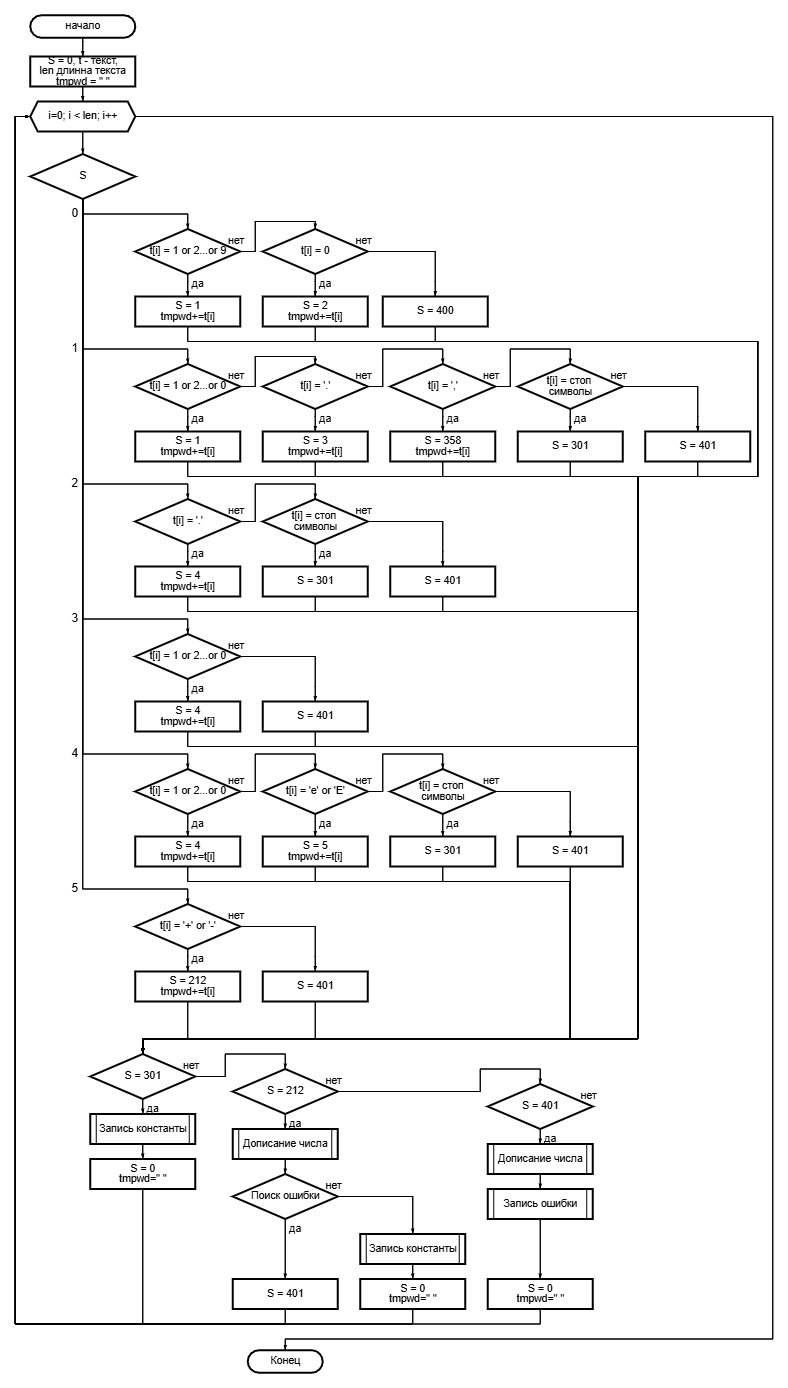


Рис 9. Блок-схема поиска констант

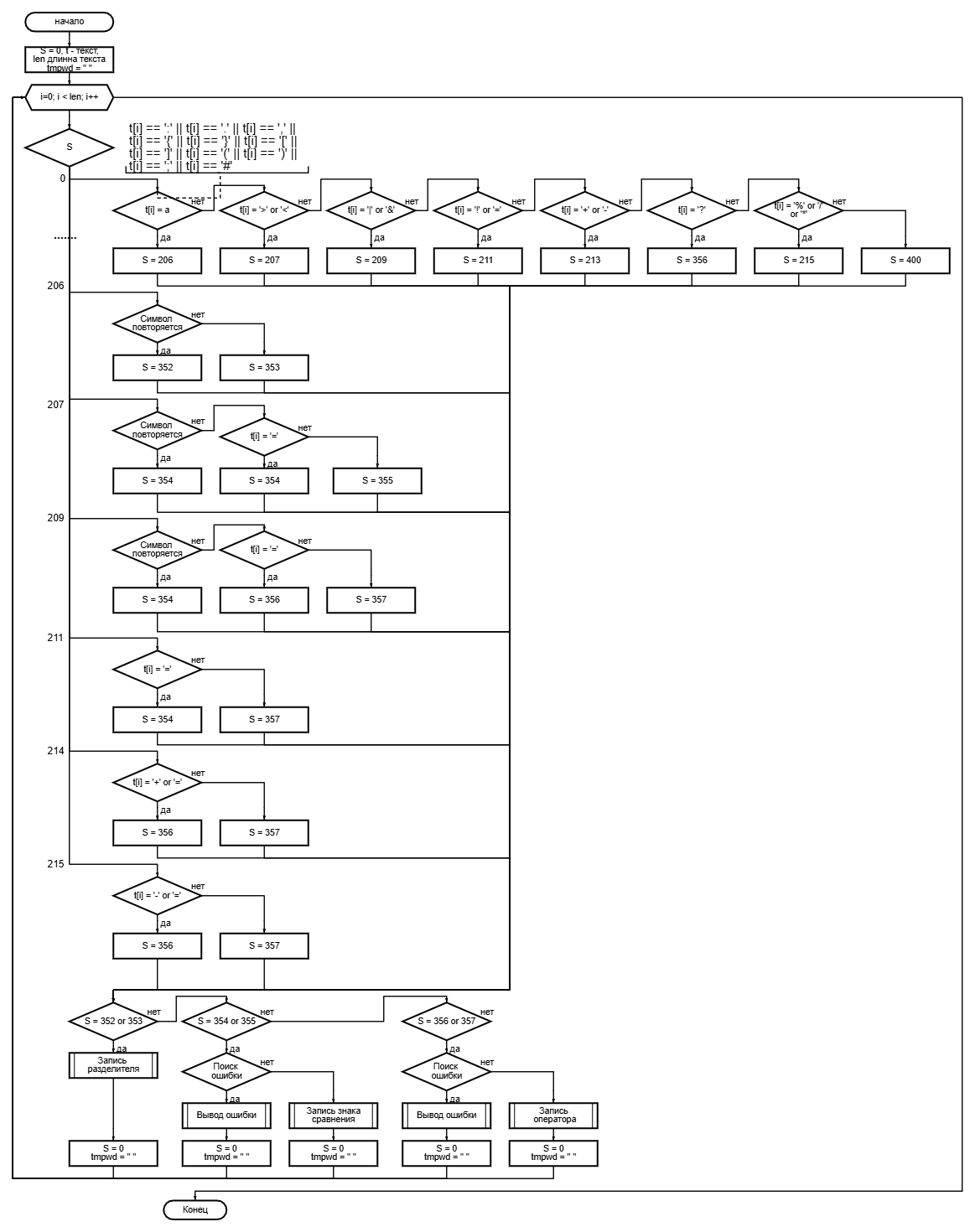


Рис 10. Блок-схема поиска операторов, знаков сравнения и разделителей

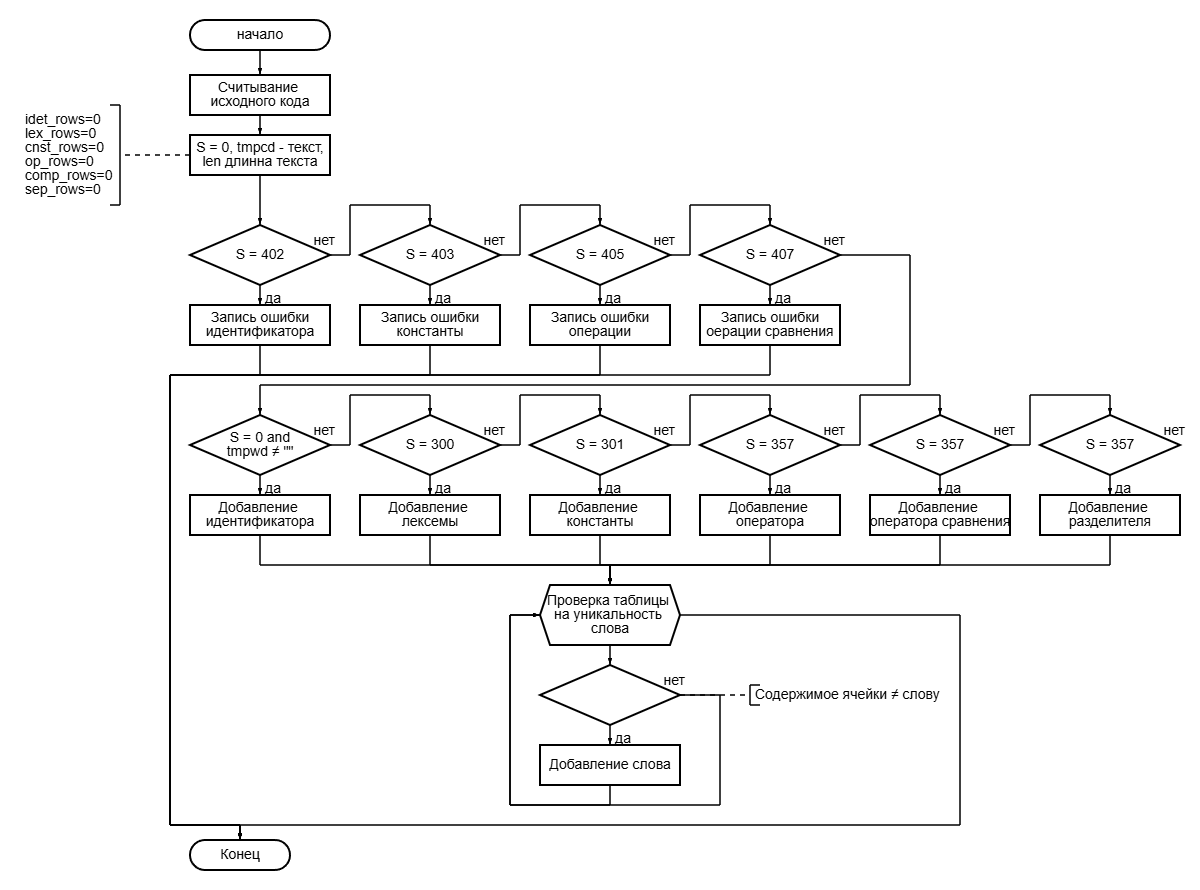


Рис 11. Блок-схема выбора лексем и ошибок

# 3.2 Оценка сложности алгоритма

По результатам анализа алгоритмов, описанных в разделе 4.1, можно сделать вывод о линейной сложности алгоритма программы, которая оценивается как O(n), поскольку все циклы, используемые в программе, имеют нулевой уровень вложенности.

Пояснения к этому выводу представлены ниже:

1) Функция удаления лишних символов содержит цикл for, которых имеет сложность O(n).

2) Функция лексического анализа состоит из одного цикла for и конструкции switch. Сложность конструкции switch составляет O(n), а суммарная сложность цикла for и switch равна O(n).

3) Итоговая сложность алгоритма программы оценивается как O(n).

# 3.3 Словесное описание работы алгоритма на примере разбора тестовых входных последовательностей

Для словесного описания работы алгоритма будут использоваться 2 последовательности:

1 – правильная:

double a = 8.7; /\*многострочный комментарий \*/

2 – неправильная:

2\_z12; // комментарий

Алгоритм работы для правильной последовательности:

1. Ввод исходного кода.

2. Считывание исходного кода.

3. Предпроцесс. Удаление комментариев, а также удаление лишних пробелов или переносов строк (стоящих подряд более чем одна штука, в том числе конструкции «пробел перенос» или «перенос пробел» или «перенос перенос», все заменяются на один перенос строки) и всей табуляции. Полученный результат далее называется «чистым кодом».

4. Посимвольный лексический анализ

4.1 – По умолчанию мы находимся в нулевом состоянии (S = 0;).

4.2 - В зависимости от взятого символа, происходит переход в соответствующее состояние, путём изменения значения переменной S, хранящей в себе номер текущего состояния.

4.3 - Обработка распознанных лексем

4.3.1 – После определения лексемы ключевого слова, идет проверка находится ли ключевое слово в таблице ключевых слов, если находится, то добавляется в таблицу ключевых слов (Те же действия проводятся с такими классами лексем как: идентификаторы, разделители, операторы, логические операторы, константы). После занесения слова в таблицу, мы переходим в нулевое состояние.

4.3.2 При возникновении ошибки производится вывод сообщения об ошибке в текстовое поле для ошибок.

4.3.3 Распознанные лексемы выводятся в таблицы пользовательского интерфейса в соответствии с их классификацией по таблицам.

5 - Производится вывод дескрипторного кода и псевдокода в пользовательский интерфейс.

5.1 По таблицам лексем находится лексеме в соответствии с исходным кодом.

5.2 Составление дескрипторного кода по найденной в таблице лексеме.

5.3 По составленному дескрипторному коду, создается псевдокод. Псевдо код каждой отдельной лексемы берется из таблиц.

4. Физическое проектирование программного обеспечения

В данном разделе описана спецификация программы.

4.1 Физическое проектирование

В данной программе были использованы как простые структуры данных – переменные, так и более сложные – контейнерный класс string для работы со строками. Описание этих структур данных можно найти в пункте 5.1.1.

Кроме того, в программе были применены различные функции, включая функцию для обработки исходной строки, функцию, реализующую работу лексического анализатора компилятора, а также функции для определения принадлежности к классу лексем. Спецификация этих функций представлена в пункте 5.1.2.

4.1.1 Выбор структур данных

В таблице 4 приведен список используемых в программе структур, данных и переменных.

Таблица 4

Структуры данных

|  |  |  |
| --- | --- | --- |
| Наименование | Обозначение | Тип |
| Переменная для хранения ссылки на поток выбранного файла | myStream | Stream |
| Имя выбранного файла | path | String |
| Кол-во строк в таблице ключ.слова | lex\_rows | int |
| Кол-во строк в таблице идентификаторы | idet\_rows | int |
| Кол-во строк в таблице разделители | sep\_rows | int |
| Кол-во строк в таблице константы | cnst\_rows | int |
| Кол-во строк в таблице знаки операций | op\_rows | int |
| Кол-во строк в таблице операторы сравнения | comp\_rows | int |
| Номер строки | line | int |
| Номер символа | i | int |
| Состояние конечного автомата | S | int |
| Длина последовательности | len | int |
| Переменная для хранения лексемы (кроме разделителей) | tmpwd | String |
| Переменная для хранения анализируемого текста. | tmpcd | String |

* + 1. Спецификация функций

Таблица 5

Спецификация функций

|  |  |  |
| --- | --- | --- |
| Заголовок функции | Формальные параметры | Выполняемое действие |
| 1 | 2 | 3 |
| bool isnum(wchar\_t tmp) | tmp – символ для проверки | Проверка является ли символ цифрой, кроме 0. |
| bool skip\_sings(wchar\_t tmp) | tmp – символ для проверки | Проверка является ли символ пробелом или переносом строки. |
| bool op\_comp(wchar\_t tmp) | tmp – символ для проверки | Проверка является ли символ оператором сравнения. |
| bool op\_sings(wchar\_t tmp) | tmp – символ для проверки | Проверка является ли символ знаком операции. |
| bool spec\_chr(wchar\_t tmp) | tmp – символ для проверки | Проверка является ли символ разделителем. |
| bool allchar(wchar\_t tmp) | tmp – символ для проверки | Проверка является ли символ допустимым для идентификатора. |
| add\_cnst(String^ wrd, int &rows) | Wrd – слово для добавления. Rows – количество заполненых строк в таблице. | Добавление нового элемента в таблицу констант. |
| add\_lex(String^ wrd, int &rows) | Wrd – слово для добавления. Rows – количество заполненых строк в таблице. | Добавление нового элемента в таблицу ключевых слов. |
| add\_idet(String^ wrd, int& rows) | Wrd – слово для добавления. Rows – количество заполненых строк в таблице. | Добавление нового элемента в таблицу идентификаторов. |
| add\_op\_s(String^ wrd, int& rows) | Wrd – слово для добавления. Rows – количество заполненых строк в таблице. | Добавление нового элемента в таблицу знаков операций. |
| add\_comp(String^ wrd, int& rows) | Wrd – слово для добавления. Rows – количество заполненых строк в таблице. | Добавление нового элемента в таблицу операторов сравнения. |
| add\_separ(String^ wrd, int& rows) | Wrd – слово для добавления. Rows – количество заполненых строк в таблице. | Добавление нового элемента в таблицу разделителей. |

Продолжение таблицы 5

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| add\_eror(int lines, int type) | Lines – строка на которой допущена ошибка. Type – тип ошибки. | Выводит в поле для ошибок, тип ошибки и номер строки, где она допущена. |
| prework() | - | Очистка исходного кода от лишних символов. |
| dop() | - | Нумерация строк в коде. |
| descriptor(  int lex\_rows,  int cnst\_rows,  int idet\_rows,  int op\_rows,  int comp\_rows,  int sep\_rows) | lex\_rows, cnst\_rows, idet\_rows, op\_rows, comp\_rows, sep\_rows – Количество строк в соответствующих таблицах. | Создание и вывод дескрипторного кода. |
| psevda() | - | Создание и вывод псевдокода. |
| cl() | - | Очистка. |

# Проектирование интерфейса

Описание интерфейса.

Слева находятся два больших текстовых поля. Первый для ввода своего исходного кода, второй для отображения текста после первоначальной обработки кода то есть, «чистого кода» (удаления лишних символов, табуляции, проставления номеров строк и прочее).

Справа сверху вниз: Одно широкое текстовое поле для вывода ошибок и два текстовых поля для дескрипторного и псевдокода. После шесть таблиц для вывода классов распознанных лексем.

В таблицах для классов лексем три столбца. Первый это номер строки в таблице и, совместив его с номером самой таблицы можно получить псевдокод этого символа. Второй используется для вывода самой лексемы, как она представлена в коде. В третьем столбце указан псевдокод, обозначающий данную лексему.

В левом нижнем углу отдельно расположено всё кнопки управления. Запуск программы лексического анализатора. Загрузка кода с файла.

В результате добавления данных компонентов получился интерфейс, показанный на рисунке 13.

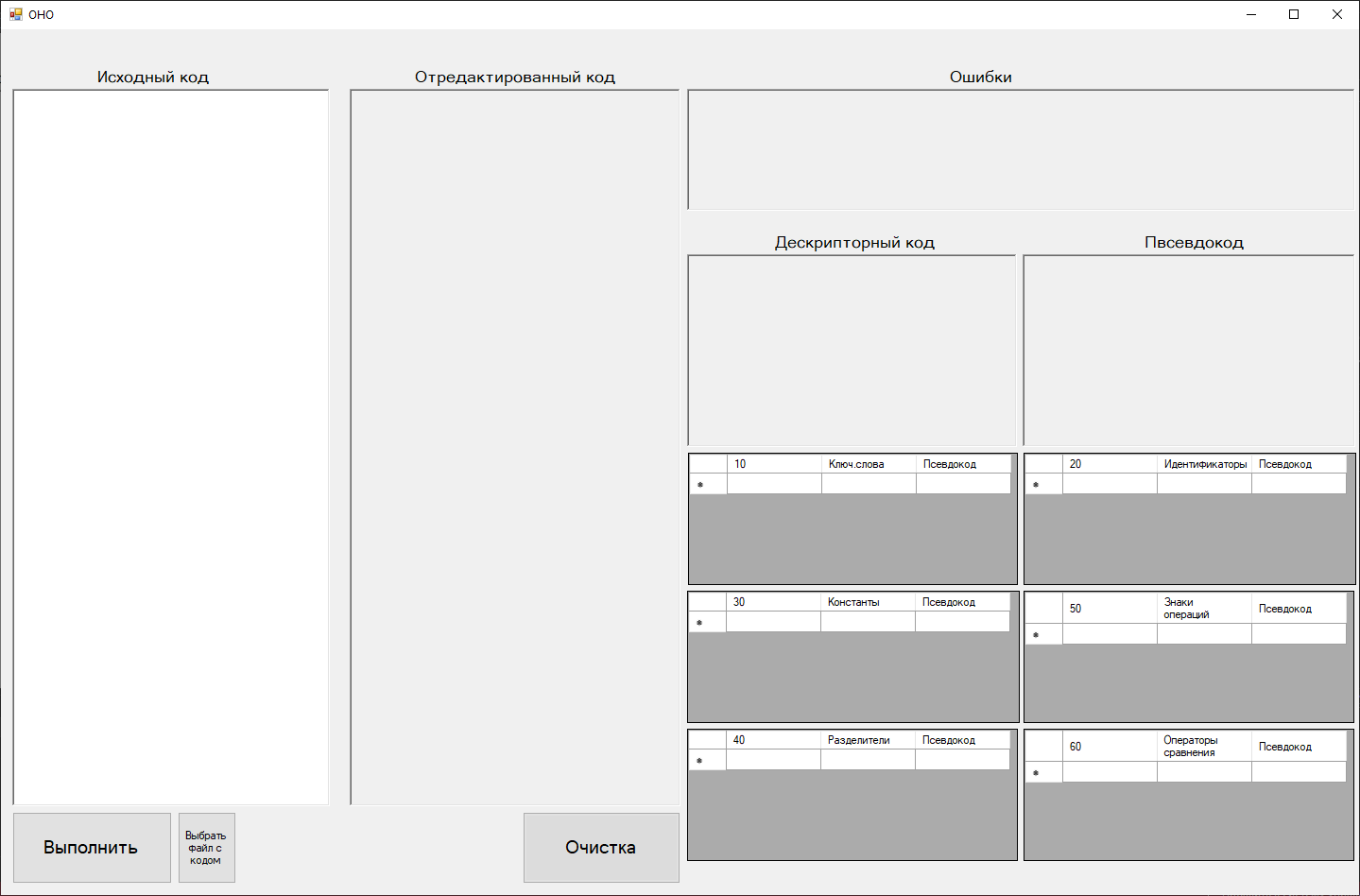


Рис 13. Интерфейс программы

# 6. Тестирование программного обеспечения

Результаты тестирования представлены на рисунках 13-20.

Исходный тестируемый код:

// 26; Оператор switch и for языка С++; Харламов Д.;

#include <iostream>

using namespace std;

// Вычисление и вывод суммы n-членов геометрической или арифметической прогрессии;

void main() {

int 1a, n;

double q, c;

cout << "Enter the arithmetic/geometric progression step = "; cin >> q; // Ввод шага прогрессии пользователем;

cout << "Enter Enter the items to be counted = "; cin >> n; // Ввод количества выводимых элементов пользователем;

cout << "Enter number: 1 - arithmetic progression; 2 - geometric progression = "; cin >> a;

/\*

Выбор пользователя каккую прогрессию использовать;

\*/

switch (a){

case 1: // если a = 1, то исполуйзуются формула арифметической прогрессии;

c = 0; // Выставления значения по умолчанию;

for (int i = 0; i < n; i++) { // Счетчик кол-во элементов выведено;

c = c + q; cout << c << endl; // Вычисление элемента прогрессии и его вывод;

}

break;

case 2: // если a = 2, то исполуйзуются формула агеометрической прогрессии;

c = 1; // Выставления значения по умолчанию;

for (int i = 0; i < n; i++) { // Счетчик кол-во элементов выведено;

c = c \* q; cout << c << endl; // Вычисление элемента прогрессии и его вывод;

}

break;

default: // Если ввели неверное число a;

cout << "Error wrong number"; // Вывод сообщения об ошибке;

break;

}

system("pause");

}

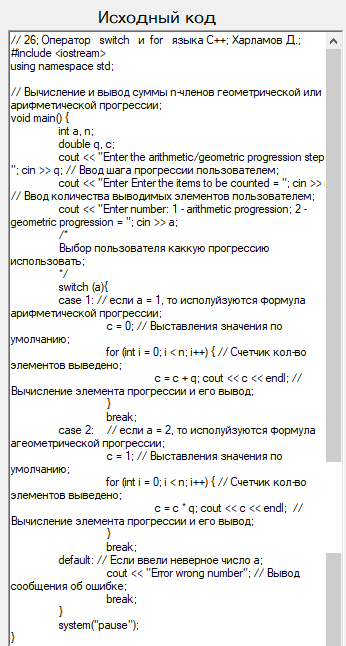


Рис. 13. Необработанный текст программы.

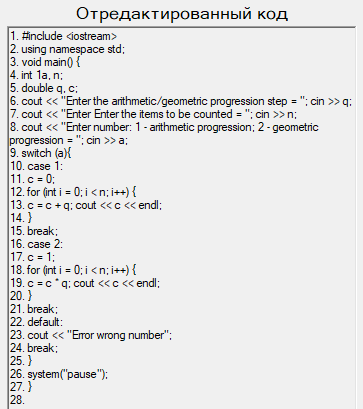


Рис. 14. Обработанный текст программы.

Результат:Были успешно удалены все комментарии и лишние символы.

Тестирование на определение классов лексем.

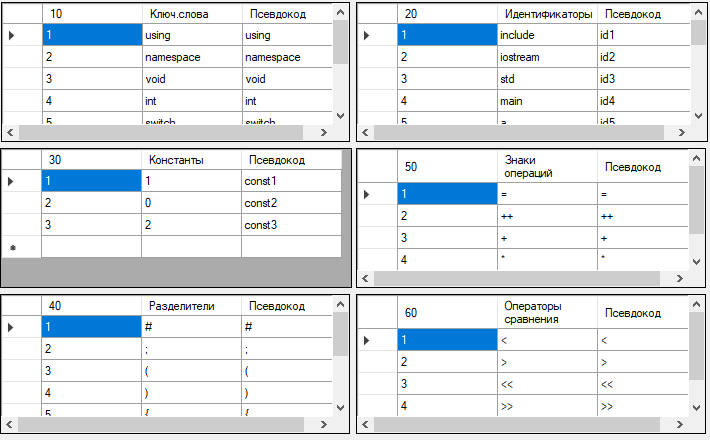


Рис. 15. Классы лексем.

Результат:все лексемы были определяются верно.

Тестирование на правильность вывода псевдо- и дескрипторного кодов:

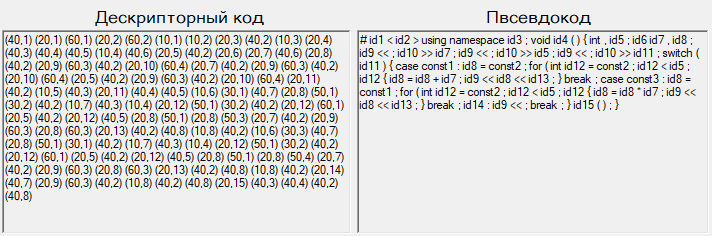


Рис. 16. Дескрипторный код и псевдокод.

Результат: псевдо- и дескрипторные кода выводятся правильно.

Тестирование на нахождение различных ошибок в коде:

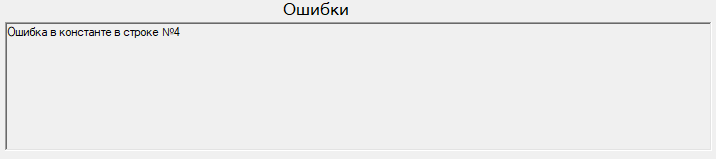


Рис. 37. Сообщение об ошибках.

Результат:ошибки были найдены и идентифицированы, верно.

Результаты тестирования программы представлены в таблице 5.

Таблица 5

Результаты тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата тестирования | Тестируемый модуль | Кто проводил тестирование | Описание теста | Результат тестирования |
| 1 | 2 | 3 | 4 | 5 |
| 20.05.2024 | TeorAvt.sln | Харламов Д.А. | Проверка удаления лишних символов и комментариев | Успех |
| 22.05.2024 | TeorAvt.sln | Новиков И.С. | Проверка классификации лексем | Успех |
| 25.05.2024 | TeorAvt.sln | Харламов Д.А. | Проверка вывода дескрипторного кода и псевдокода | Успех |
| 1.06.2024 | TeorAvt.sln | Белоус Г.П. | Проверка на нахождение различных ошибок в коде | Успех |

# 7. Заключение

При выполнении данной курсовой работы были проведены детальные анализы лексики и синтаксиса. Был создан конечный автомат, который эмулирует функционирование лексического анализатора компилятора для языка программирования C++, а также разработан алгоритм для программы, которая имитирует его работу, и определена сложность данного алгоритма.

Кроме того, были разработаны блок-схемы для алгоритма программы и оператора выбора switch, который используется в работе конечного автомата.

Была создана программа на основе разработанного конечного автомата, которая удаляет лишние символы из исходного кода, анализирует каждую лексему в заданной языковой конструкции, формирует таблицы дескрипторов и выводит дескрипторный и псевдокод. Кроме того, были разработаны регулярные грамматики для конечных автоматов и лексических и синтаксических анализаторов, а также контекстно-свободная грамматика, описывающая синтаксис инструкций языка программирования.

В итоге полученный код является рабочим и готовым к использованию в лексическом анализаторе. Результаты работы программы были предоставлены в качестве подтверждения ее функциональности.

# 8. Список литературы

1. Ганичева О.Г. Теория языков программирования и методы трансляции. [Текст] / / Ганичева О.Г.

2. Ершов, Е.В. Методика и организация самостоятельной работы: учебное пособие. [Текст] / Ершов Е.В., Виноградова Л.Н., Селивановских В.В. / / Череповец: ЧГУ, 2015.

3. Молчанов А.Ю. Системное программное обеспечение: лабораторный практикум [Электронный ресурс] / А.Ю. Молчанов. – URL: https: / /massolit.site /book /sistemnoe-programmnoe-obespechenie-laboratornij-praktikum /reading. (дата обращения: 19.04.2023.)

4. Типы формальных языков и грамматик. Классификация по Хомскому / / База и Генератор Образовательных Ресурсов. [Сайт] – URL: http: / /bigor.bmstu.ru /?cnt /?prn=y /?doc=LO-SAPR /gram04.mod (дата обращения: 25.04.2023).

5. Лаздин А. В. Формальные языки, грамматики, автоматы : учебное пособие [Электронный ресурс] / А. В. Лаздин. – URL: https: / /goo.su /jYfxq (дата обращения: 27.04.2023).

6. Лиогонький М.И., Береговая Т.А. Элементы теории конечных автоматов и регулярных языков [Электронный ресурс] / М.И. Лиогонький. – URL: https: / /goo.su /l3PzQCh (дата обращения: 27.04.2023).

7. Выхованец В. С. Теория автоматов : учебное пособие [Электронный ресурс] / В. С. Выхованец. – URL: https: / /goo.su /ipbcQ

8. Марышева Л.Т., Эргашев А. Системное программное обеспечение: Методическое пособие / Л.Т. Марышева. – URL: https: / /www.studmed.ru /gordeev-av-molchanov-ayu-sistemnoe-programmnoe-obespechenie\_7a0543bf78e.html

# 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                  ,

д. т. н., профессор                    Ершов Е. В.

«      »                       2024 г.

Разработка алгоритмического обеспечения и построение лексического анализатора компилятора

Техническое задание на курсовую работу

Листов 6

Руководитель    Пышницкий К.М.

    Ганичева О.Г.

ФИО преподавателя

Исполнитель:

Студент    1ПИб-02-1оп-22

группа

   Харламов Денис Алексеевич

Фамилия, имя, отчество

2024 год

Введение

В мире программирования компиляторы играют важную роль в создании программного обеспечения. Одина из главных составляющих компилятора – лексический анализатор, отвечающий за разбор текста на лексемы. Данная программа в ходе обработки исходного кода выделяет из текста все ключевые слова, операторы и остальные важные элементы, входящие в заданную языковую конструкцию.

1. Основания для разработки

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

Дата утверждения: 9 февраля 2024 года.

Наименование темы разработки: Разработка алгоритмического обеспечения и построение лексического анализатора компилятора.

2. Назначение разработки

Основная задача курсовой работы заключаются в практическом закреплении освоения материала, полученного в процессе изучения дисциплин «Теория автоматов и формальных языков» и «Структуры и алгоритмы обработки данных», а также разработка алгоритмического обеспечения, построение лексического анализатора компилятора С++ и конечного автомата. Такая программа разрабатывается с целью моделирования работы лексического анализатора компилятора и решения следующих задач: выделение лексем, удаление лишних пробелов и комментариев, а также подсчёт количества строк в программе.

3. Требования к программе

3.1. Требования к функциональным характеристикам

Программа, моделирующая работу лексического анализатора по варианту №26 (Оператор switch и for языка С++), должна выполнять следующие функции:

1. Выделять из текста входной программы все лексемы и сортировать их по таблицам.
2. Удалять лишние пробелы и комментарии из входной строки.
3. Подсчитывать количество строк в программе.
4. Формировать дескрипторный код и псевдокод на выходе
5. Диагностика и локализация лексических ошибок;

3.2. Специальные требования

1. Построить конечный автомат для моделирования работы этапа лексического анализа компилятора.
2. Построить регулярную грамматику, соответствующую конечному автомату.
3. Построить КС-грамматику, описывающую синтаксис инструкции языка программирования.
4. Составить дерево вывода и соответствующее ему левостороннее порождение;
5. Разработать алгоритмическое обеспечение для программы, моделирующей работу лексического анализатора (блок-схемы алгоритмов, словесное описание алгоритмов).
6. Определить и обосновать сложность полученного алгоритма.

3.3. Требования к надежности

Программа должна обрабатывать все символы в тексте исходной программы без ошибок.

3.4. Условия эксплуатации

Правила эксплуатации компьютера в закрытом помещении:

1. Компьютер должен находиться на ровной поверхности, защищенной от пыли и вибрации.
2. Окружающая температура не должна превышать 35 градусов и не должна падать ниже 10 градусов.
3. Влажность в помещении должна быть не более 60%.
4. Компьютер должен быть подключен к надежному источнику питания.
5. В помещении не должно быть никаких вспышек или сильных электромагнитных помех.

3.5. Требования к составу и параметрам технических средств

Программа должна корректно работать при соблюдении минимальных системных требований:

1. Процессор с тактовой частотой не менее 1 ГГц.
2. Оперативная память не менее 512 Мб.
3. Видеокарта, поддерживающая DirectX 9 совместимость и минимум 128 Мб видеопамяти.
4. Свободное место на жестком диске не менее 50 Мб.
5. Устройства ввода: клавиатура и мышь.

3.6. Требования к информационной и программной совместимости

Код программы реализуется на языке C++ в среде разработки Visual Studio 2022. Данный программный продукт может использоваться в операционных системах Windows 7 / 10 при наличии установленного набора библиотек и системных компонентов - .NET Framework 4.5 или выше.

3.7. Требования к маркировке и упаковке

Программа должна быть удобно скомпонована, чтобы её можно было легко распаковать и запустить на компьютере пользователя.

3.8. Требования к транспортированию и хранению

Созданный проект будет помещен в папку, где будет храниться, а также будет записан на отдельный информационный накопитель для возможности восстановления.

4. Стадии и этапы разработки

В данном пункте в табличном виде представлена информация о стадиях и этапах разработки приложения (табл.П1.1).

Таблица П1.1.

Стадии и этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование  этапа разработки | Сроки разработки | Результат выполнения | Отметка о выполнении |
| 1 | 2 | 3 | 4 |
| Поиск необходимой информации по предметной области | 25.04.24-29.04.24 | Найдены необходимые материалы | Выполнено |
| Реализация функции обработки текста лексического анализатора | 05.05.24-07.05.24 | Реализованная функция обработки текста лексического анализатора | Выполнено |
| Реализация функции для распознавания нужных слов и классификации их | 07.04.24-09.04.24 | Готовая функция для распознавания заданных слов и классификация их | Выполнено |

Продолжение таблицы П1.1.

|  |  |  |  |
| --- | --- | --- | --- |
| Реализация алгоритма для построения дескрипторного кода | 09.05.24-12.05.24 | Реализован алгоритм для построения дескрипторного кода | Выполнено |
| Реализация функции для распознавания и обработки ошибок | 12.05.2024-15.05.2024 | Реализована функция распознания и обработки ошибок | Выполнено |
| Построение КС-грамматики, дерево вывода и порождение | 15.05.2024-27.05.2024 | Построена КС-грамматика древо вывода и порождение | Выполнено |

5. Порядок контроля и приемки

В данном пункте в табличном виде представлена информация о порядке и контроля приемки курсовой работы (табл.П1.2).

Таблица П1.2.

Порядок контроля и приемки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование  контрольного этапа  выполнения  курсовой работы | Сроки  контроля | Результат выполнения | Отметка о приемке  результата  контрольного этапа |
| Создание технического задания | 10.04.24 | Готовое техническое задание | Выполнено |
| Демонстрация программы | 21.05.24 | Готовая конечная программа | Выполнено |
| Создание расчётно-пояснительной записки | 30.05.24 – 07.06.24 | Готовая РПЗ | Выполнено |
| Защита курсовой работы | 07.06.24 – 13.06.24 | Итоговая оценка за курсовую работу |  |

# Приложение 2. Текст программы

#pragma once

#include <string>

namespace A1 {

using namespace std;

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Сводка для MyForm

/// </summary>

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

{

public:

MyForm(void)

{

InitializeComponent();

}

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

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

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

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

private:System::ComponentModel::Container^ components;

private: System::Windows::Forms::DataGridView^ t\_lexme;

private: System::Windows::Forms::DataGridView^ t\_cnst;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Clex1;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Clex2;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Ccnst1;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Ccnst2;

private: System::Windows::Forms::DataGridView^ t\_id;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Cidet1;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Cidet2;

private: System::Windows::Forms::DataGridView^ t\_op\_sings;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn1;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn2;

private: System::Windows::Forms::DataGridView^ t\_op\_comp;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn3;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn4;

private: System::Windows::Forms::DataGridView^ t\_sep;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn5;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn6;

private: System::Windows::Forms::RichTextBox^ pseTextBox3;

private: System::Windows::Forms::RichTextBox^ desTextBox4;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Clex3;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Ccnst;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Cidet;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column1;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column2;

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

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::Button^ button3;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column3;

protected:

~MyForm()

{

if (components)

{

delete components;

}

}

private:

bool isnum(wchar\_t tmp)

{

return (tmp == '1' || tmp == '2' || tmp == '3' || tmp == '4' || tmp == '5' || tmp == '6' || tmp == '7' || tmp == '8' || tmp == '9') ? true : false;

}

bool skip\_sings(wchar\_t tmp)

{

return (tmp == ' ' || tmp == '\n' || tmp == '\r') ? true : false;

}

bool op\_comp(wchar\_t tmp)

{

return (tmp == '<' || tmp == '>' || tmp == '=' || tmp == '!') ? true : false;

}

bool op\_sings(wchar\_t tmp)

{

return (tmp == '=' || tmp == '-' || tmp == '+' || tmp == '\*' || tmp == '/' || tmp == '%' || tmp == '|' || tmp == '&') ? true : false;

}

bool spec\_chr(wchar\_t tmp)

{

return (tmp == ';' || tmp == '(' || tmp == ')' || tmp == '{' || tmp == '}' || tmp == ':' || tmp == '#' || tmp == ',' || tmp == '[' || tmp == ']' || tmp == '.') ? true : false;

}

bool stop\_sing(wchar\_t tmp)

{

return (spec\_chr(tmp) || op\_comp(tmp) || op\_sings(tmp) || skip\_sings(tmp)) ? true : false;

}

bool allchar(wchar\_t tmp)

{

return (tmp == '\_' || tmp == 'a' || tmp == 'b' || tmp == 'c' || tmp == 'd' || tmp == 'e' || tmp == 'f' || tmp == 'g' || tmp == 'h' || tmp == 'i' || tmp == 'j' || tmp == 'k' || tmp == 'l' || tmp == 'm'

|| tmp == 'n' || tmp == 'o' || tmp == 'p' || tmp == 'q' || tmp == 'r' || tmp == 's' || tmp == 't' || tmp == 'u' || tmp == 'v' || tmp == 'w' || tmp == 'x' || tmp == 'y' || tmp == 'z' || tmp == 'A'

|| tmp == 'B' || tmp == 'C' || tmp == 'D' || tmp == 'E' || tmp == 'F' || tmp == 'G' || tmp == 'H' || tmp == 'I' || tmp == 'J' || tmp == 'K' || tmp == 'L' || tmp == 'M' || tmp == 'N' || tmp == 'O'

|| tmp == 'P' || tmp == 'Q' || tmp == 'R' || tmp == 'S' || tmp == 'T' || tmp == 'U' || tmp == 'V' || tmp == 'W' || tmp == 'X' || tmp == 'Y' || tmp == 'Z' || isnum(tmp) || tmp == '0') ? true : false;

}

bool search\_c(String^ wrd, int rows) {

for (int j = 0; j <= rows - 1; j++) {

if (Convert::ToString(t\_cnst->Rows[j]->Cells[1]->Value) == wrd) return true;

}

return false;

}

bool search\_l(String^ wrd, int rows) {

for (int j = 0; j <= rows - 1; j++) {

if (Convert::ToString(t\_lexme->Rows[j]->Cells[1]->Value) == wrd) return true;

}

return false;

}

bool search\_id(String^ wrd, int rows) {

for (int j = 0; j <= rows - 1; j++) {

if (Convert::ToString(t\_id->Rows[j]->Cells[1]->Value) == wrd) return true;

}

return false;

}

bool search\_op\_s(String^ wrd, int rows) {

for (int j = 0; j <= rows - 1; j++) {

if (Convert::ToString(t\_op\_sings->Rows[j]->Cells[1]->Value) == wrd) return true;

}

return false;

}

bool search\_comp(String^ wrd, int rows) {

for (int j = 0; j <= rows - 1; j++) {

if (Convert::ToString(t\_op\_comp->Rows[j]->Cells[1]->Value) == wrd) return true;

}

return false;

}

bool search\_separ(String^ wrd, int rows) {

for (int j = 0; j <= rows - 1; j++) {

if (Convert::ToString(t\_sep->Rows[j]->Cells[1]->Value) == wrd) return true;

}

return false;

}

void add\_cnst(String^ wrd, int &rows)

{

if (!search\_c(wrd, rows)) {

t\_cnst->RowCount = rows + 1;

t\_cnst->Rows[rows - 1]->Cells[0]->Value = Convert::ToString(rows);

t\_cnst->Rows[rows - 1]->Cells[1]->Value = wrd;

t\_cnst->Rows[rows - 1]->Cells[2]->Value = "const" + Convert::ToString(rows);

rows++;

}

}

void add\_lex(String^ wrd, int &rows)

{

if (!search\_l(wrd, rows)) {

t\_lexme->RowCount = rows + 1;

t\_lexme->Rows[rows - 1]->Cells[0]->Value = Convert::ToString(rows);

t\_lexme->Rows[rows - 1]->Cells[1]->Value = wrd;

t\_lexme->Rows[rows - 1]->Cells[2]->Value = wrd;

rows++;

}

}

void add\_idet(String^ wrd, int& rows)

{

if (!search\_id(wrd, rows)) {

t\_id->RowCount = rows + 1;

t\_id->Rows[rows - 1]->Cells[0]->Value = Convert::ToString(rows);

t\_id->Rows[rows - 1]->Cells[1]->Value = wrd;

t\_id->Rows[rows - 1]->Cells[2]->Value = "id" + Convert::ToString(rows);

rows++;

}

}

void add\_op\_s(String^ wrd, int& rows)

{

if (!search\_op\_s(wrd, rows)) {

t\_op\_sings->RowCount = rows + 1;

t\_op\_sings->Rows[rows - 1]->Cells[0]->Value = Convert::ToString(rows);

t\_op\_sings->Rows[rows - 1]->Cells[1]->Value = wrd;

t\_op\_sings->Rows[rows - 1]->Cells[2]->Value = wrd;

rows++;

}

}

void add\_comp(String^ wrd, int& rows)

{

if (!search\_comp(wrd, rows)) {

t\_op\_comp->RowCount = rows + 1;

t\_op\_comp->Rows[rows - 1]->Cells[0]->Value = Convert::ToString(rows);

t\_op\_comp->Rows[rows - 1]->Cells[1]->Value = wrd;

t\_op\_comp->Rows[rows - 1]->Cells[2]->Value = wrd;

rows++;

}

}

void add\_separ(String^ wrd, int& rows)

{

if (!search\_separ(wrd, rows)) {

t\_sep->RowCount = rows + 1;

t\_sep->Rows[rows - 1]->Cells[0]->Value = Convert::ToString(rows);

t\_sep->Rows[rows - 1]->Cells[1]->Value = wrd;

t\_sep->Rows[rows - 1]->Cells[2]->Value = wrd;

rows++;

}

}

void add\_eror(int lines, int type)

{

String^ tmp = "";

if (type == 0) tmp = "в идентификаторе";

else if (type == 1) tmp = "в константе";

else if (type == 2) tmp = "в знаке операции";

else if (type == 3) tmp = "в знаке операции сравнения";

else if (type == 4) tmp = "не закрыт многострочный комментарий";

ErorBox->Text += "Ошибка " + tmp + " в строке №" + (lines) + "\n";

}

#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->richTextBox1 = (gcnew System::Windows::Forms::RichTextBox());

this->richTextBox2 = (gcnew System::Windows::Forms::RichTextBox());

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

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

this->t\_lexme = (gcnew System::Windows::Forms::DataGridView());

this->Clex1 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Clex2 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Clex3 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->t\_cnst = (gcnew System::Windows::Forms::DataGridView());

this->Ccnst1 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Ccnst2 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Ccnst = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->t\_id = (gcnew System::Windows::Forms::DataGridView());

this->Cidet1 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Cidet2 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Cidet = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->t\_op\_sings = (gcnew System::Windows::Forms::DataGridView());

this->dataGridViewTextBoxColumn1 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dataGridViewTextBoxColumn2 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Column1 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->t\_op\_comp = (gcnew System::Windows::Forms::DataGridView());

this->dataGridViewTextBoxColumn3 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dataGridViewTextBoxColumn4 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

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

this->t\_sep = (gcnew System::Windows::Forms::DataGridView());

this->dataGridViewTextBoxColumn5 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dataGridViewTextBoxColumn6 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

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

this->pseTextBox3 = (gcnew System::Windows::Forms::RichTextBox());

this->desTextBox4 = (gcnew System::Windows::Forms::RichTextBox());

this->ErorBox = (gcnew System::Windows::Forms::RichTextBox());

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->button3 = (gcnew System::Windows::Forms::Button());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->t\_lexme))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->t\_cnst))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->t\_id))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->t\_op\_sings))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->t\_op\_comp))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->t\_sep))->BeginInit();

this->SuspendLayout();

//

// richTextBox1

//

this->richTextBox1->Location = System::Drawing::Point(12, 63);

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

this->richTextBox1->Size = System::Drawing::Size(336, 759);

this->richTextBox1->TabIndex = 15;

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

//

// richTextBox2

//

this->richTextBox2->Location = System::Drawing::Point(369, 63);

this->richTextBox2->Name = L"richTextBox2";

this->richTextBox2->ReadOnly = true;

this->richTextBox2->Size = System::Drawing::Size(350, 759);

this->richTextBox2->TabIndex = 16;

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

//

// button1

//

this->button1->Font = (gcnew System::Drawing::Font(L"Segoe UI Symbol", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->button1->Location = System::Drawing::Point(12, 828);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(169, 76);

this->button1->TabIndex = 17;

this->button1->Text = L"Выполнить";

this->button1->UseVisualStyleBackColor = true;

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

//

// button2

//

this->button2->BackColor = System::Drawing::Color::Gainsboro;

this->button2->Font = (gcnew System::Drawing::Font(L"Segoe UI Symbol", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->button2->Location = System::Drawing::Point(552, 828);

this->button2->Name = L"button2";

this->button2->Size = System::Drawing::Size(167, 76);

this->button2->TabIndex = 20;

this->button2->Text = L"Очистка";

this->button2->UseVisualStyleBackColor = false;

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

//

// t\_lexme

//

this->t\_lexme->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->t\_lexme->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->Clex1, this->Clex2,

this->Clex3

});

this->t\_lexme->Location = System::Drawing::Point(727, 448);

this->t\_lexme->Name = L"t\_lexme";

this->t\_lexme->ReadOnly = true;

this->t\_lexme->Size = System::Drawing::Size(349, 140);

this->t\_lexme->TabIndex = 21;

//

// Clex1

//

this->Clex1->HeaderText = L"10";

this->Clex1->Name = L"Clex1";

this->Clex1->ReadOnly = true;

//

// Clex2

//

this->Clex2->HeaderText = L"Ключ.слова";

this->Clex2->Name = L"Clex2";

this->Clex2->ReadOnly = true;

//

// Clex3

//

this->Clex3->HeaderText = L"Псевдокод";

this->Clex3->Name = L"Clex3";

this->Clex3->ReadOnly = true;

//

// t\_cnst

//

this->t\_cnst->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->t\_cnst->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->Ccnst1, this->Ccnst2,

this->Ccnst

});

this->t\_cnst->Location = System::Drawing::Point(726, 594);

this->t\_cnst->Name = L"t\_cnst";

this->t\_cnst->ReadOnly = true;

this->t\_cnst->Size = System::Drawing::Size(352, 140);

this->t\_cnst->TabIndex = 22;

//

// Ccnst1

//

this->Ccnst1->HeaderText = L"30";

this->Ccnst1->Name = L"Ccnst1";

this->Ccnst1->ReadOnly = true;

//

// Ccnst2

//

this->Ccnst2->HeaderText = L"Константы";

this->Ccnst2->Name = L"Ccnst2";

this->Ccnst2->ReadOnly = true;

//

// Ccnst

//

this->Ccnst->HeaderText = L"Псевдокод";

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

this->Ccnst->ReadOnly = true;

//

// t\_id

//

this->t\_id->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->t\_id->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->Cidet1, this->Cidet2,

this->Cidet

});

this->t\_id->Location = System::Drawing::Point(1082, 448);

this->t\_id->Name = L"t\_id";

this->t\_id->Size = System::Drawing::Size(352, 140);

this->t\_id->TabIndex = 23;

//

// Cidet1

//

this->Cidet1->HeaderText = L"20";

this->Cidet1->Name = L"Cidet1";

//

// Cidet2

//

this->Cidet2->HeaderText = L"Идентификаторы";

this->Cidet2->Name = L"Cidet2";

//

// Cidet

//

this->Cidet->HeaderText = L"Псевдокод";

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

//

// t\_op\_sings

//

this->t\_op\_sings->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->t\_op\_sings->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->dataGridViewTextBoxColumn1,

this->dataGridViewTextBoxColumn2, this->Column1

});

this->t\_op\_sings->Location = System::Drawing::Point(1082, 594);

this->t\_op\_sings->Name = L"t\_op\_sings";

this->t\_op\_sings->ReadOnly = true;

this->t\_op\_sings->Size = System::Drawing::Size(350, 140);

this->t\_op\_sings->TabIndex = 24;

//

// dataGridViewTextBoxColumn1

//

this->dataGridViewTextBoxColumn1->HeaderText = L"50";

this->dataGridViewTextBoxColumn1->Name = L"dataGridViewTextBoxColumn1";

this->dataGridViewTextBoxColumn1->ReadOnly = true;

//

// dataGridViewTextBoxColumn2

//

this->dataGridViewTextBoxColumn2->HeaderText = L"Знаки операций";

this->dataGridViewTextBoxColumn2->Name = L"dataGridViewTextBoxColumn2";

this->dataGridViewTextBoxColumn2->ReadOnly = true;

//

// Column1

//

this->Column1->HeaderText = L"Псевдокод";

this->Column1->Name = L"Column1";

this->Column1->ReadOnly = true;

//

// t\_op\_comp

//

this->t\_op\_comp->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->t\_op\_comp->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->dataGridViewTextBoxColumn3,

this->dataGridViewTextBoxColumn4, this->Column2

});

this->t\_op\_comp->Location = System::Drawing::Point(1082, 740);

this->t\_op\_comp->Name = L"t\_op\_comp";

this->t\_op\_comp->ReadOnly = true;

this->t\_op\_comp->Size = System::Drawing::Size(350, 140);

this->t\_op\_comp->TabIndex = 25;

//

// dataGridViewTextBoxColumn3

//

this->dataGridViewTextBoxColumn3->HeaderText = L"60";

this->dataGridViewTextBoxColumn3->Name = L"dataGridViewTextBoxColumn3";

this->dataGridViewTextBoxColumn3->ReadOnly = true;

//

// dataGridViewTextBoxColumn4

//

this->dataGridViewTextBoxColumn4->HeaderText = L"Операторы сравнения";

this->dataGridViewTextBoxColumn4->Name = L"dataGridViewTextBoxColumn4";

this->dataGridViewTextBoxColumn4->ReadOnly = true;

//

// Column2

//

this->Column2->HeaderText = L"Псевдокод";

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

this->Column2->ReadOnly = true;

//

// t\_sep

//

this->t\_sep->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->t\_sep->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->dataGridViewTextBoxColumn5,

this->dataGridViewTextBoxColumn6, this->Column3

});

this->t\_sep->Location = System::Drawing::Point(726, 740);

this->t\_sep->Name = L"t\_sep";

this->t\_sep->ReadOnly = true;

this->t\_sep->Size = System::Drawing::Size(350, 140);

this->t\_sep->TabIndex = 26;

//

// dataGridViewTextBoxColumn5

//

this->dataGridViewTextBoxColumn5->HeaderText = L"40";

this->dataGridViewTextBoxColumn5->Name = L"dataGridViewTextBoxColumn5";

this->dataGridViewTextBoxColumn5->ReadOnly = true;

//

// dataGridViewTextBoxColumn6

//

this->dataGridViewTextBoxColumn6->HeaderText = L"Разделители";

this->dataGridViewTextBoxColumn6->Name = L"dataGridViewTextBoxColumn6";

this->dataGridViewTextBoxColumn6->ReadOnly = true;

//

// Column3

//

this->Column3->HeaderText = L"Псевдокод";

this->Column3->Name = L"Column3";

this->Column3->ReadOnly = true;

//

// pseTextBox3

//

this->pseTextBox3->Location = System::Drawing::Point(1081, 238);

this->pseTextBox3->Name = L"pseTextBox3";

this->pseTextBox3->ReadOnly = true;

this->pseTextBox3->Size = System::Drawing::Size(352, 204);

this->pseTextBox3->TabIndex = 27;

this->pseTextBox3->Text = L"";

//

// desTextBox4

//

this->desTextBox4->ImeMode = System::Windows::Forms::ImeMode::NoControl;

this->desTextBox4->Location = System::Drawing::Point(726, 238);

this->desTextBox4->Name = L"desTextBox4";

this->desTextBox4->ReadOnly = true;

this->desTextBox4->Size = System::Drawing::Size(349, 204);

this->desTextBox4->TabIndex = 28;

this->desTextBox4->Text = L"";

//

// ErorBox

//

this->ErorBox->Location = System::Drawing::Point(726, 63);

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

this->ErorBox->ReadOnly = true;

this->ErorBox->Size = System::Drawing::Size(707, 129);

this->ErorBox->TabIndex = 29;

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

//

// label1

//

this->label1->AutoSize = true;

this->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 12.4F));

this->label1->Location = System::Drawing::Point(98, 40);

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

this->label1->Size = System::Drawing::Size(127, 20);

this->label1->TabIndex = 30;

this->label1->Text = L"Исходный код";

//

// label2

//

this->label2->AutoSize = true;

this->label2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 12.4F));

this->label2->Location = System::Drawing::Point(434, 40);

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

this->label2->Size = System::Drawing::Size(221, 20);

this->label2->TabIndex = 31;

this->label2->Text = L"Отредактированный код";

//

// label3

//

this->label3->AutoSize = true;

this->label3->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 12.4F));

this->label3->Location = System::Drawing::Point(1000, 40);

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

this->label3->Size = System::Drawing::Size(75, 20);

this->label3->TabIndex = 32;

this->label3->Text = L"Ошибки";

//

// label4

//

this->label4->AutoSize = true;

this->label4->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 12.4F));

this->label4->Location = System::Drawing::Point(815, 215);

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

this->label4->Size = System::Drawing::Size(178, 20);

this->label4->TabIndex = 33;

this->label4->Text = L"Дескрипторный код";

//

// label5

//

this->label5->AutoSize = true;

this->label5->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 12.4F));

this->label5->Location = System::Drawing::Point(1206, 215);

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

this->label5->Size = System::Drawing::Size(114, 20);

this->label5->TabIndex = 34;

this->label5->Text = L"Пвсевдокод";

//

// button3

//

this->button3->Location = System::Drawing::Point(187, 828);

this->button3->Name = L"button3";

this->button3->Size = System::Drawing::Size(62, 76);

this->button3->TabIndex = 35;

this->button3->Text = L"Выбрать файл с кодом";

this->button3->UseVisualStyleBackColor = true;

this->button3->Click += gcnew System::EventHandler(this, &MyForm::button3\_Click);

//

// MyForm

//

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

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(1437, 916);

this->Controls->Add(this->button3);

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->ErorBox);

this->Controls->Add(this->desTextBox4);

this->Controls->Add(this->pseTextBox3);

this->Controls->Add(this->t\_sep);

this->Controls->Add(this->t\_op\_comp);

this->Controls->Add(this->t\_op\_sings);

this->Controls->Add(this->t\_id);

this->Controls->Add(this->t\_cnst);

this->Controls->Add(this->t\_lexme);

this->Controls->Add(this->button2);

this->Controls->Add(this->button1);

this->Controls->Add(this->richTextBox2);

this->Controls->Add(this->richTextBox1);

this->Name = L"MyForm";

this->Text = L"ОНО";

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->t\_lexme))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->t\_cnst))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->t\_id))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->t\_op\_sings))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->t\_op\_comp))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->t\_sep))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

public:

int S = 0, a = 0; //S - состояние автомата, a - количество удаленных элементов

int tmpid; //временная переменная для обозначения начального элемента удаления

void prework()

{

richTextBox1->Text += "\n";

String^ tmpcd = richTextBox1->Text; //обрабатываемый программой текстовый массив

int line = 0;

for (int i = 0; i < richTextBox1->Text->Length; i++) {

switch (S) {

case 0:

switch (richTextBox1->Text[i]) {

case '/':

S = 2;

break;

case ' ': //пробел

S = 1;

break;

case '\t':

tmpcd = tmpcd->Remove(i - a, 1); //удаление табуляции

a += 1;

break;

case '\n': //перенос

line++;

S = 6;

break;

}

break;

case 1:

switch (richTextBox1->Text[i]) {

case ' ': //пробел повторяется вновь

tmpcd = tmpcd->Remove(i - a, 1);

a += 1;

break;

case '/':

S = 2;

break;

default:

S = 0;

break;

}

break;

case 2:

switch (richTextBox1->Text[i]) {

case '/': //второй слэш - однострочный

tmpid = i - 1; //начальный элемент, с которого осуществляется удаление

S = 3;

break;

case '\*': //звездочка полсе слэша - начало многострочного

tmpid = i - 1;

S = 4;

break;

default:

S = 0;

break;

}

break;

case 3:

switch (richTextBox1->Text[i]) {

case '\n': //переход на новую строку после однострочного комментария

line++;

tmpcd = tmpcd->Remove(tmpid - a, i - tmpid);

a += i - tmpid;

if (tmpid == 0 || tmpcd[i - a - 1] == '\n') {

tmpcd = tmpcd->Remove(i - a, 1);

a += 1;

}

S = 0;

break;

}

break;

case 4:

switch (richTextBox1->Text[i]) {

case '\*': //вторая звездочка - многострочный комментарий заканчивается

S = 5;

break;

}

break;

case 5:

switch (richTextBox1->Text[i]) {

case '/': //слэш после звезды - конец многострочного комментария

tmpcd = tmpcd->Remove(tmpid - a, i - tmpid + 2);

a += i - tmpid + 2;

S = 0;

break;

default: //возврат

S = 4;

break;

}

break;

case 6:

switch (richTextBox1->Text[i]) {

case '\n':

tmpcd = tmpcd->Remove(i - a, 1);

a += 1;

break;

case '/':

S = 2;

break;

case '\t': // табуляция

tmpcd = tmpcd->Remove(i - a, 1);

a += 1;

break;

default: // сброс

S = 0;

break;

}

break;

}

}

if (S == 4) {

tmpcd = tmpcd->Remove(tmpid - a, richTextBox1->Text->Length - tmpid);

add\_eror(line, 4);

}

richTextBox2->Text = tmpcd;

}

void dop()

{

String^ tmpcd = richTextBox2->Text;

int a = 1;

for (int i = 0; i < tmpcd->Length; i++) {

if (i == 0) {

tmpcd = tmpcd->Insert(i, a.ToString() + ". ");

a++;

}

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

tmpcd = tmpcd->Insert(i + 1, a.ToString() + ". ");

a++;

}

}

richTextBox2->Text = tmpcd;

}

void work(int &lex\_rows, int &cnst\_rows, int &idet\_rows, int &op\_rows, int &comp\_rows, int &sep\_rows)

{

int S = 0, line = 1;

String^ tmpcd = richTextBox2->Text;

String^ tmpwd = "";

for (int i = 0; i < richTextBox2->Text->Length; i++) {

switch (S) {

case 0:

if (skip\_sings(tmpcd[i])) S = 0;

else if (isnum(tmpcd[i])) S = 1;

else if (tmpcd[i] == '"') S = 351;

else if (tmpcd[i] == Convert::ToChar("'")) S = 399;

else if (tmpcd[i] == '0') S = 2;

else if (tmpcd[i] == 'a') S = 7;

else if (tmpcd[i] == 'b') S = 12;

else if (tmpcd[i] == 'c') S = 23;

else if (tmpcd[i] == 'd') S = 218;

else if (tmpcd[i] == 'e') S = 50;

else if (tmpcd[i] == 'f') S = 59;

else if (tmpcd[i] == 'i') S = 67;

else if (tmpcd[i] == 'n') S = 73;

else if (tmpcd[i] == 'o') S = 91;

else if (tmpcd[i] == 'p') S = 112;

else if (tmpcd[i] == 'r') S = 128;

else if (tmpcd[i] == 'g') S = 133;

else if (tmpcd[i] == 's') S = 136;

else if (tmpcd[i] == 't') S = 155;

else if (tmpcd[i] == 'u') S = 177;

else if (tmpcd[i] == 'v') S = 181;

else if (tmpcd[i] == 'w') S = 194;

else if (tmpcd[i] == 'm') S = 198;

else if (tmpcd[i] == 'x') S = 204;

else if (tmpcd[i] == ':' || tmpcd[i] == '.' || tmpcd[i] == ',' || tmpcd[i] == '{' || tmpcd[i] == '}' || tmpcd[i] == '[' || tmpcd[i] == ']' || tmpcd[i] == '(' || tmpcd[i] == ')' || tmpcd[i] == ';' || tmpcd[i] == '#') S = 206;

else if (tmpcd[i] == '<' || tmpcd[i] == '>') S = 207;

else if (tmpcd[i] == '|' || tmpcd[i] == '&') S = 209;

else if (tmpcd[i] == '=' || tmpcd[i] == '!') S = 211;

else if (tmpcd[i] == '-') S = 213;

else if (tmpcd[i] == '?') S = 356;

else if (tmpcd[i] == '+' || tmpcd[i] == '%' || tmpcd[i] == '/' || tmpcd[i] == '\*') S = 215;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 1:

if (isnum(tmpcd[i]) || tmpcd[i] == '0') S = 1;

else if (tmpcd[i] == '.') S = 3;

else if (tmpcd[i] == ',') S = 358;

else if (stop\_sing(tmpcd[i])) S = 301;

else S = 401;

break;

case 2:

if (tmpcd[i] == '.') S = 3;

else if (stop\_sing(tmpcd[i])) S = 301;

else S = 401;

break;

case 3:

if (isnum(tmpcd[i]) || tmpcd[i] == '0') S = 4;

else S = 401;

break;

case 4:

if (isnum(tmpcd[i]) || tmpcd[i] == '0') S = 4;

else if (tmpcd[i] == 'E' || tmpcd[i] == 'e') S = 5;

else if (stop\_sing(tmpcd[i])) S = 301;

else S = 401;

break;

case 5:

if (tmpcd[i] == '+' || tmpcd[i] == '-') S = 6;

else S = 401;

break;

case 6:

if (isnum(tmpcd[i]) || tmpcd[i] == '0') S = 212;

else S = 401;

break;

case 7:

if (tmpcd[i] == 's') S = 8;

if (tmpcd[i] == 'n') S = 9;

if (tmpcd[i] == 'u') S = 10;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 8:

if (tmpcd[i] == 'm') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 9:

if (tmpcd[i] == 'd') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 10:

if (tmpcd[i] == 't') S = 11;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 11:

if (tmpcd[i] == 'o') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 12:

if (tmpcd[i] == 'i') S = 13;

else if (tmpcd[i] == 'o') S = 18;

else if (tmpcd[i] == 'r') S = 20;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 13:

if (tmpcd[i] == 't') S = 14;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 14:

if (tmpcd[i] == 'a') S = 15;

else if (tmpcd[i] == 'o') S = 17;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 15:

if (tmpcd[i] == 'n') S = 16;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 16:

if (tmpcd[i] == 'd') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 17:

if (tmpcd[i] == 'r') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 18:

if (tmpcd[i] == 'o') S = 19;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 19:

if (tmpcd[i] == 'l') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 20:

if (tmpcd[i] == 'e') S = 21;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 21:

if (tmpcd[i] == 'a') S = 22;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 22:

if (tmpcd[i] == 'k') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 23:

if (tmpcd[i] == 'a') S = 24;

else if (tmpcd[i] == 'h') S = 28;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 24:

if (tmpcd[i] == 's') S = 25;

else if (tmpcd[i] == 't') S = 26;

else if (tmpcd[i] == 'l') S = 30;

else if (tmpcd[i] == 'o') S = 33;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 25:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 26:

if (tmpcd[i] == 'c') S = 27;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 27:

if (tmpcd[i] == 'h') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 28:

if (tmpcd[i] == 'a') S = 29;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 29:

if (tmpcd[i] == 'r') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 30:

if (tmpcd[i] == 'a') S = 31;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 31:

if (tmpcd[i] == 's') S = 32;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 32:

if (tmpcd[i] == 's') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 33:

if (tmpcd[i] == 'n') S = 34;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 34:

if (tmpcd[i] == 's') S = 35;

else if (tmpcd[i] == 't') S = 36;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 35:

if (tmpcd[i] == 't') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 36:

if (tmpcd[i] == 'i') S = 38;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 38:

if (tmpcd[i] == 'n') S = 39;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 39:

if (tmpcd[i] == 'u') S = 40;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 40:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 41:

if (tmpcd[i] == 'e') S = 42;

else if (tmpcd[i] == 'o') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 42:

if (tmpcd[i] == 'f') S = 43;

else if (tmpcd[i] == 'l') S = 47;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 43:

if (tmpcd[i] == 'a') S = 44;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 44:

if (tmpcd[i] == 'u') S = 45;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 45:

if (tmpcd[i] == 'l') S = 46;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 46:

if (tmpcd[i] == 't') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 47:

if (tmpcd[i] == 'e') S = 48;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 48:

if (tmpcd[i] == 't') S = 49;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 49:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 50:

if (tmpcd[i] == 'l') S = 51;

else if (tmpcd[i] == 'n') S = 53;

else if (tmpcd[i] == 'x') S = 55;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 51:

if (tmpcd[i] == 's') S = 52;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 52:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 53:

if (tmpcd[i] == 'u') S = 54;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 54:

if (tmpcd[i] == 'm') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 55:

if (tmpcd[i] == 'p') S = 56;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 56:

if (tmpcd[i] == 'o') S = 57;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 57:

if (tmpcd[i] == 'r') S = 58;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 58:

if (tmpcd[i] == 't') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 59:

if (tmpcd[i] == 'a') S = 60;

else if (tmpcd[i] == 'l') S = 63;

else if (tmpcd[i] == 'o') S = 66;

else if (tmpcd[i] == 'r') S = 102;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 60:

if (tmpcd[i] == 'l') S = 61;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 61:

if (tmpcd[i] == 's') S = 62;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 62:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 63:

if (tmpcd[i] == 'o') S = 64;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 64:

if (tmpcd[i] == 'a') S = 65;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 65:

if (tmpcd[i] == 't') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 66:

if (tmpcd[i] == 'r') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 67:

if (tmpcd[i] == 'f') S = 300;

else if (tmpcd[i] == 'n') S = 68;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 68:

if (tmpcd[i] == 't') S = 300;

else if (tmpcd[i] == 'l') S = 99;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 69:

if (tmpcd[i] == 'o') S = 70;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 70:

if (tmpcd[i] == 'o') S = 71;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 71:

if (tmpcd[i] == 'n') S = 72;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 72:

if (tmpcd[i] == 'g') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 73:

if (tmpcd[i] == 'a') S = 74;

else if (tmpcd[i] == 'e') S = 81;

else if (tmpcd[i] == 'o') S = 82;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 74:

if (tmpcd[i] == 'm') S = 75;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 75:

if (tmpcd[i] == 'e') S = 76;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 76:

if (tmpcd[i] == 's') S = 77;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 77:

if (tmpcd[i] == 'p') S = 78;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 78:

if (tmpcd[i] == 'a') S = 79;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 79:

if (tmpcd[i] == 'c') S = 80;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 80:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 81:

if (tmpcd[i] == 'w') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 82:

if (allchar(tmpcd[i + 1]) && tmpcd[i] == 't') S = 83;

else if (!allchar(tmpcd[i + 1]) && tmpcd[i] == 't') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 83:

if (tmpcd[i] == 'e') S = 84;

else if (tmpcd[i] == '\_') S = 89;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

case 84:

if (tmpcd[i] == 'x') S = 85;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 85:

if (tmpcd[i] == 'p') S = 86;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 86:

if (tmpcd[i] == 'e') S = 87;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 87:

if (tmpcd[i] == 'c') S = 88;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 88:

if (tmpcd[i] == 't') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 89:

if (tmpcd[i] == 'e') S = 90;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 90:

if (tmpcd[i] == 'q') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 91:

if (tmpcd[i] == 'p') S = 92;

else if (tmpcd[i] == 'r') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 92:

if (tmpcd[i] == 'e') S = 93;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 93:

if (tmpcd[i] == 'r') S = 94;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 94:

if (tmpcd[i] == 'a') S = 95;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 95:

if (tmpcd[i] == 't') S = 96;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 96:

if (tmpcd[i] == 'o') S = 97;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 98:

if (tmpcd[i] == 'r') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 99:

if (tmpcd[i] == 'i') S = 100;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 100:

if (tmpcd[i] == 'n') S = 101;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 101:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 102:

if (tmpcd[i] == 'i') S = 103;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 103:

if (tmpcd[i] == 'e') S = 104;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 104:

if (tmpcd[i] == 'n') S = 105;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 105:

if (tmpcd[i] == 'd') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 106:

if (tmpcd[i] == 'u') S = 107;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 107:

if (tmpcd[i] == 'l') S = 108;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 108:

if (tmpcd[i] == 'l') S = 109;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 109:

if (tmpcd[i] == 'p') S = 110;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 110:

if (tmpcd[i] == 't') S = 111;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 111:

if (tmpcd[i] == 'r') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 112:

if (tmpcd[i] == 'r') S = 113;

else if (tmpcd[i] == 'u') S = 124;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 113:

if (tmpcd[i] == 'i') S = 114;

else if (tmpcd[i] == 'o') S = 118;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 114:

if (tmpcd[i] == 'v') S = 115;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 115:

if (tmpcd[i] == 'a') S = 116;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 116:

if (tmpcd[i] == 't') S = 117;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 117:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 118:

if (tmpcd[i] == 't') S = 119;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 119:

if (tmpcd[i] == 'e') S = 120;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 120:

if (tmpcd[i] == 'c') S = 121;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 121:

if (tmpcd[i] == 't') S = 122;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 122:

if (tmpcd[i] == 'e') S = 123;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 123:

if (tmpcd[i] == 'd') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 124:

if (tmpcd[i] == 'b') S = 125;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 125:

if (tmpcd[i] == 'l') S = 126;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 126:

if (tmpcd[i] == 'i') S = 127;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 127:

if (tmpcd[i] == 'c') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 128:

if (tmpcd[i] == 'e') S = 129;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 129:

if (tmpcd[i] == 't') S = 130;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 130:

if (tmpcd[i] == 'u') S = 131;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 131:

if (tmpcd[i] == 'r') S = 132;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 132:

if (tmpcd[i] == 'n') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 133:

if (tmpcd[i] == 'o') S = 134;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 134:

if (tmpcd[i] == 't') S = 135;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 135:

if (tmpcd[i] == 'o') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 136:

if (tmpcd[i] == 'h') S = 137;

else if (tmpcd[i] == 'i') S = 140;

else if (tmpcd[i] == 't') S = 140;

else if (tmpcd[i] == 'w') S = 151;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 137:

if (tmpcd[i] == 'o') S = 138;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 138:

if (tmpcd[i] == 'r') S = 139;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 139:

if (tmpcd[i] == 't') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 140:

if (tmpcd[i] == 'z') S = 141;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 141:

if (tmpcd[i] == 'e') S = 142;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 142:

if (tmpcd[i] == 'o') S = 143;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 143:

if (tmpcd[i] == 'f') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 144:

if (tmpcd[i] == 'a') S = 145;

else if (tmpcd[i] == 'r') S = 148;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 145:

if (tmpcd[i] == 't') S = 146;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 146:

if (tmpcd[i] == 'i') S = 147;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 147:

if (tmpcd[i] == 'c') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 148:

if (tmpcd[i] == 'u') S = 149;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 149:

if (tmpcd[i] == 'c') S = 150;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 150:

if (tmpcd[i] == 't') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 151:

if (tmpcd[i] == 'i') S = 152;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 152:

if (tmpcd[i] == 't') S = 153;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 153:

if (tmpcd[i] == 'c') S = 154;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 154:

if (tmpcd[i] == 'h') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 155:

if (tmpcd[i] == 'e') S = 156;

else if (tmpcd[i] == 'h') S = 162;

else if (tmpcd[i] == 'r') S = 166;

else if (tmpcd[i] == 'y') S = 168;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 156:

if (tmpcd[i] == 'm') S = 157;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 157:

if (tmpcd[i] == 'p') S = 158;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 158:

if (tmpcd[i] == 'l') S = 159;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 159:

if (tmpcd[i] == 'a') S = 160;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 160:

if (tmpcd[i] == 't') S = 161;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 161:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 162:

if (tmpcd[i] == 'i') S = 163;

else if (tmpcd[i] == 'r') S = 164;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 163:

if (tmpcd[i] == 's') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 164:

if (tmpcd[i] == 'o') S = 165;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 165:

if (tmpcd[i] == 'w') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 166:

if (tmpcd[i] == 'u') S = 167;

else if (tmpcd[i] == 'y') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 167:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 168:

if (tmpcd[i] == 'p') S = 169;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 169:

if (tmpcd[i] == 'e') S = 170;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 170:

if (tmpcd[i] == 'n') S = 171;

else if (tmpcd[i] == 'i') S = 174;

else if (tmpcd[i] == 'd') S = 175;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 171:

if (tmpcd[i] == 'a') S = 172;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 172:

if (tmpcd[i] == 'm') S = 173;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 173:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 174:

if (tmpcd[i] == 'd') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 175:

if (tmpcd[i] == 'e') S = 175;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 176:

if (tmpcd[i] == 'f') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 177:

if (tmpcd[i] == 's') S = 178;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 178:

if (tmpcd[i] == 'i') S = 179;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 179:

if (tmpcd[i] == 'n') S = 180;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 180:

if (tmpcd[i] == 'g') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 181:

if (tmpcd[i] == 'i') S = 182;

else if (tmpcd[i] == 'o') S = 187;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 182:

if (tmpcd[i] == 'r') S = 183;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 183:

if (tmpcd[i] == 't') S = 184;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 184:

if (tmpcd[i] == 'u') S = 185;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 185:

if (tmpcd[i] == 'a') S = 186;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 186:

if (tmpcd[i] == 'l') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 187:

if (tmpcd[i] == 'i') S = 188;

else if (tmpcd[i] == 'l') S = 189;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 188:

if (tmpcd[i] == 'd') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 189:

if (tmpcd[i] == 'a') S = 190;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 190:

if (tmpcd[i] == 't') S = 191;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 191:

if (tmpcd[i] == 'i') S = 192;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 192:

if (tmpcd[i] == 'l') S = 193;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 193:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 194:

if (tmpcd[i] == 'h') S = 195;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 195:

if (tmpcd[i] == 'i') S = 196;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 196:

if (tmpcd[i] == 'l') S = 197;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 197:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 198:

if (tmpcd[i] == 'u') S = 199;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 199:

if (tmpcd[i] == 't') S = 200;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 200:

if (tmpcd[i] == 'a') S = 201;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 201:

if (tmpcd[i] == 'b') S = 202;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 202:

if (tmpcd[i] == 'l') S = 203;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 203:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 204:

if (tmpcd[i] == 'o') S = 205;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 205:

if (tmpcd[i] == 'r') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 218:

if (tmpcd[i] == 'e') S = 219;

else if (tmpcd[i] == 'o') S = 224;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 219:

if (tmpcd[i] == 'l') S = 220;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 221:

if (tmpcd[i] == 'e') S = 222;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 222:

if (tmpcd[i] == 't') S = 223;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break; \

case 223:

if (tmpcd[i] == 'e') S = 300;

else if (stop\_sing(tmpcd[i])) S = 0;

else if (allchar(tmpcd[i])) S = 350;

else S = 400;

break;

case 224:

if (allchar(tmpcd[i])) S = 350;

else if (stop\_sing(tmpcd[i])) S = 300;

else S = 400;

break;

case 206:

if (tmpcd[i] == ':') S = 352;

else S = 353;

break;

case 207:

if (tmpcd[i] == '=') S = 354;

else if (tmpcd[i] == '<' || tmpcd[i] == '>') S = 354;

else if (skip\_sings(tmpcd[i]) || allchar(tmpcd[i])) S = 355;

else S = 406;

break;

case 209:

if (tmpcd[i] == '=') S = 356;

else if (tmpcd[i] == '|' || tmpcd[i] == '&') S = 354;

else if (skip\_sings(tmpcd[i]) || allchar(tmpcd[i])) S = 357;

else S = 404;

break;

case 211:

if (tmpcd[i] == '=') S = 354;

else if (skip\_sings(tmpcd[i]) || allchar(tmpcd[i])) S = 357;

else S = 404;

break;

case 214:

if (isnum(tmpcd[i])) S = 1;

else if (tmpcd[i] == '-' || tmpcd[i] == '=') S = 356;

else if (skip\_sings(tmpcd[i]) || allchar(tmpcd[i])) S = 357;

else S = 404;

break;

case 215:

if (tmpcd[i] == '+' || tmpcd[i] == '=') S = 356;

else if (skip\_sings(tmpcd[i]) || allchar(tmpcd[i])) S = 357;

else S = 404;

break;

case 212:

if (isnum(tmpcd[i]) || tmpcd[i] == '0') S = 212;

else if (stop\_sing(tmpcd[i])) S = 301;

else S = 401;

break;

case 350:

if (isnum(tmpcd[i]) || tmpcd[i] == '0') S = 350;

else if (allchar(tmpcd[i])) S = 350;

else if (stop\_sing(tmpcd[i])) S = 0;

else S = 400;

break;

case 351:

if (tmpcd[i] == '"' || tmpcd[i] == Convert::ToChar("'") ) { tmpwd = ""; S = 0; }

break;

case 352:

if (skip\_sings(tmpcd[i]) || allchar(tmpcd[i])) S = 353;

else S = 404;

break;

case 354:

if (skip\_sings(tmpcd[i]) || allchar(tmpcd[i])) S = 355;

else S = 406;

break;

case 356:

if (skip\_sings(tmpcd[i]) || allchar(tmpcd[i]) || spec\_chr(tmpcd[i])) S = 357;

else S = 404;

break;

case 358:

if (isnum(tmpcd[i]) || tmpcd[i] == '0') S = 401;

break;

case 399:

if (tmpcd[i] == Convert::ToChar("'")) { tmpwd = ""; S = 0; }

break;

case 400:

//error word

if (stop\_sing(tmpcd[i])) S = 402;

break;

case 401:

//eror cnst

if (skip\_sings(tmpcd[i]) || spec\_chr(tmpcd[i])) S = 403;

break;

case 404:

//eror sing

if (skip\_sings(tmpcd[i]) || allchar(tmpcd[i]) || spec\_chr(tmpcd[i]) ) S = 405;

break;

case 406:

//eror sing

if (skip\_sings(tmpcd[i]) || allchar(tmpcd[i]) || spec\_chr(tmpcd[i])) S = 407;

break;

}

if (S != 0 && S != 400 && S != 301 && S != 353 && S != 355 && S != 357) tmpwd += tmpcd[i];

if (S == 353 || S == 355 || S ==357 || S == 301 || S == 402 || S == 403 || S == 405) i--;

if (S == 402) {

add\_eror(line, 0);

S = 0;

tmpwd = "";

}

else if (S == 403) {

add\_eror(line, 1);

S = 0;

tmpwd = "";

}

else if (S == 405) {

add\_eror(line, 2);

S = 0;

tmpwd = "";

}

else if (S == 407) {

add\_eror(line, 3);

S = 0;

tmpwd = "";

}

else if (S == 0 && tmpwd != "") {

S = 0;

add\_idet(tmpwd, idet\_rows);

tmpwd = "";

i--;

}

else if (S == 300) {

S = 0;

add\_lex(tmpwd, lex\_rows);

tmpwd = "";

}

else if (S == 301) {

S = 0;

add\_cnst(tmpwd, cnst\_rows);

tmpwd = "";

}

else if (S == 357) {

S = 0;

add\_op\_s(tmpwd, op\_rows);

tmpwd = "";

}

else if (S == 355) {

S = 0;

add\_comp(tmpwd, comp\_rows);

tmpwd = "";

}

else if (S == 353) {

S = 0;

add\_separ(tmpwd, sep\_rows);

tmpwd = "";

}

if (tmpcd[i] == '\n') line++;

}

}

void descriptor(int lex\_rows, int cnst\_rows, int idet\_rows, int op\_rows, int comp\_rows, int sep\_rows)

{

String^ tmpcd = richTextBox2->Text;

String^ tmpwd = "";

for (int i = 0; i < tmpcd->Length; i++) {

if (!skip\_sings(tmpcd[i]) && stop\_sing(tmpcd[i])) {

if (op\_sings(tmpcd[i]) || op\_comp(tmpcd[i])) {

while (i < tmpcd->Length && !skip\_sings(tmpcd[i]) && !allchar(tmpcd[i]) && tmpcd[i] != ';') tmpwd += tmpcd[i++];

i--;

}

else if (spec\_chr(tmpcd[i])) {

tmpwd += tmpcd[i++];

if (i < tmpcd->Length && tmpcd[i] == ':') {

tmpwd += tmpcd[i];

}

else i--;

}

}

else if (tmpcd[i] == '"') {

i++;

while (i < tmpcd->Length && tmpcd[i] != '"') tmpwd += tmpcd[i++];

tmpwd = "";

}

else if (isnum(tmpcd[i]) || tmpcd[i] == '0') {

while (i < tmpcd->Length && (tmpcd[i] == '.' || tmpcd[i] == '-' || tmpcd[i] == '+' || !stop\_sing(tmpcd[i]))) tmpwd += tmpcd[i++];

i--;

}

else if (allchar(tmpcd[i])) {

while(i < tmpcd->Length && !stop\_sing(tmpcd[i])) tmpwd += tmpcd[i++];

i--;

}

if (tmpwd != "") {

for (int j = 0; j < idet\_rows - 1; j++) {

if (Convert::ToString(t\_id->Rows[j]->Cells[1]->Value) == tmpwd) {

tmpwd = "(" + Cidet1->HeaderCell->Value + "," + (j + 1) + ")";

desTextBox4->Text += tmpwd + " ";

tmpwd = "";

break;

}

}

for (int j = 0; j < lex\_rows - 1; j++) {

if (Convert::ToString(t\_lexme->Rows[j]->Cells[1]->Value) == tmpwd) {

tmpwd = "(" + Clex1->HeaderCell->Value + "," + (j + 1) + ")";

desTextBox4->Text += tmpwd + " ";

tmpwd = "";

break;

}

}

for (int j = 0; j < cnst\_rows - 1; j++) {

if (Convert::ToString(t\_cnst->Rows[j]->Cells[1]->Value) == tmpwd) {

tmpwd = "(" + Ccnst1->HeaderCell->Value + "," + (j + 1) + ")";

desTextBox4->Text += tmpwd + " ";

tmpwd = "";

break;

}

}

for (int j = 0; j < op\_rows - 1; j++) {

if (Convert::ToString(t\_op\_sings->Rows[j]->Cells[1]->Value) == tmpwd) {

tmpwd = "(" + dataGridViewTextBoxColumn1->HeaderCell->Value + "," + (j + 1) + ")";

desTextBox4->Text += tmpwd + " ";

tmpwd = "";

break;

}

}

for (int j = 0; j < comp\_rows - 1; j++) {

if (Convert::ToString(t\_op\_comp->Rows[j]->Cells[1]->Value) == tmpwd) {

tmpwd = "(" + dataGridViewTextBoxColumn3->HeaderCell->Value + "," + (j + 1) + ")";

desTextBox4->Text += tmpwd + " ";

tmpwd = "";

break;

}

}

for (int j = 0; j < sep\_rows - 1; j++) {

if (Convert::ToString(t\_sep->Rows[j]->Cells[1]->Value) == tmpwd) {

tmpwd = "(" + dataGridViewTextBoxColumn5->HeaderCell->Value + "," + (j + 1) + ")";

desTextBox4->Text += tmpwd + " ";

tmpwd = "";

break;

}

}

tmpwd = "";

}

}

}

void psevda()

{

String^ tmpcd = desTextBox4->Text;

String^ tmpwd = "";

for (int i = 0; i < tmpcd->Length; i++) {

if (tmpcd[i] == '(') {

i++;

while (i < tmpcd->Length && tmpcd[i] != ',') tmpwd += tmpcd[i++];

i++;

if (tmpwd == Convert::ToString(Cidet1->HeaderCell->Value)) {

tmpwd = "";

while (tmpcd[i] != ')') tmpwd += tmpcd[i++];

i++;

tmpwd = Convert::ToString(t\_id->Rows[Convert::ToInt32(tmpwd) - 1]->Cells[2]->Value);

}

if (tmpwd == Convert::ToString(Clex1->HeaderCell->Value)) {

tmpwd = "";

while (tmpcd[i] != ')') tmpwd += tmpcd[i++];

i++;

tmpwd = Convert::ToString(t\_lexme->Rows[Convert::ToInt32(tmpwd) - 1]->Cells[2]->Value);

}

if (tmpwd == Convert::ToString(Ccnst1->HeaderCell->Value)) {

tmpwd = "";

while (tmpcd[i] != ')') tmpwd += tmpcd[i++];

i++;

tmpwd = Convert::ToString(t\_cnst->Rows[Convert::ToInt32(tmpwd) - 1]->Cells[2]->Value);

}

if (tmpwd == Convert::ToString(dataGridViewTextBoxColumn1->HeaderCell->Value)) {

tmpwd = "";

while (tmpcd[i] != ')') tmpwd += tmpcd[i++];

i++;

tmpwd = Convert::ToString(t\_op\_sings->Rows[Convert::ToInt32(tmpwd) - 1]->Cells[2]->Value);

}

if (tmpwd == Convert::ToString(dataGridViewTextBoxColumn3->HeaderCell->Value)) {

tmpwd = "";

while (tmpcd[i] != ')') tmpwd += tmpcd[i++];

i++;

tmpwd = Convert::ToString(t\_op\_comp->Rows[Convert::ToInt32(tmpwd) - 1]->Cells[2]->Value);

}

if (tmpwd == Convert::ToString(dataGridViewTextBoxColumn5->HeaderCell->Value)) {

tmpwd = "";

while (tmpcd[i] != ')') tmpwd += tmpcd[i++];

i++;

tmpwd = Convert::ToString(t\_sep->Rows[Convert::ToInt32(tmpwd) - 1]->Cells[2]->Value);

}

}

pseTextBox3->Text += tmpwd + " ";

tmpwd = "";

}

}

void cl()

{

richTextBox2->Clear();

pseTextBox3->Clear();

desTextBox4->Clear();

ErorBox->Clear();

t\_lexme->RowCount = 1;

t\_cnst->RowCount = 1;

t\_id->RowCount = 1;

t\_op\_sings->RowCount = 1;

t\_sep->RowCount = 1;

t\_op\_comp->RowCount = 1;

S = 0;

a = 0;

}

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e)

{

cl();

prework();

int lex\_row = 1, cnst\_row = 1, idet\_row = 1, op\_row = 1, comp\_row = 1, sep\_row = 1;

work(lex\_row, cnst\_row, idet\_row, op\_row, comp\_row, sep\_row);

descriptor(lex\_row, cnst\_row, idet\_row, op\_row, comp\_row, sep\_row);

psevda();

dop();

};

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e)

{

richTextBox1->Clear();

cl();

}

private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e) {

}

};

}

# Приложение 3. Руководство пользователя

1. Общие сведения о программе

Исполнительный файл: A\_1.exe. (рис. П3.1)

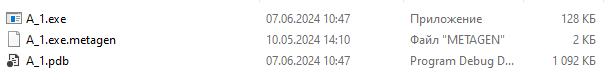


Рис. П3.1. Исполнительный файл внутри папки

2. Запуск программы

Двойным щелчком левой кнопки мыши по ярлыку программы «A\_1.exe» производится запуск программы.

3. Инструкция по работе

После запуска программы появляется окно интерфейса (рис.20), в котором можно ввести языковую конструкцию, либо выбрать текстовый файл, нажав на кнопку «Выбрать файл с кодом» (рис. П3.2), и текст загрузится в текстовое поле.

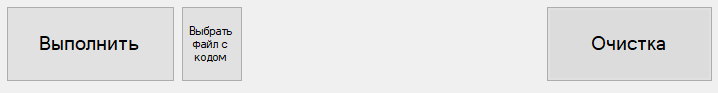


Рис. П3.2. Кнопки интерфейса

После ввода текста необходимо нажать кнопку «Выполнить». Программа начнёт предобработку текста: удалит пустые строки, табуляцию, комментарии, добавит нумерацию строк. (рис. П3.3)

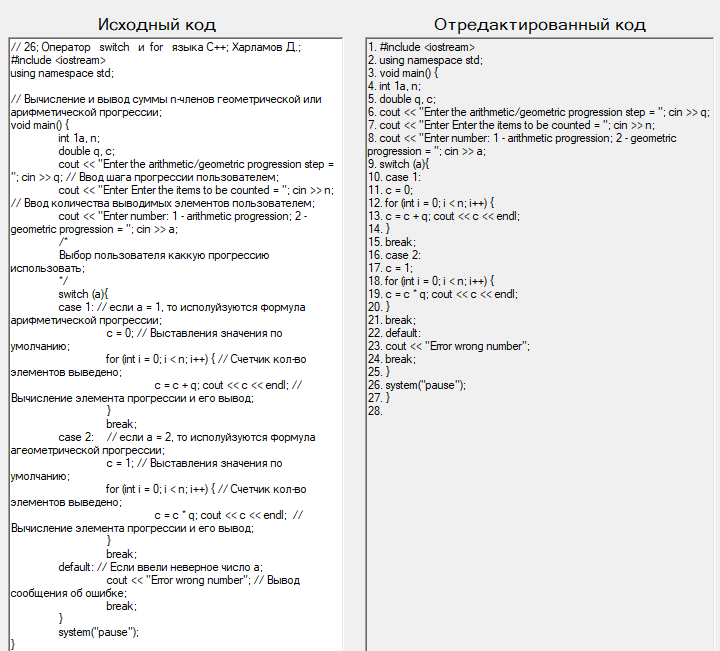


Рис. П3.3. Обработанный текст

Далее программа посимвольно обрабатывает лексемы, делит их на классы и выведет на экран таблицы классов лексем. (рис. П3.4)

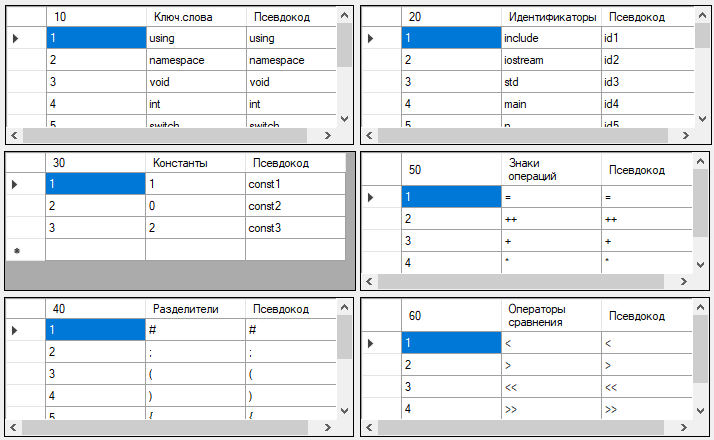


Рис. П3.4. Таблицы классов лексем

На основе таблиц на экран будет выведен псевдокод и дескрипторный код. (рис. П3.5)

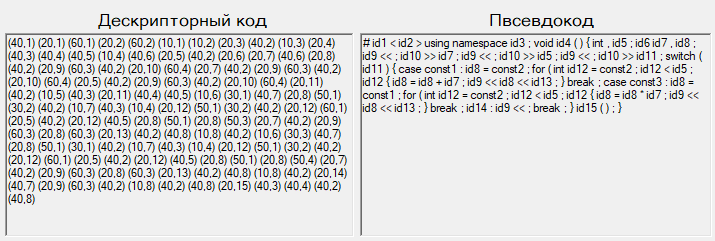


Рис. П3.5. Дескрипторный код и псевдокод

При возникновении ошибки в текстовое поле будет выведено сообщение с номером строки, где была допущена ошибка. (рис. П3.6)

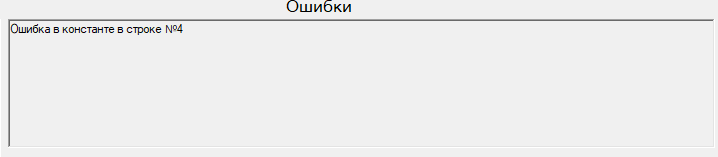


Рис. П3.6. Окно ошибок