минобрнауки россии

федеральное государственное бюджетное

образовательное учреждениевысшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

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

**КУРСОВАЯ РАБОТА**

|  |
| --- |
| по модулю Информатика |
| На тему Разработка алгоритмического обеспечения и построение лексического и синтаксического анализаторов |  |

|  |
| --- |
| Выполнил студент группы |
| Программная инженерия |
| *группа* |
| 1ПИб–01–21оп |
| *шифр, наименование* |
| Лебедев Вадим Максимович |
| *фамилия, имя, отчество* |

|  |
| --- |
| Руководитель |
| Ганичева Оксана Георгиевна |
| *фамилия, имя, отчество* |
| Доцент |
| *должность* |

|  |
| --- |
| Дата представления работы |
| «\_18 » июня\_\_\_\_\_\_2021 \_\_ г. |
|  |
| Заключение о допуске к защите |
|  |
|  |
|  |
|  |
| Оценка \_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| *количество баллов* |
| Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |

Череповец, 2021 год

Целью данной курсовой работы является разработка лексического и синтаксического анализаторов (регулярной и контекстно-свободной грамматик для них соответственно), а также их алгоритмов (блок-схем, графа конечного автомата, словесного описания) и программного обеспечения, моделирующего работу лексического анализатора. Разработка проводилась в соответствии с требованиями, изложенными в техническом задании (см. Приложение 1).

​Содержание

[Введение 4](#_Toc80884731)

[Описание и анализ предметной области 6](#_Toc80884732)

[Основная часть 9](#_Toc80884733)

[1.Постановка задачи 9](#_Toc80884734)

[2. Описание конечного автомата 10](#_Toc80884735)

[3. Построение грамматики по конечному автомату. 17](#_Toc80884736)

[4. Построение КС-грамматики, порождения и дерева вывода. 27](#_Toc80884737)

[Логическое проектирование 31](#_Toc80884738)

[1.Блок-схемы алгоритмов. 31](#_Toc80884739)

[2.Словесное описание алгоритмов 36](#_Toc80884740)

[3.Оценка сложности алгоритма 37](#_Toc80884741)

[Физическое проектирование 37](#_Toc80884742)

[1.Выбор структур данных 37](#_Toc80884743)

[2.Спецификация функций 38](#_Toc80884744)

[Проектирование интерфейса 38](#_Toc80884745)

[Тестирование программного обеспечения 39](#_Toc80884746)

[Заключение 44](#_Toc80884747)

[Список литературы 45](#_Toc80884748)

[Приложение 1 – Техническое задание 46](#_Toc80884749)

[Приложение 2 – Руководство пользователя 53](#_Toc80884750)

[Приложение 3 – Текст программы 54](#_Toc80884751)

# Введение

Целью курсовой работы является разработать грамматику для моделирования работы компилятора (этапов лексического и синтаксического анализов) согласно своему варианту. Для синтаксического анализа – КС-грамматику, для лексического анализа – регулярную грамматику, а также их алгоритмов (блок-схем, графа конечного автомата, словесного описания) и программного обеспечения, моделирующего работу лексического анализатора).

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

Лексический анализатор — это программа или часть программы, выполняющая лексический анализ. Лексический анализатор обычно работает в две стадии: сканирование и оценка.

Синтаксический анализ в информатике — процесс сопоставления линейной последовательности лексем (слов, токенов) естественного или формального языка с его формальной грамматикой. Результатом обычно является дерево разбора. Обычно применяется совместно с лексическим анализом. В ходе синтаксического анализа исходный текст преобразуется в структуру данных, обычно — в дерево, которое отражает синтаксическую структуру входной последовательности и хорошо подходит для дальнейшей обработки.

# Описание и анализ предметной области

В данной курсовой работе выполняется конструирование модели лексического и синтаксического анализаторов.

Лексический анализатор – это часть компилятора, которая читает исходную программу и выделяет в ее тексте лексемы входного языка. На вход лексического анализатора поступает текст исходной программы, а выходная информация передается для дальнейшей обработки компилятором на этапе синтаксического анализа и разбора.

С теоретической точки зрения лексический анализатор не является обязательной частью компилятора. Все его функции могут выполняться на этапе синтаксического разбора, поскольку полностью регламентированы синтаксисом входного языка. Однако существует несколько причин, по которым в состав практически всех компиляторов включают лексический анализ:

* применение лексического анализатора упрощает работу с текстом исходной программы на этапе синтаксического разбора
* сокращает объем обрабатываемой информации, так как лексический анализатор структурирует поступающий на вход исходный текст программы и выкидывает всю незначащую информацию;
* для выделения в тексте и разбора лексем возможно применять простую, эффективную и теоретически хорошо проработанную технику анализа, в то время как на этапе синтаксического анализа конструкций исходного языка используются достаточно сложные алгоритмы разбора;
* сканер отделяет сложный по конструкции синтаксический анализатор от работы непосредственно с текстом исходной программы, структура которого может варьироваться в зависимости от версии входного языка – при такой конструкции компилятора для перехода от одной версии языка к другой достаточно только перестроить относительно простой лексический анализатор.

Функции, выполняемые лексическим анализатором, и состав лексем, которые он выделяет в тексте исходной программы, могут меняться в зависимости от версии компилятора. То, какие функции должен выполнять лексический анализатор и какие типы лексем он должен выделять во входной программе, а какие оставлять для этапа синтаксического разбора, решают разработчики компилятора. В основном лексические анализаторы выполняют исключение из текста исходной программы комментариев, незначащих пробелов, символов табуляции и перевода строки, а также выделение лексем следующих типов: идентификаторов, строковых, символьных и числовых констант, ключевых слов входного языка, знаков операций и разделителей.

В нашем случае лексический анализатор выделяет лексемы из непрерывной последовательности символов, поэтому любой входной символ принадлежит к определенному классу лексем или разделителю (знак, разделяющий лексемы между собой). К классам лексем относятся:

- константы (целые или действительные числа),

- библиотеки (подключаемые к программе файлы или пространства имен),

- идентификаторы,

- ключевые слова,

- символы сравнения.

Результатом работы лексического анализатора является список всех лексем в виде таблицы, найденных в коде программы. То есть все лексемы, которые могут встретится в тексте программы входят в данную таблицу, при этом если лексема уже встречалась в коде ранее, она не записывается, поскольку уже имеется в таблице и имеет свой условный код. В таблице фактически записана вся исходная программа. Однако каждая таблица содержит в себе только один из классов лексем, например, таблица ключевых слов содержит только предварительно определенные зарезервированные идентификаторы, имеющие специальные значения и которые нельзя использовать в качестве идентификаторов в программе.

Конечный автомат для каждой входной цепочки языка дает ответ на вопрос о том, принадлежит или нет цепочка языку, заданному автоматом, однако в общем случае задача лексического анализатора несколько шире, чем просто проверка цепочки символов лексемы на соответствие входному языку. Кроме этого, он должен выполнить следующие действия:

- выполнить необходимые для сохранения информации об обнаруженной лексеме действия или выдать сообщение об ошибке,

- определить границы лексем, которые явно не указаны в тексе программы.

Моделирование лексического анализатора выполняется посредством построения конечного автомата для считывания и анализа входной последовательности символов кода программы и дальнейшего соотнесения их с определенными классами лексем. По сути конечный автомат представляет собой диаграмму переходов, построенной в соответствии с текстом программы. Сам текст программы должен удовлетворять конструкциям самого языка программирования. Для осуществления анализа был написан следующий код, представленный на рис.1.

# Основная часть

# 1.Постановка задачи

Код программы представляет из себя простой алгоритм работы операторов «for» и «if» со строками. С помощью оператора «for» мы перебираем строку посимвольно, после чего с помощью оператора «if» определяется, что будет выполнено далее. Внутри скобок, после самого «if» указывается условие, при верности которого будет выполнятся блок внутри фигурных скобок указанный далее.

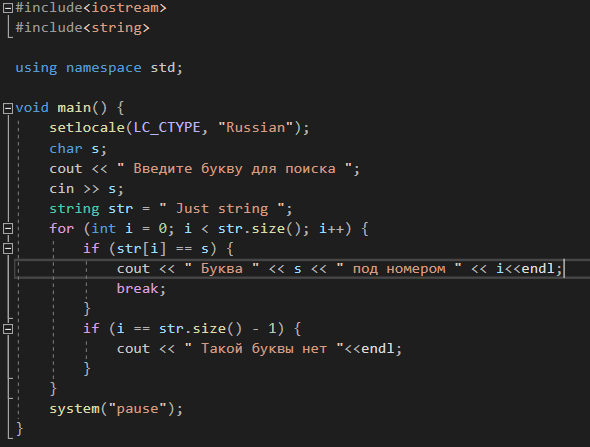


Рис. 1. Код программы

Текст программы, указанный на рис.1. был проверен в консольном приложении Microsoft Visual Studio на языке C++ на работоспособность и правильность написания, результаты тестирования кода приставлены ниже на рис.2.

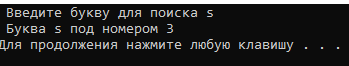


Рис.2. Результат работы программы

# 2. Описание конечного автомата

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

Лексема - это структурная единица языка, которая состоит из элементарных символов языка и не содержит в своем составе других структурных единиц языка. В обычном языке это не буквы, а слова, в языке программирования - идентификаторы, константы, ключевые слова языка, знаки операций и т.п. Состав возможных лексем каждого конкретного языка программирования определяется синтаксисом этого языка.

У лексического анализа имеются свои определённые функции:

- исключение из кода программы незначащих пробелов, знаков табуляции и символов перевода строки,

- исключение из исходного текста программы комментариев,

- выявление таких типов лексем как: ключевые слова, идентификаторы, константы и т.д.

При необходимости пользователь может определить свои функции, которые также могут определять собственный класс лексем.

С помощью лексического анализатора выполняется лексический анализ. Сам он является частью компилятора, который читает исходную программу и выделяет в ней лексемы выходного языка. Основными формализмами, лежащим в основе реализации лексических анализаторов, являются конечные автоматы и регулярные выражения. Лексический анализатор может работать в двух основных режимах: либо как подпрограмма, вызываемая синтаксическим анализатором для получения очередной лексемы, либо как полный проход, результатом которого является файл лексем.

Алгоритм работы простейшего анализатора выглядит следующим образом:

1. берется символ из входного потока, от которого зависит какой из лексических анализаторов должен запуститься, также символ может быть признан ошибочным или проигнорирован,
2. лексический анализатор выделяет символы, входящие в требуемую лексему до обнаружения ошибочного или ограничивающего лексему символа,
3. успешно распознанная лексема заносится в таблицу лексем и алгоритм возвращается к первому этапу иначе выдается сообщение об ошибке, и работа анализатора или прекращается, или идет возврат к первому этапу алгоритма.

Для построения работы лексического анализатора был смоделирован детерминированный конечный автомат, поскольку такая техника не требует сложной математической обработки и принципиально важных преобразований входных грамматик.

Конечный автомат (КА) — математическая абстракция в теории алгоритмов, модель дискретного устройства, в каждый момент времени находящегося в одном состоянии из множества возможных. Является частным случаем абстрактного дискретного автомата, число возможных внутренних состояний которого конечно.

При работе на вход КА поступают последовательно входные воздействия, а на выходе КА формирует выходные сигналы. Обычно под входными воздействиями принимают подачу на вход автомата символов одного алфавита, а на выход КА в процессе работы выдаёт символы в общем случае другого, возможно даже не пересекающегося со входным, алфавита.

Такой автомат может быть детерминированным или недетерминированным, то есть может иметь более одного перехода из состояния при одном и том же входном символе.

Конечный автомат обладает рядом свойств:

- конечный автомат может решать ряд легких задач компиляции. Лексический блок почти всегда строится на основе конечного автомата.

- работа конечного автомата отличается высоким быстродействием.

- моделирование конечного автомата требует фиксированного объема памяти, что упрощает управление памятью.

- существует ряд теорем и алгоритмов, позволяющих конструировать и упрощать конечные автоматы.

Конечные автоматы обычно используются для организации и представления потока выполнения чего-либо. В один так времени у конечного автомата активно только одно состояние.

Конечным автоматом называется совокупность пяти компонентов следующего вида: A = (X, S, F, s0, ∂), где:

* X – входной алфавит;
* S – конечное множество состояний;
* F – множество заключительных состояний;
* s0 – начальное состояние;
* ∂ - отображение вида (S, X) -> S;

Грамматика – математический аппарат для изучения синтаксиса языков. Фактически определение грамматики языка, указывает правила порождения цепочек символов, принадлежащих этому языку. Таким образом, еще одно определение грамматики – это генератор цепочек языка. Она относится ко второму способу определения языков - порождению цепочек символов.

Для многих языков (и для синтаксической части языков программирования в том числе) допустимо использовать формальное описание грамматики, построенное на основе системы правил. Для полного формального определения грамматики кроме правил порождения цепочек языка необходимо задать также алфавит языка.

Грамматику можно описать различными способами, но существует единый метод, называемый формальным определением грамматики. Также все грамматики делятся на типы, которые были выведены на основе структуры их правил профессором Массачусетского технологического института, лингвистом Ноамом Хомским. Данная классификация получила название Иерархия Хомского.

Если все без исключения продукции грамматики удовлетворяют некоторой заданной структуре, то данную структуру будут соотносить с определенным типом, описанным в иерархии. В любой грамматике достаточно иметь одно правило, которое не будет удовлетворять общему требованию структуры, чтобы грамматика не попала в заданный тип. Так, одним из типов по Иерархии Хомского является регулярные или автоматные грамматики, который и будет использоваться в ходе работы над данной курсовой работой.

Формально грамматика G определяется как четверка G = <T, N, P, S>, где:

T – конечное не пустое множество терминальных символов (символы, которые входят в алфавит языка, порождаемого грамматикой);

N – конечное не пустое множество не терминальных символов (представляет собой переменные, которые обозначают множество строк, которые помогают в определении языка порождающих грамматикой);

P – конечное множество правил грамматик(продукции). Продукции определяют способ, который терминалы и не терминалы могут объединяться для создания строк;

S – стартовый символ грамматики или аксиома грамматики (символ грамматики, который не отражается в общем множестве нетерминальных символов).

По классификации Хомского выделяют четыре типа грамматик.

Тип 0: грамматики с фразовой структурой

На структуру их правил не накладывается никаких ограничений: для грамматики вида G(T,N,P,S), V= NUT правила имеют вид: α→β, где α∈V+, β∈V\*. Это самый общий тип грамматик, в него попадают все без исключения формальные грамматики. 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

В этот тип входят два основных класса грамматик.

Контекстно-зависимые грамматики G(T,N,P,S), V= NUT правила имеют вид: α1A α2→ α1β α2, где α1 ,α2∈V\*,A∈ N, β∈ V+.

Не укорачивающие грамматики G (T, N, P, S), V= NUT правила имеют вид: α→β, где α,β∈V+, |β|≥|α|.

Не укорачивающие грамматики имеют такую структуру правил, что при построении предложений языка, заданного грамматикой, любая цепочка символов может быть заменена на цепочку символов не меньшей длины.

Структура правил КЗ – грамматик такова, что при построении предложений заданного ими языка один и тот же нетерминальный символ может быть заменен на ту или иную цепочку символов в зависимости от того контекста, в котором он встречается. Именно поэтому эти грамматики называются «контекстно-зависимыми».

Доказано, что эти два класса грамматик эквивалентны. Это значит, что для любого языка, заданного контекстно-зависимой грамматикой, можно построить не укорачивающую грамматику, которая будет задавать эквивалентный язык.

Тип 2: контекстно-свободные (КС) грамматики

КС грамматики G (T, N, P, S); V= NUT правила имеют вид: A→β, где A∈ N, β∈V+. Такие грамматики также иногда называют не укорачивающими контекстно-свободными (НКС) грамматиками (потому, что в правой части правил у них должен всегда стоять как минимум один символ).

Существует также почти эквивалентный им класс грамматик – укорачивающие контекстно-свободные (УКС) грамматики G (T, N, P, S), V=NUT правила имеют вид: A→β, где β∈V\*, A∈ VN.

Разница между этими двумя классами грамматик заключается лишь в том, что в УКС -грамматиках в правой части правил может присутствовать пустая цепочка (l), а в НКС-грамматиках – нет. Язык, заданный НКС-грамматикой, не может содержать пустой цепочки. Доказано, что эти два класса грамматик почти эквивалентны. В дальнейшем, говоря о КС грамматиках, не будет уточняться, какой класс грамматик (УКС или НКС) имеется в виду, если возможность наличия в языке пустой цепочке не имеет принципиального значения.

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

Тип 3: регулярные грамматики

К типу регулярных относятся два эквивалентных класса грамматик: леволинейные и праволинейные.

Леволинейные грамматики G(T,N,P,S), V= NUT правила имеют вид: A→Bγ или A→γ, где B,A∈ N, γ∈VT\*.

В свою очередь, праволинейные грамматики G(T,N,P,S), V= NUT правила имеют вид: A→γB или A→γ, где B,A∈ N, γ∈VT\*.

Эти два класса грамматик эквивалентны и относятся к типу регулярных грамматик.

Регулярные грамматики используются при описании простейших конструкций языков программирования: идентификаторов, констант, строк, комментариев и т.д. Эти грамматики исключительно просты и удобны в использовании, поэтому в компиляторах на их основе строятся функции лексического анализа входного языка.

Регулярная грамматика, строящаяся на основе конечного автомата, задается следующим образом:

G = (N, T, S, P)

1. S = N ⋃{ S1, …, S10, S14,…S64,S70};
2. Т = {X} = { a..z, A..Z, 0..9, +, -, \*, /, %, ,”, ‘, ;, (, ), {, }, [, ], <, >, =, &, |, ! , “ “,}
3. S = S0;
4. F = { S1, S2, S3, S4, S5, S6, S7, S8, S9, S 10};
5. Отображение продукций грамматики строится по определенным правилам:

- Каждому правилу конечного автомата вида (A, a) -> B должна соответствовать продукция грамматики вида A -> a, B;

- Каждому правилу конечного автомата вида (A, a) -> Z, где Z – является конечным состоянием, должна соответствовать продукция грамматики вида A -> a;

Схема конечного автомата представлена на рис.3.

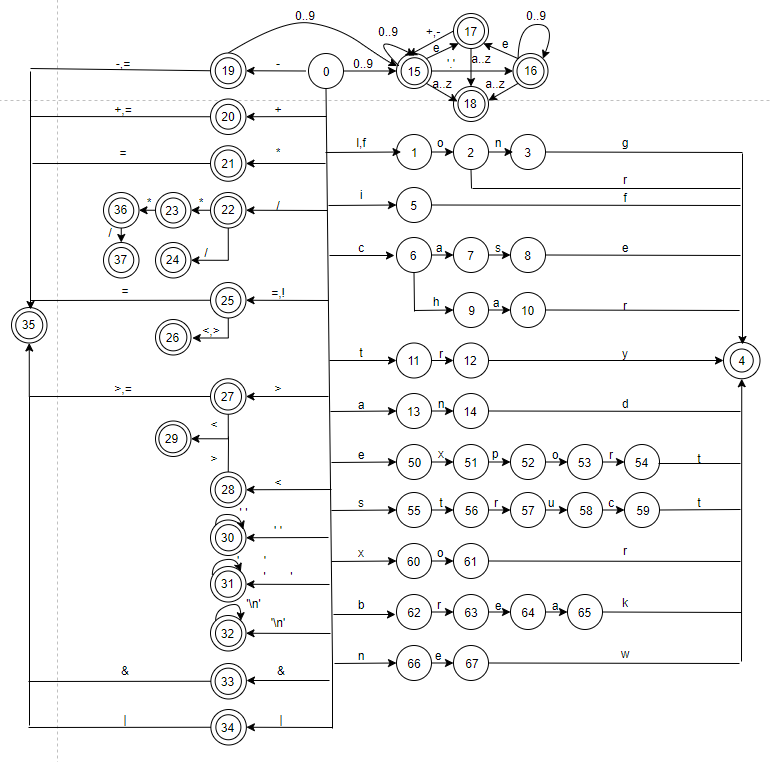


Рис.3.Схема конечного автомата.

# 3. Построение грамматики по конечному автомату.

Регулярная грамматика — формальная грамматика типа 3 по иерархии Хомского, регулярные грамматики определяют в точности все регулярные языки, и поэтому эквивалентны конечным автоматам и регулярным выражениям. Регулярные грамматики являются подмножеством контекстно-свободных.

К типу регулярных относятся два эквивалентных класса грамматик: леволинейные и праволинейные.

Леволинейные грамматики G(T,N,P,S), V=N∪T могут иметь правила двух видов: A→Bα или A→α, где A, B∈N , α ∈VT\*.

Праволинейные грамматики G(T,N,P,S), V=N∪T могут иметь правила тоже двух видов:A→αB или A→α, где A, B∈N, α ∈ VT\*.

Конечному автомату А= <V, S, δ, s0 , F> можно поставить в соответствие регулярную грамматику G = <N, T, P, S>, у которой:

1. T=V;
2. N=S;
3. S={s0};
4. Множество правил подстановки Р строится таким образом: каждой команде автомата (si, a) –> sk ставится в соответствие правило подстановки   si –> а sk, если sk∈S, либо si –> а , если sk∈Z, где Z – заключительное состояние.

Преобразование грамматики по исходному конечному автомату представлено в таблице 1.

Таблица 1

Формальное описание конечного автомата и регулярной грамматики по заданному конечному автомату

|  |  |
| --- | --- |
| Конечный автомат | Регулярная грамматика |
| А= <X, S, δ, s0 , F>, где  S = {S0, S1, …, S67};  X = {A..z, 0..9, (, ), {, }, [, ], <, >, |, &, -, =, +, /,  #, :, ; }  S={S0..S67} | G = <N, T, P, S>, где  T = {A..z, 0..9, (, ), {, }, [, ], <, >, |, &, -, =, +, /,  #, :, ; }  S=S0 |

Продолжение таблицы 1

|  |  |
| --- | --- |
| F = {S32, S18, S29, S4, S40, S15, S16, S17, S45, S19, S20, S21, S22, S35, S25, S27, S28, S33, S34};  где  S32 – перенос строки;  S18, S29 – ошибки;  S4 – ключевое слово;  S40 – идентификатор;  S15, S16, S17 – числа;  S45 – символ;  S19, S20, S21, S22, S35, S25, S27, S28, S33, S34 – операции;  δ:  (S0, l) → S1  (S0, f) → S1  (S0, не l) → S40  (S0, не f) → S40  (S1, o) → S2  (S1, не o) → S40  (S2, n) → S3  (S2, не n) → S40  (S2,r) →S4  (S2,не r) →S40  (S3, g) → S4  (S3, не g) → S40 | S0 → l,S1  S0→l  S0 → f,S1  S0→l  S0 → не l,S40  S0 → не f,S40  S1→o,S2  S1→o  S1→не o,S40  S2→n,S3  S2→n  S2→не n,S40  S2→r,S4  S2→r  S2→не r,S40  S3→g,S4  S3→g  S3→ не g,S40 |

Продолжение таблицы 1

|  |  |
| --- | --- |
| (S0,i) → S5  (S0, не i)→ S40  (S5,f) → S4  (S5, не f) → S40  (S0, c) → S6  (S0, не c) → S40  (S6, a) → S7  (S6, не a) → S40  (S7, s) → S8  (S7, не s) → S40  (S8, e) → S4  (S8, не e) → S40  (S0, c) → S6  (S0, не c) → S40  (S6, h) → S9  (S6, не h) → S40  (S9, a) → S10  (S7, не a) → S40  (S10, r) → S4  (S10, не r) → S40 | S0→i,S5  S0→i  S0→не i,S40  S5→f,S4  S5→f  S5→не f,S40  S0→c,S6  S0→c  S0→не c,S40  S6→a,S7  S6→a  S6→не a,S40  S7→s,S8  S7→s  S7→не s,S40  S8→e,S4  S8→e  S8→не e,S40  S0→c,S6  S0→c  S0→не c,S40  S6→h,S9  S6→h  S6→не h,S40  S9→a,S10  S9→a  S9→не a,S40  S10→r,S4  S10→r  S10→не r,S40 |

Продолжение таблицы 1

|  |  |
| --- | --- |
| (S0, t) → S11  (S0, не t) → S40  (S11, r) → S12  (S11, не r) → S40  (S12, y) → S4  (S12, не y) → S40  (S0, a) → S13  (S0, не a) → S40  (S13, n) → S14  (S13, не n) → S40  (S14, d) → S4  (S14, не d) → S40  (S0, e) → S50  (S0, не e) → S40  (S50, x) → S51  (S50, не x) → S40  (S51, p) → S52  (S51, не p) → S40  (S52, o) → S53  (S52, не o) → S40  (S53, r) → S54  (S53, не r) → S40  (S54, t) → S4  (S54, не t) → S40 | S0→t,S11  S0→t  S0→не t,S40  S11→r,S12  S11→r  S11→не r,S40  S12→y,S4  S12→y  S12→не y,S40  S0→a,S13  S0→a  S0→не a,S40  S13→n,S14  S13→n  S13→не n,S40  S14→d,S4  S14→d  S14→не d,S40  S0→e,S50  S0→e  S0→не e,S40  S50→x,S51  S50→x  S50→не x,S40  S51→p,S52  S51→p  S51→не p,S40  S52→o,S53  S52→o  S52→не o,S40  S53→r,S54  S53→r |

Продолжение таблицы 1

|  |  |
| --- | --- |
| (S0, s) → S55  (S0, не s) → S40  (S55, t) → S56  (S55, не t) → S40  (S56, r) → S57  (S56, не r) → S40  (S57, u) → S58  (S57, не u) → S40  (S58, c) → S59  (S58, не c) → S40  (S59, t) → S4  (S59, не t) → S40  (S0, x) → S60  (S0, не x) → S40  (S60, o) → S61  (S60, не o) → S40  (S61, r) → S4  (S61, не r) → S40 | S53→не r,S40  S54→t,S4  S54→t  S54→не t,S40  S0→s,S55  S0→s  S0→не s,S40  S55→t,S56  S55→t  S55→не t,S40  S56→r,S57  S56→r  S56→не r,S40  S57→u,S58  S57→u  S57→не u,S40  S58→c,S59  S58→c  S58→не c,S40  S59→t,S4  S59→t  S59→не t,S40  S0→x,S60  S0→x  S0→не x,S40  S60→o,S61  S60→o  S60→не o,S40  S61→r,S4  S0→r  S0→не r,S40 |

Продолжение таблицы 1

|  |  |
| --- | --- |
| (S0, b) → S62  (S0, не b) → S40  (S62, r) → S63  (S62, не r) → S40  (S63, e) → S64  (S63, не e) → S40  (S64, a) → S65  (S64, не a) → S40  (S65, k) → S4  (S65, не k) → S40  (S0, n) → S66  (S0, не n) → S40  (S66, e) → S67  (S66, не e) → S40  (S67, w) → S4  (S67, не w) → S40  (S0, 0..9) → S15  (S0, -) → S19  (S0, +) → S20  (S0, \*) → S21  (S0, /) → S22  (S0, =) → S25  (S0, !) → S25  (S0, >) → S27  (S0, <) → S28  (S0, ‘ ’) → S30  (S0, ‘ ’) → S31  (S0, \n) → S32  (S0, &) → S33  (S0, |) → S34  (S15, 0..9) → S15  (S15, .) → S16  (S15, e) → S17  (S15, A..z) → S18  (S16, 0..9) → S16  (S16, e) → S17 | S0→b,S62  S0→b  S0→не b,S40  S62→r,S63  S62→r  S62→не r,S40  S63→e,S64  S63→e  S63→не e,S40  S64→a,S65  S64→a  S64→не a,S40  S65→k,S4  S65→k  S65→не k,S40  S0→n,S66  S0→n  S0→не n,S40  S66→e,S67  S66→e  S66→не e,S40  S67→w,S4  S67→w  S67→не w,S40  S0→0..9,S15  S0→0..9  S0→-,S19  S0→-  S0→+,S20  S0→+  S0→\*,S21  S0→\*  S0→/,S22  S0→/  S0→=,S25  S0→=  S0→!,S25  S0→!  S0→>,S27  S0→>  S0→<,S28  S0→<  S0→’ ’,S30  S0→’ ’  S0→’ ’,S31  S0→’ ’  S0→\n,S32  S0→\n  S0→&,S33  S0→&  S0→|,S34  S0→|  S15→0..9,S15  S15→0..9  S15→.,S16  S15→.  S15→e,S17  S15→e  S15→A..z,S18  S15→A..z  S16→0..9,S16  S16→0..9  S16→e,S17  S16→e |

Продолжение таблицы 1

|  |  |
| --- | --- |
| (S16, A..z) → S18  (S17, A..z) → S18  (S17, +) → S15  (S17, -) → S15  (S19, 0..9) → S15  (S19, -) → S35  (S19, =) → S35  (S20, +) → S35  (S20, =) → S35  (S21, =) → S35  (S22, \*) → S23  (S22, /) → S24 | S16→A..z,S18  S16→A..z  S17→A..z,S18  S17→A..z  S17→+,S15  S17→+  S17→-,S15  S17→-  S19→0..9,S15  S19→0..9  S19→-,S35  S19→-  S19→=,S35  S19→=  S20→+,S35  S20→+  S20→=,S35  S20→=  S21→=,S35  S21→=  S22→\*,S23  S22→\*  S22→/,S24  S22→/ |

Продолжение таблицы 1

|  |  |
| --- | --- |
| (S23, \*) → S36  (S25, =) → S35  (S25, <) → S26  (S25, >) → S26  (S27, >) → S35  (S27, =) → S35  (S27, <) → S29  (S28, >) → S29  (S30, ‘ ’) → S30  (S31, ‘ ’) → S31 | S23→\*,S36  S23→\*  S25→=,S35  S25→=  S25→<,S26  S25→<  S25→>,S26  S25→>  S27→>,S35  S27→>  S27→=,S35  S27→=  S27→<,S29  S27→<  S28→>,S29  S28→>  S30→’ ’,S30  S30→’ ’  S31→’ ’,S31  S31→’ ’ |

Продолжение таблицы 1

|  |  |
| --- | --- |
| (S32, \n) → S32  (S33, &) → S35  (S34, |) → S35  (S36, /) → S37 | S32→\n,S32  S32→\n  S33→&,S35  S33→&  S34→|,S35  S34→|  S36→/,S37  S36→/ |

# 4. Построение КС-грамматики, порождения и дерева вывода.

В основе синтаксического анализатора лежит распознаватель, построенный на основе какого-нибудь класса КС-грамматик.

Контекстно-свободная грамматика (КС-грамматика) – это частный случай формальной грамматики (второй тип по иерархии Хомского). В данной грамматике левые части продукция – это нетерминалы. С помощью КС-грамматик описывается грамматическая структура многих языков программирования.

Язык определяется либо с помощью построения дерева вывода, либо с помощью порождения.

В **порождении** правилами грамматики являются правила, в которых нетерминал из левой части заменяется строкой из правой части продукции. Из стартового символа грамматики выводится строка, содержащая терминальные символы. Такая строка называется предложением грамматики.

Построение порождения выполняется следующим образом: на каждом шаге делается одно из действий:

- выбирается заменяемый нетерминал;

- выбирается продукция для этого терминала.

**Деревом вывода** является графическим представлением, из которого удалена информация о порядке замещения. Каждый внутренний узел дерева помечается некоторым нетерминалом А, а узлы слева направо – символами из правой части продукции для этого нетерминала. Листья дерева вывода помечены нетерминалами или терминалами и, будучи прочитаны слева направо, образуют сентенциальную форму.

Упорядоченное помеченное дерево D называется деревом вывода цепочки в КС-грамматике G = (N, T, P, S), если выполнены следующие условия:

1) корень дерева D помечен как S;

2) каждый лист помечен либо как a ∈ T, либо как e;

3) каждая внутренняя вершина помечена как нетерминал A ∈ N;

4) если X - это нетерминал, которым помечена внутренняя вершина и X1, ..., Xn – метки ее прямых потомков в указанном порядке, то X → X1...Xk - правило из множества P;

5) Цепочка, составленная из выписанных слева направо меток листьев, равна w.

Существуют две основные стратегии синтаксического анализа:

1) нисходящий анализ (называемый также анализом "сверху вниз", или анализом "разверткой");

2) восходящий анализ (называемый также анализом "снизу вверх", или анализом "сверткой").

В нисходящем анализе дерево вывода цепочки строится от корня к листьям, т.е. дерево вывода "реконструируется" в прямом порядке, и аксиома грамматики "развертывается" в цепочку. В восходящем анализе дерево вывода строится от листьев к корню, и анализируемая цепочка "свертывается" в аксиому.

Каждому выводу в КС-грамматике, который начинается с нетерминального символа, однозначно сопоставляется ориентированный граф, который, в свою очередь, является деревом и называется деревом вывода. Дерево вывода сопоставляется не грамматике, а конкретному выводу в данной грамматике.

**Построение КС-грамматики (для заданного варианта задания)**

**Код для КС-грамматики:**

#include<iostream>

#include<string>

using namespace std;

void main() {

setlocale(LC\_CTYPE, "Russian");

char s;

cout << " Введите букву для поиска ";

cin >> s;

string str = " Just string ";

for (int i = 0; i < str.size(); i++) {

if (str[i] == s) {

cout << " Буква " << s << " под номером " << i<<endl;

break;

}

if (i == str.size() - 1) {

cout << " Такой буквы нет "<<endl;

}

}

system("pause");

}

**Описание КС-грамматики:**

G = (T,N,P,S), где

T = {if, (, ), {, }, =, for, <<,  ; , ==,  <,>, +, -,\* , /, &&,||}

N = {A,B,C,D,E}

S = F

P:

F->for(int A;A;A) {F} | if(A) {F} | break;

A->BCB | BCBD| BC

B-> б|ц

C->+ | - | \* | / | ++ | = | <

D-> .size()

Для рассмотрения процесса определения языка грамматики также может быть использовано построения порождения, согласно правилам представленной грамматики. Основная идея состоит в том, что продукция рассматривается как переписывающее правило, в котором нетерминал из левой части замещается строкой из правой части продукции.

Так, было построено левостороннее порождение, на основе имеющейся и составленной ранее КС-грамматики для тестовых данных, приведенных ниже, которое представлено далее:

F=>

for(int A;A;A) {F} =>

for(int BCB;A;A) {F} =>

for(int бCB; A;A) {F} =>

for(int б=B; A;A) {F} =>

for(int б=ц; A;A) {F} =>

for(int б=ц; BCBD;A) {F} =>

for(int б=ц; бCBD;A) {F} =>

for(int б=ц; б<BD;A) {F} =>

for(int б=ц; б<бD;A) {F} =>

for(int б=ц; б<б.size();A) {F} =>

for(int б=ц; б<б.size();BC) {F} =>

for(int б=ц; б<б.size();бC) {F} =>

for(int б=ц; б<б.size();б++) {F} =>

for(int б=ц; б<б.size();б++) { if(A) {F} }=>

for(int б=ц; б<б.size();б++) { if(BCB) {F} }=>

for(int б=ц; б<б.size();б++) { if(бCB) {F} }=>

for(int б=ц; б<б.size();б++) { if(б==B) {F} }=>

for(int б=ц; б<б.size();б++) { if(б==б) {F} }=>

for(int б=ц; б<б.size();б++) { if(б==б) {break} }

Нисходящее дерево вывода по заданной грамматике представлено на рис.3.

Изображение выглядит как текст, клавиатура, снимок экрана

Автоматически созданное описание

Рис.4. Дерево порождения.

# Логическое проектирование

# 1.Блок-схемы алгоритмов.

Ниже представлена общая блок-схема алгоритма программы (рис.4).

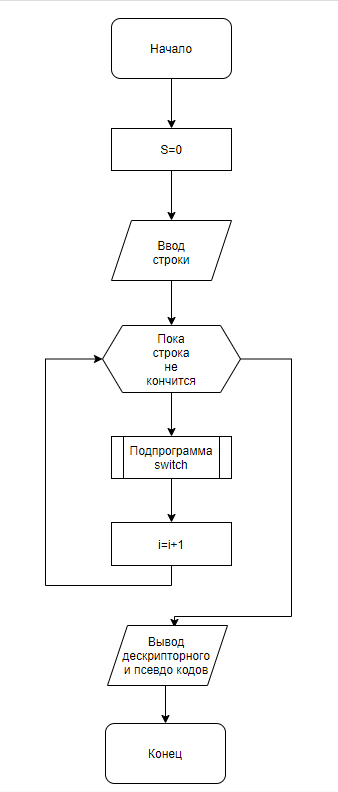


Рис. 4 Общая блок-схема алгоритма программы

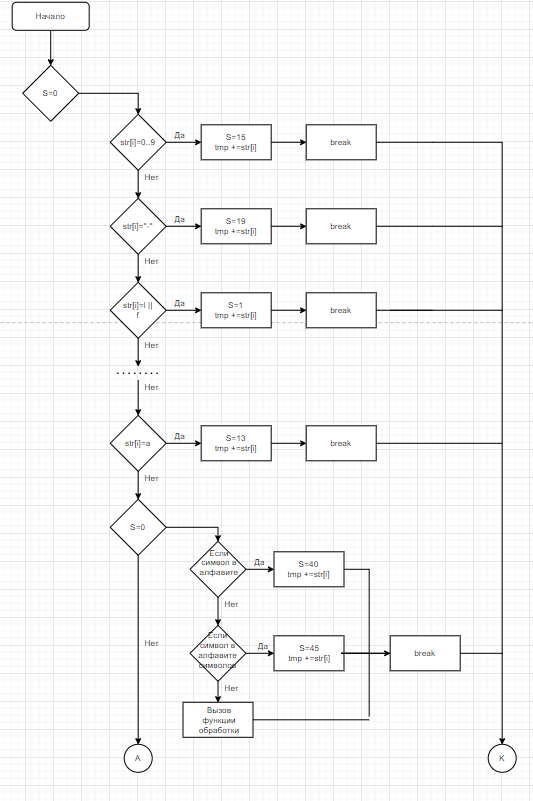
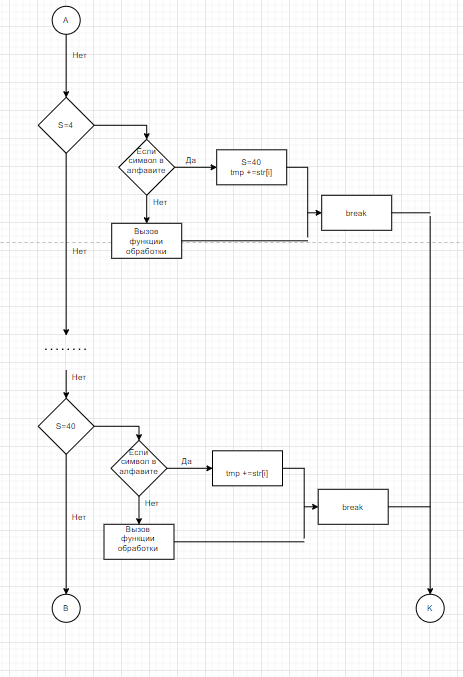


Рис. 5 Блок-схема Switch



Продолжение Рис. 5 Блок-схема Switch

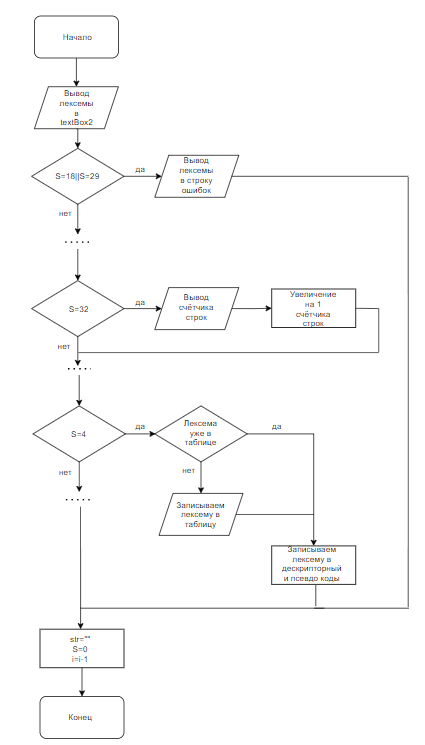


Рис. 6 Блок-схема подпрограммы output

На рис.5 и рис.6 представлены неполные блок-схемы подпрограмм switch и output(подпрограмма вывода лексем, распознанных с помощью подпрограммы switch). Пропущенные элементы подпрограмм работают по такому же принципу, что и представленные.

# 2.Словесное описание алгоритмов

2.2. Словесное описание алгоритма лексического анализатора (распознавание и занесение лексем в таблицу, удаление лишних пробелов и знаков табуляции, удаление комментариев, формирование дескрипторного и псевдокода, а также идентификация ошибок)

На вход поступает строка символов. Каждый символ рассматривается отдельно и каждый символ проходит switch (рис.5). По началу состояние нулевое. Если при рассмотрении следующего символа программа переходит в какое-то состояние то значит эти символы образуют лексему и так продолжается пока программа не встретится символ который не может быть продолжением лексемы. Если встречается символ, который не может продолжить лексему, то вызывается подпрограмма output куда передаётся эта лексема, состояние, управляющий элемент цикла (i) и счётчик строк. Исходя из состояния лексема записывается в соответствующую таблицу и записывается в строки дескрипторного и псевдо кодов. Ошибочные символы записываются в строку ошибок, остальные записываются во второе окно программы. Если переданное состояние соответствует лексеме переноса строки, то допишем к отображаемой строке счетчик (переданный ранее) и увеличим его на 1. После добавление лексемы в таблицу обнуляется состояние для следующего символа, а также управляющий элемент цикла (i) уменьшается на 1, так как вывод о том, закончилась ли лексема определяется на следующем, после ее конца символе, следовательно, нужно вернуться назад, чтобы не пропустить этот символ. После прохода основного цикла, на экран выводится дескрипторный и псевдо-коды.

# 3.Оценка сложности алгоритма

Каждый символ строки рассматривается 1 раз, следовательно, сложность алгоритма будет O(n).

В лучшем случае, если строка одна лексема, сложность будет O(n).

В худшем случае, если каждый символ это отдельная лексема, тогда сложность будет равна O(2n).

# Физическое проектирование

# 1.Выбор структур данных

В таблице 2 приведены структуры данных программы.

Таблица 2

Структуры данных

|  |  |  |
| --- | --- | --- |
| Наименование | Обозначение | Тип |
| Входная строка | str | String^ |
| Лексема | tmp | String^ |
| Строка символов | symbols | String^ |
| Строка цифр | numbers | String^ |
| Строка букв | alph | String^ |
| Строка букв и цифр | fullAlph | String^ |
| Счётчик строк | strCountr | int |
| Состояние | S | int |
| Индекс проверяемого символа | i | int |
| Флаг | flag | bool |
| Строка псевдокода | psevdocode | String^ |
| Строка дескрипторного кода | dscode | String^ |
| Счётчик чисел | numcount | int |
| Счётчик ключевых слов | keywordcount | int |
| Счётчик идентификаторов | idcount | int |
| Счётчик символов | symcount | int |
| Счётчик операторов | opcount | int |

# 2.Спецификация функций

Ниже приведены спецификации функций.

Таблица 3

Спецификации функций

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Обозначение | Тип | Параметры |
| Функция обработки лексемы | output | void | String^& str, int& S,int& I,int& strcount |
| Функция при нажатии кнопки Run | Button1\_Click | System::Void | System::Object^ sender, System::EventArgs^ e |
| Функция при нажатии кнопки Reset | Button2\_Click | System::Void | System::Object^ sender, System::EventArgs^ e |

# Проектирование интерфейса

Интерфейс программы (рис.7) представляет 5 текстовых полей, в которых отображается исходный код, обработанный код, дескрипторный код, псевдокод и строка ошибок, а также 5 таблиц, в которых отображаются ключевые слова, идентификаторы, числа, символ и операции.

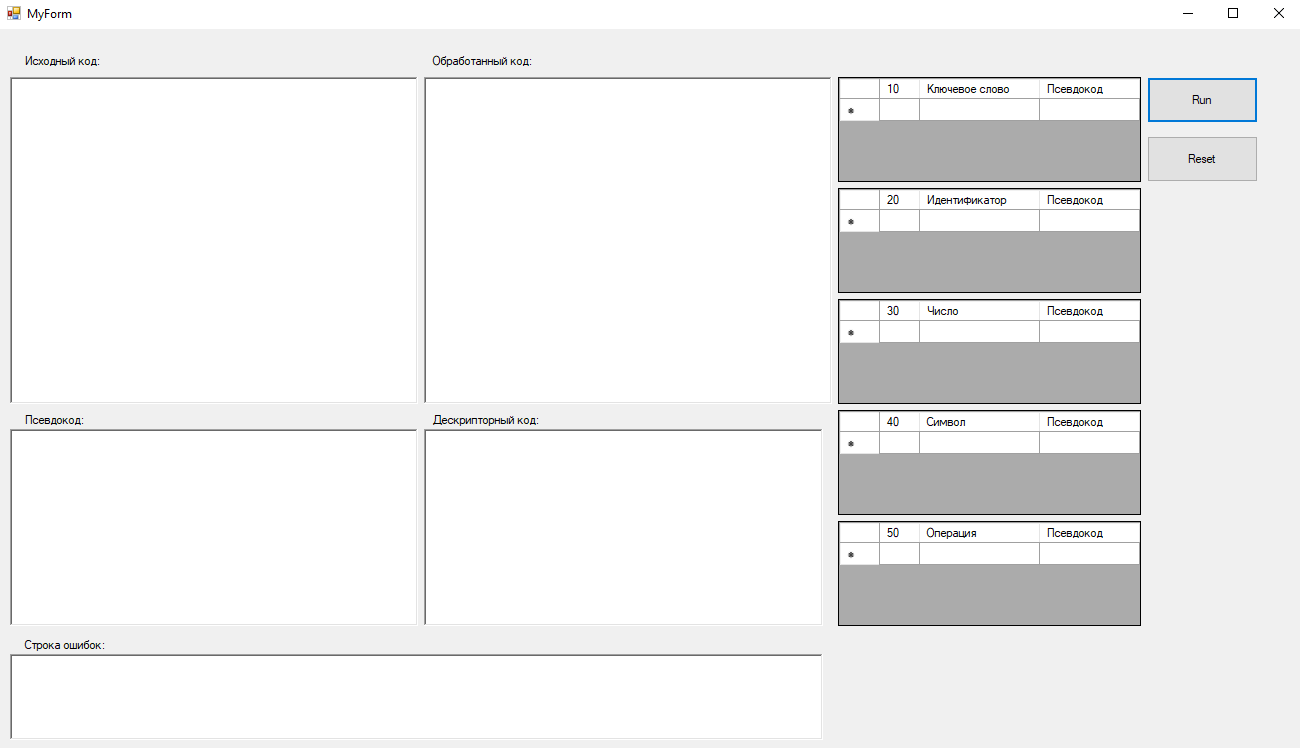


Рис.7 Интерфейс программы

# Тестирование программного обеспечения

Ниже приведены тесты и результаты.

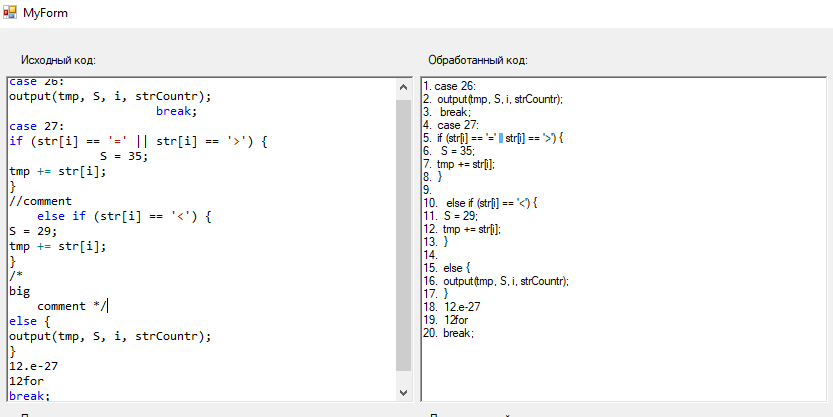


Рис.8 Удаление пробелов и комментариев

Изображение выглядит как текст, доска

Автоматически созданное описание

Рис.9 Псевдокод и дескрипторный код



Рис.10 Строка ошибок

Изображение выглядит как стол

Автоматически созданное описание

Рис.11 Ключевые слова

Изображение выглядит как стол

Автоматически созданное описание

Рис.12 Идентификаторы

Изображение выглядит как стол

Автоматически созданное описание

Рис.12 Идентификаторы(продолжение)

Изображение выглядит как стол

Автоматически созданное описание

Рис.12 Идентификаторы(продолжение)

Изображение выглядит как стол

Автоматически созданное описание

Рис.13 Числа

Изображение выглядит как стол

Автоматически созданное описание

Рис.13 Числа(продолжение)

Изображение выглядит как стол

Автоматически созданное описание

Рис.14 Символы

Изображение выглядит как стол

Автоматически созданное описание

Рис.14 Символы(продолжение)

Изображение выглядит как стол

Автоматически созданное описание

Рис.14 Символы(продолжение)

Изображение выглядит как стол

Автоматически созданное описание

Рис.14 Символы(продолжение)

Изображение выглядит как стол

Автоматически созданное описание

Рис.15 Операции

Изображение выглядит как стол

Автоматически созданное описание

Рис.15 Операции(продолжение)

В таблице 4 приведены все результаты тестирования данной программы.

Таблица 4

Результаты тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата тестирования | Тестируемый модуль | Тестировщик | Описание теста | Результат |
| 01.05.2021 | finalTry.exe | Лебедев В.М. | Проверка правильности определения ключевых слов | Успех |
| 02.05.2021 | finalTry.exe | Лебедев В.М. | Проверка правильности определения чисел | Успех |
| 03.05.2021 | finalTry.exe | Лебедев В.М. | Проверка правильности определения идентификаторов | Успех |
| 04.05.2021 | finalTry.exe | Лебедев В.М. | Проверка правильности определения символов | Успех |
| 05.05.2021 | finalTry.exe | Лебедев В.М. | Проверка правильности определения операций | Успех |
| 06.05.2021 | finalTry.exe | Лебедев В.М. | Проверка правильности определения ошибок | Успех |
| 07.05.2021 | finalTry.exe | Лебедев В.М. | Проверка правильности дескрипторного и псевдо кодов | Успех |
| 08.05.2021 | finalTry.exe | Лебедев В.М. | Проверка правильности удаления лишних пробелов | Успех |

# Заключение

В ходе данной курсовой работы была разработана регулярная грамматика для лексического анализатора и КС-грамматика для синтаксического анализатора, построено дерево вывода и разобрано порождение, а также написана программа, производящая лексический анализ кода. Помимо этого были разработаны различные блок – схемы, описывающие работу программы и подсчитана сложность алгоритма.

Благодаря этому были получены знания в области теории автоматов, на основе которых строятся лексические и синтаксические анализаторы.

# Список литературы

1. Теория языков программирования и методы трансляции: учебное пособие / О. Г. Ганичева ; М-во образования и науки Российской Федерации, Федеральное агентство по образованию, ГОУ ВПО "Череповецкий гос. ун-т", Ин-т информ. технологий. - Череповец : ГОУ ВПО "Череповецкий гос. ун-т", 2011. – 185 с.

2. Е.В. Ершов, д-р техн. наук, проф.; Л.Н. Виноградова и др. «Методика и организация самостоятельной работы студентов» Коллектив авторов, 2012 ФГБОУ ВПО «Череповецкий государственный университет», 2012. – 208 с.

3. Теория автоматов. Ю.Г. Карпов – СПб.: Питер, 2003. – 208 с.

4. Кудрявцев, В. Б.  Теория автоматов : учебник для бакалавриата и магистратуры / В. Б. Кудрявцев, С. В. Алешин, А. С. Подколзин. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2017. — 320 с.

5. А.А. Ожиганов Теория автоматов. Учебное пособие - Санкт-Петербург: НИУ ИТМО, 2013. - 84 с. - экз.

6. Электронный ресурс «Википедия» - https://www.wikipedia.org. Дата обращения: 17.06.2021.

# Приложение 1

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное

образовательное учреждение высшего профессионального образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий

наименование института (факультета)

Кафедра математического и программного обеспечения ЭВМ

наименование кафедры

Модуль «Информатика»

наименование дисциплины в соответствии с учебным планом

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ,

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

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

Разработка алгоритмического обеспечения и построение лексического и синтаксического анализаторов

Техническое задание на курсовую работу

Листов 7

Руководитель: Ганичева О. Г.

Исполнитель: студент группы 1ПИб-01-21оп Лебедев В.М.

Череповец

2021 год

Введение

Целью данной курсовой работы является разработка лексического и синтаксического анализаторов (регулярной и контекстно-свободной грамматик для них соответственно), а также их алгоритмов (блок-схем, графа конечного автомата, словесного описания) и программного обеспечения, моделирующего работу лексического анализатора.

Лексический анализ — процесс аналитического разбора входной последовательности символов на распознанные группы — лексемы — с целью получения на выходе идентифицированных последовательностей.

Синтаксический анализатор — часть программы, преобразующей входные данные в структурированный формат, нужный для задач последующего их анализа и использования.

Конечный автомат — математическая абстракция, модель дискретного устройства, имеющего один вход, один выход и в каждый момент времени находящегося в одном состоянии из множества возможных.

1. Основания для разработки

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

Дата утверждения:

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

2. Назначение разработки

Разработка лексического и синтаксического анализаторов выполняется с целью получения знаний и навыков построения КС-грамматики, для конструкций for и if языка C++, и конечного автомата для различных задач. Выполнение главной цели данной курсовой работы повышает как умение разработки программных продуктов, так и знания в области работы лексического, синтаксического анализаторов и алгоритмов работы конструкций языка C++.

3. Требования к программе

3.1. Требования к функциональным характеристикам

Модель лексического анализатора должна проводить обработку вводимого текста программы и удалять лишние пробелы, знаки табуляции, комментарии, излишние переходы на новую строку, выполнять подсчет количества строк в программе, диагностировать лексические ошибки, выводить таблицу классов лексем, дескрипторный код и псевдокод.

Работа лексического анализатора должна моделироваться конечным автоматом, граф переходов которого должен быть представлен в пояснительной записке.

3.2. Требования к надёжности

Для правильной работы программы необходимо:

1) вводимые лексемы должны удовлетворять возможным, которые определены с помощью состояний конечного автомата.

3.3. Условия эксплуатации

Условия эксплуатации ограниченны нормами работы в офисном помещении:

* оптимальное расположение клавиатуры – 0,1 – 0,3 м от края стола;
* расстояние между рабочими столами – не менее 2 м;
* между боковыми гранями мониторов – не менее 1,2 м;
* угол наклона спинки стула – около 30°;
* высота стола – не менее 0,6 м;
* время между перерывами – не более 2 часов;
* уровень влажности воздуха – 40-60%;
* освещённость рабочего места должна составлять 300-500 люкс;
* оптимальный уровень шума – 80 дБ.

3.4. Требования к составу и параметрам технических средств

* Оперативная память 512 Мб или больше;
* Свободное место на жестком диске от 120 Мб;
* Клавиатура
* Мышь
* Монитор
* Архитектура с разрядностью 32 или 64 бит

3.5. Требования к информационной и программной совместимости

Текст программы реализуется на языке высоко уровня – C++ с помощью компьютерного приложения Visual Studio 2019. Для выполнения действий, связанных с редактированием или дописыванием кода, в том случае, если на персональном электронном устройстве установлена более ранняя версия Visual Studio, необходимо установить дополнительно набор инструментов платформы v142 и пакет SDK для Windows версией 10.0. Для выполнения всех действий, связанных с работой программы, подходят такие операционные системы как Windows 10.

3.6. Требования к маркировке и упаковке

Требования не предъявляются.

3.7. Специальный требования

Требования не предъявляются.

4. Требования к программной документации

Программная документация должна содержать расчётно-пояснительную записку с содержанием:

Введение;

Описание предметной области;

Основная часть:

1) Постановка задачи. (Описание заданной конструкции на примере рабочего фрагмента программы);

2) Описание конечного автомата  (определение, математическое, табличное, графическое);

3) Построение грамматики по конечному автомату  (алгоритм, грамматика);

4) Построение  КС-грамматики, порождения и дерева вывода;

Логическое проектирование:

1) Блок-схемы алгоритмов;

2) Словесное описание алгоритмов;

3) Оценка сложности алгоритма;

Физическое проектирование:

1) Выбор структур данных;

2) Спецификация функций;

Проектирование интерфейса;

Тестирование программного обеспечения;

Заключение;

Список литературы;

Приложения:

2) Руководство пользователя;

3) Программный код.

5. Технико-экономические показатели

Не рассчитываются.

6. Стадии и этапы разработки

Ниже представлены стадии и этапы разработки (табл. П1.1).

Таблица П1.1

Стадии и этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапа разработки ПО | Сроки разработки | Результат выполнения | Отметка о выполнении |
| Анализ предметной области | До 25.03.2021 | Получение необходимых знаний для написания работы |  |
| Продумывание алгоритма | До 20.04.2021 | Составление алгоритма программы |  |
| Реализация алгоритма | До 20.05.2021 | Написание кода программы |  |
| Тестирование | До 10.06.2021 | Исправление багов программы |  |

7. Порядок контроля и приемки

На таблице ниже представлен порядок контроля и приемки (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 |
| Получение задания | До 25.03.2021 | Полученное задание курсовой работы |  |
| Написание технического задания | До 25.03.2021 | Готовое техническое задание |  |
| Анализ предметной области | До 25.03.2021 | Получение необходимой информации |  |
| Написание программы | До 10.06.2021 | Готовая протестированная программа |  |
| Написание Рассчётно – пояснительной записки | До 20.06.2021 | Готовая курсовая работа |  |
| Защита курсовой работы | 21.06.2021 | Оценка за курсовую работу |  |

# Приложение 2

Руководство пользователя

1. Общие сведения о программе

Программа выполняет лексический анализ программного кода, написанного на языке C++: удаляет лишние пробелы, знаки табуляции, переводы на новую строку и комментарии, а также заносит все лексемы в таблицу, диагностирует ошибки в коде и создаёт дескрипторный и псевдокод.

2. Описание установки

Данная программа не требует установки.

3. Описание запуска

Запустите файл «finalTry.exe», щёлкнув дважды по нему левой кнопкой мыши.

4. Инструкция по работе

Введите необходимый код в первое текстовое поле (слева сверху) и нажмите «Run». После этого программа приступит к работе. Результаты представлены на (рис.8 – рис.15).

# Приложение 3

Текст программы

#pragma once

namespace finalTry {

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();

//

//TODO: добавьте код конструктора

//

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

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

protected:

private: System::Windows::Forms::Label^ label1;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn7;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn8;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn9;

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

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn10;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn11;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn12;

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

private: System::Windows::Forms::RichTextBox^ richTextBox4;

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::RichTextBox^ richTextBox5;

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

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

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

this->label1 = (gcnew System::Windows::Forms::Label());

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

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

this->dataGridView1 = (gcnew System::Windows::Forms::DataGridView());

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

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

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

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

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

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

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

this->dataGridView3 = (gcnew System::Windows::Forms::DataGridView());

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

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

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

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

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

this->dataGridViewTextBoxColumn7 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dataGridViewTextBoxColumn8 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dataGridViewTextBoxColumn9 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

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

this->dataGridViewTextBoxColumn10 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dataGridViewTextBoxColumn11 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dataGridViewTextBoxColumn12 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->richTextBox3 = (gcnew System::Windows::Forms::RichTextBox());

this->richTextBox4 = (gcnew System::Windows::Forms::RichTextBox());

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->richTextBox5 = (gcnew System::Windows::Forms::RichTextBox());

this->label6 = (gcnew System::Windows::Forms::Label());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView1))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView2))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView3))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView4))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView5))->BeginInit();

this->SuspendLayout();

//

// button1

//

this->button1->Location = System::Drawing::Point(1149, 48);

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

this->button1->Size = System::Drawing::Size(111, 46);

this->button1->TabIndex = 0;

this->button1->Text = L"Run";

this->button1->UseVisualStyleBackColor = true;

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

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(87, 119);

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

this->label1->Size = System::Drawing::Size(0, 13);

this->label1->TabIndex = 3;

//

// richTextBox1

//

this->richTextBox1->Location = System::Drawing::Point(12, 48);

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

this->richTextBox1->Size = System::Drawing::Size(408, 327);

this->richTextBox1->TabIndex = 4;

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

//

// richTextBox2

//

this->richTextBox2->Location = System::Drawing::Point(426, 48);

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

this->richTextBox2->Size = System::Drawing::Size(408, 327);

this->richTextBox2->TabIndex = 5;

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

//

// dataGridView1

//

this->dataGridView1->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->dataGridView1->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->Column1,

this->KeyWord, this->Column2

});

this->dataGridView1->Location = System::Drawing::Point(840, 48);

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

this->dataGridView1->Size = System::Drawing::Size(303, 105);

this->dataGridView1->TabIndex = 6;

//

// Column1

//

this->Column1->HeaderText = L"10";

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

this->Column1->Width = 40;

//

// KeyWord

//

this->KeyWord->HeaderText = L"Ключевое слово";

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

this->KeyWord->Width = 120;

//

// Column2

//

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

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

//

// dataGridView2

//

this->dataGridView2->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->dataGridView2->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->dataGridViewTextBoxColumn1,

this->dataGridViewTextBoxColumn2, this->dataGridViewTextBoxColumn3

});

this->dataGridView2->Location = System::Drawing::Point(840, 159);

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

this->dataGridView2->Size = System::Drawing::Size(303, 105);

this->dataGridView2->TabIndex = 7;

//

// dataGridViewTextBoxColumn1

//

this->dataGridViewTextBoxColumn1->HeaderText = L"20";

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

this->dataGridViewTextBoxColumn1->Width = 40;

//

// dataGridViewTextBoxColumn2

//

this->dataGridViewTextBoxColumn2->HeaderText = L"Идентификатор";

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

this->dataGridViewTextBoxColumn2->Width = 120;

//

// dataGridViewTextBoxColumn3

//

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

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

//

// dataGridView3

//

this->dataGridView3->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->dataGridView3->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->dataGridViewTextBoxColumn4,

this->dataGridViewTextBoxColumn5, this->dataGridViewTextBoxColumn6

});

this->dataGridView3->Location = System::Drawing::Point(840, 270);

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

this->dataGridView3->Size = System::Drawing::Size(303, 105);

this->dataGridView3->TabIndex = 8;

//

// dataGridViewTextBoxColumn4

//

this->dataGridViewTextBoxColumn4->HeaderText = L"30";

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

this->dataGridViewTextBoxColumn4->Width = 40;

//

// dataGridViewTextBoxColumn5

//

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

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

this->dataGridViewTextBoxColumn5->Width = 120;

//

// dataGridViewTextBoxColumn6

//

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

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

//

// button2

//

this->button2->Location = System::Drawing::Point(1149, 107);

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

this->button2->Size = System::Drawing::Size(111, 46);

this->button2->TabIndex = 9;

this->button2->Text = L"Reset";

this->button2->UseVisualStyleBackColor = true;

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

//

// dataGridView4

//

this->dataGridView4->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->dataGridView4->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->dataGridViewTextBoxColumn7,

this->dataGridViewTextBoxColumn8, this->dataGridViewTextBoxColumn9

});

this->dataGridView4->Location = System::Drawing::Point(840, 381);

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

this->dataGridView4->Size = System::Drawing::Size(303, 105);

this->dataGridView4->TabIndex = 10;

//

// dataGridViewTextBoxColumn7

//

this->dataGridViewTextBoxColumn7->HeaderText = L"40";

this->dataGridViewTextBoxColumn7->Name = L"dataGridViewTextBoxColumn7";

this->dataGridViewTextBoxColumn7->Width = 40;

//

// dataGridViewTextBoxColumn8

//

this->dataGridViewTextBoxColumn8->HeaderText = L"Символ";

this->dataGridViewTextBoxColumn8->Name = L"dataGridViewTextBoxColumn8";

this->dataGridViewTextBoxColumn8->Width = 120;

//

// dataGridViewTextBoxColumn9

//

this->dataGridViewTextBoxColumn9->HeaderText = L"Псевдокод";

this->dataGridViewTextBoxColumn9->Name = L"dataGridViewTextBoxColumn9";

//

// dataGridView5

//

this->dataGridView5->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->dataGridView5->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->dataGridViewTextBoxColumn10,

this->dataGridViewTextBoxColumn11, this->dataGridViewTextBoxColumn12

});

this->dataGridView5->Location = System::Drawing::Point(840, 492);

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

this->dataGridView5->Size = System::Drawing::Size(303, 105);

this->dataGridView5->TabIndex = 11;

//

// dataGridViewTextBoxColumn10

//

this->dataGridViewTextBoxColumn10->HeaderText = L"50";

this->dataGridViewTextBoxColumn10->Name = L"dataGridViewTextBoxColumn10";

this->dataGridViewTextBoxColumn10->Width = 40;

//

// dataGridViewTextBoxColumn11

//

this->dataGridViewTextBoxColumn11->HeaderText = L"Операция";

this->dataGridViewTextBoxColumn11->Name = L"dataGridViewTextBoxColumn11";

this->dataGridViewTextBoxColumn11->Width = 120;

//

// dataGridViewTextBoxColumn12

//

this->dataGridViewTextBoxColumn12->HeaderText = L"Псевдокод";

this->dataGridViewTextBoxColumn12->Name = L"dataGridViewTextBoxColumn12";

//

// richTextBox3

//

this->richTextBox3->Location = System::Drawing::Point(12, 400);

this->richTextBox3->Name = L"richTextBox3";

this->richTextBox3->Size = System::Drawing::Size(408, 197);

this->richTextBox3->TabIndex = 12;

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

//

// richTextBox4

//

this->richTextBox4->Location = System::Drawing::Point(426, 400);

this->richTextBox4->Name = L"richTextBox4";

this->richTextBox4->Size = System::Drawing::Size(399, 197);

this->richTextBox4->TabIndex = 13;

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

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(24, 384);

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

this->label2->Size = System::Drawing::Size(66, 13);

this->label2->TabIndex = 14;

this->label2->Text = L"Псевдокод:";

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(432, 384);

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

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

this->label3->TabIndex = 15;

this->label3->Text = L"Дескрипторный код:";

//

// label4

//

this->label4->AutoSize = true;

this->label4->Location = System::Drawing::Point(24, 25);

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

this->label4->Size = System::Drawing::Size(82, 13);

this->label4->TabIndex = 16;

this->label4->Text = L"Исходный код:";

//

// label5

//

this->label5->AutoSize = true;

this->label5->Location = System::Drawing::Point(432, 25);

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

this->label5->Size = System::Drawing::Size(106, 13);

this->label5->TabIndex = 17;

this->label5->Text = L"Обработанный код:";

//

// richTextBox5

//

this->richTextBox5->Location = System::Drawing::Point(12, 625);

this->richTextBox5->Name = L"richTextBox5";

this->richTextBox5->Size = System::Drawing::Size(813, 86);

this->richTextBox5->TabIndex = 18;

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

//

// label6

//

this->label6->AutoSize = true;

this->label6->Location = System::Drawing::Point(24, 609);

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

this->label6->Size = System::Drawing::Size(87, 13);

this->label6->TabIndex = 19;

this->label6->Text = L"Строка ошибок:";

//

// MyForm

//

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

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

this->ClientSize = System::Drawing::Size(1304, 721);

this->Controls->Add(this->label6);

this->Controls->Add(this->richTextBox5);

this->Controls->Add(this->label5);

this->Controls->Add(this->label4);

this->Controls->Add(this->label3);

this->Controls->Add(this->label2);

this->Controls->Add(this->richTextBox4);

this->Controls->Add(this->richTextBox3);

this->Controls->Add(this->dataGridView5);

this->Controls->Add(this->dataGridView4);

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

this->Controls->Add(this->dataGridView3);

this->Controls->Add(this->dataGridView2);

this->Controls->Add(this->dataGridView1);

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

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

this->Controls->Add(this->label1);

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

this->Name = L"MyForm";

this->Text = L"MyForm";

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView1))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView2))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView3))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView4))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView5))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

int numcount = 1, keywordcount = 1, idcount = 1, opcount = 1, symcount = 1;

String^ psevdocode;

String^ dscode;

void output(String^& str, int& S, int& i, int& strcount)

{

//{

richTextBox2->Text += str;

if (S == 32) richTextBox2->Text += ++strcount + ". ";

//}

if (S == 18 || S == 29) { richTextBox5->Text += "Error : " + (strcount)+" : " + str + "\n"; }

if (S == 4) {

bool flag = false;

for (int i = 0; i < keywordcount - 1; i++) {

if (dataGridView1[1, i]->Value->ToString() == str) flag = true;

}

if (!flag) dataGridView1->Rows->Add(keywordcount++, str, str);

psevdocode += str + " ";

for (int i = 0; i < keywordcount - 1; i++) {

if (dataGridView1[1, i]->Value->ToString() == str) {

dscode += "(10."; dscode += dataGridView1[0, i]->Value->ToString() + ") ";

}

}

flag = false;

}

if (S == 40) {

bool flag = false;

for (int i = 0; i < idcount - 1; i++)

{

if (dataGridView2[1, i]->Value->ToString() == str) {

flag = true;

}

}

String^ idd = "id";

if (!flag) {

dataGridView2->Rows->Add(idcount++, str, idd += idcount - 1);

idd = "id";

}

for (int i = 0; i < idcount - 1; i++) {

if (dataGridView2[1, i]->Value->ToString() == str) {

psevdocode += dataGridView2[2, i]->Value->ToString() + " ";

dscode += "(20."; dscode += dataGridView2[0, i]->Value->ToString() + ") ";

}

}

flag = false;

}

if (S == 15 || S == 16 || S == 17) {

bool flag = false;

for (int i = 0; i < numcount - 1; i++) {

if (dataGridView3[1, i]->Value->ToString() == str) {

flag = true;

}

}

if (!flag)dataGridView3->Rows->Add(numcount++, str, str);

psevdocode += str + " ";

for (int i = 0; i < numcount - 1; i++) {

if (dataGridView3[1, i]->Value->ToString() == str) {

dscode += "(30."; dscode += dataGridView3[0, i]->Value->ToString() + ") ";

}

}

flag = false;

}

if (S == 45){

bool flag = false;

for (int i = 0; i < symcount - 1; i++)

{

if (dataGridView4[1, i]->Value->ToString() == str) flag = true;

}

if (!flag) dataGridView4->Rows->Add(symcount++, str, str);

psevdocode += str + " ";

for (int i = 0; i < symcount - 1; i++){

if (dataGridView4[1, i]->Value->ToString() == str){

dscode += "(40."; dscode += dataGridView4[0, i]->Value->ToString() + ") ";

}

}

flag = false;

}

if (S == 19 || S == 20 || S == 21 || S == 22 || S == 35 || S == 25 || S == 27 || S == 28 || S == 33 || S == 34) {

bool flag = false;

for (int i = 0; i < opcount - 1; i++) {

if (dataGridView5[1, i]->Value->ToString() == str) flag = true;

}

if (!flag) dataGridView5->Rows->Add(opcount++, str, str);

psevdocode += str + " ";

for (int i = 0; i < opcount - 1; i++) {

if (dataGridView5[1, i]->Value->ToString() == str) {

dscode += "(50."; dscode += dataGridView5[0, i]->Value->ToString() + ") ";

}

}

flag = false;

}

str = "";

S = 0; i--;

}

private: System::Void label1\_Click(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void dataGridView3\_CellContentClick(System::Object^ sender, System::Windows::Forms::DataGridViewCellEventArgs^ e) {

}

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

String^ str;

str = richTextBox1->Text;

str += " ";

String^ alph = "AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz";

String^ symbols = ":;,.^'()[]{}#\"";

String^ numbers = "1234567890";

String^ fullAlph = alph + numbers;

String^ tmp;

int strCountr = 0;

int S = 0;

richTextBox2->Text += ++strCountr + ". ";

int i = 0;

while (i < str->Length) {

bool flag;

switch (S) {

case 0:

if (str[i] == '1' || str[i] == '2' || str[i] == '3' || str[i] == '4' || str[i] == '5' || str[i] == '6' || str[i] == '7' || str[i] == '8' || str[i] == '9' || str[i] == '0') {

S = 15;

tmp += str[i];

}

if (str[i] == 'l' || str[i] == 'f') {

S = 1;

tmp += str[i];

}

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

S = 5;

tmp += str[i];

}

if (str[i] == 'c') {

S = 6;

tmp += str[i];

}

if (str[i] == 't') {

S = 11;

tmp += str[i];

}

if (str[i] == 'a') {

S = 13;

tmp += str[i];

}

if (str[i] == 'e') {

S = 50;

tmp += str[i];

}

if (str[i] == 's') {

S = 55;

tmp += str[i];

}

if (str[i] == 'x') {

S = 60;

tmp += str[i];

}

if (str[i] == 'b') {

S = 62;

tmp += str[i];

}

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

S = 66;

tmp += str[i];

}

if (str[i] == '-') {

S = 19;

tmp += str[i];

}

if (str[i] == '+') {

S = 20;

tmp += str[i];

}

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

S = 21;

tmp += str[i];

}

if (str[i] == '/') {

S = 22;

tmp += str[i];

}

if (str[i] == '=' || str[i] == '!') {

S = 25;

tmp += str[i];

}

if (str[i] == '>') {

S = 27;

tmp += str[i];

}

if (str[i] == '<') {

S = 28;

tmp += str[i];

}

if (str[i] == '&') {

S = 33;

tmp += str[i];

}

if (str[i] == '|') {

S = 34;

tmp += str[i];

}

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

S = 30;

tmp += str[i];

}

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

S = 31;

tmp += str[i];

}

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

S = 32;

tmp += str[i];

}

if (S == 0) {

bool flag = false;

for (int j = 0; j < alph->Length; j++) {

if (str[i] == alph[j]) {

flag = true;

}

}if (flag) {

S = 40;

tmp += str[i];

}

else {

bool flag = false;

for (int j = 0; j < symbols->Length; j++) {

if (str[i] == symbols[j]) {

flag = true;

}

}if (flag) {

S = 45;

tmp += str[i];

}

else {

tmp += str[i];

output(tmp, S, i, strCountr);

}

}

}

break;

case 1:

if (str[i] == 'o') {

S = 2;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 2:

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

S = 3;

tmp += str[i];

}

else if (str[i] == 'r') {

S = 4;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 3:

if (str[i] == 'g') {

S = 4;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 4:

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

S = 40;

}

else {

output(tmp, S, i, strCountr);

}

flag = false;

break;

case 5:

if (str[i] == 'f') {

S = 4;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 6:

if (str[i] == 'a') {

S = 7;

tmp += str[i];

}

else if (str[i] == 'h') {

S = 9;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 7:

if (str[i] == 's') {

S = 8;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 8:

if (str[i] == 'e') {

S = 4;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 9:

if (str[i] == 'a') {

S = 10;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 10:

if (str[i] == 'r') {

S = 4;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 11:

if (str[i] == 'r') {

S = 12;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 12:

if (str[i] == 'y') {

S = 4;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 13:

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

S = 14;

tmp += str[i];

}

else {

S = 40;//25??

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 14:

if (str[i] == 'd') {

S = 4;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 50:

if (str[i] == 'x') {

S = 51;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 51:

if (str[i] == 'p') {

S = 52;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 52:

if (str[i] == 'o') {

S = 53;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 53:

if (str[i] == 'r') {

S = 54;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 54:

if (str[i] == 't') {

S = 4;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 55:

if (str[i] == 't') {

S = 56;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 56:

if (str[i] == 'r') {

S = 57;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 57:

if (str[i] == 'u') {

S = 58;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 58:

if (str[i] == 'c') {

S = 59;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 59:

if (str[i] == 't') {

S = 4;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 60:

if (str[i] == 'o') {

S = 61;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 61:

if (str[i] == 'r') {

S = 4;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 62:

if (str[i] == 'r') {

S = 63;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 63:

if (str[i] == 'e') {

S = 64;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 64:

if (str[i] == 'a') {

S = 65;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 65:

if (str[i] == 'k') {

S = 4;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 66:

if (str[i] == 'e') {

S = 67;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 67:

if (str[i] == 'w') {

S = 4;

tmp += str[i];

}

else {

S = 40;

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 15:

if (str[i] == '1' || str[i] == '2' || str[i] == '3' || str[i] == '4' || str[i] == '5' || str[i] == '6' || str[i] == '7' || str[i] == '8' || str[i] == '9' || str[i] == '0') {

S = 15;

tmp += str[i];

}

else if (str[i] == '.') {

S = 16;

tmp += str[i];

}

else if (str[i] == 'e') {

S = 17;

tmp += str[i];

}

else {

bool flag = false;

for (int j = 0; j < alph->Length; j++) {

if (str[i] == alph[j]) {

flag = true;

}

}if (flag) {

S = 18;

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 16:

if (str[i] == '1' || str[i] == '2' || str[i] == '3' || str[i] == '4' || str[i] == '5' || str[i] == '6' || str[i] == '7' || str[i] == '8' || str[i] == '9' || str[i] == '0') {

S = 16;

tmp += str[i];

}

else if (str[i] == 'e') {

S = 17;

tmp += str[i];

}

else {

bool flag = false;

for (int j = 0; j < alph->Length; j++) {

if (str[i] == alph[j]) {

flag = true;

}

}if (flag) {

S = 18;

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 17:

if (str[i] == '-' || str[i] == '+') {

S = 15;

tmp += str[i];

}

else {

bool flag = false;

for (int j = 0; j < alph->Length; j++) {

if (str[i] == alph[j]) {

flag = true;

}

}if (flag) {

S = 18;

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

}

flag = false;

break;

case 18:

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) {

flag = true;

}

}if (flag) {

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

flag = false;

break;

case 19:

if (str[i] == '1' || str[i] == '2' || str[i] == '3' || str[i] == '4' || str[i] == '5' || str[i] == '6' || str[i] == '7' || str[i] == '8' || str[i] == '9' || str[i] == '0') {

S = 15;

tmp += str[i];

}

else if (str[i] == '-' || str[i] == '=') {

S = 35;

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

break;

case 20:

if (str[i] == '+' || str[i] == '=') {

S = 35;

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

break;

case 21:

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

S = 35;

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

break;

case 22:

if (str[i] == '/') {

S = 24;

}

else if (str[i] == '=') {

S = 35;

tmp += str[i];

}

else if (str[i] == '\*') {

S = 23;

}

else {

output(tmp, S, i, strCountr);

}

break;

case 23:

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

S = 36;

}

else {

S = 23;

}

break;

case 36:

if (str[i] == '/') {

S = 37;

}

break;

case 37:

tmp = " ";

output(tmp, S, i, strCountr);

break;

case 24:

tmp = "";

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

output(tmp, S, i, strCountr);

}

break;

case 25:

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

S = 35;

tmp += str[i];

}

else if (str[i] == '>' || str[i] == '<') {

S = 26;

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

break;

case 26:

output(tmp, S, i, strCountr);

break;

case 27:

if (str[i] == '=' || str[i] == '>') {

S = 35;

tmp += str[i];

}

else if (str[i] == '<') {

S = 29;

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

break;

case 28:

if (str[i] == '=' || str[i] == '<') {

S = 35;

tmp += str[i];

}

else if (str[i] == '>') {

S = 29;

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

break;

case 29:

output(tmp, S, i, strCountr);

break;

case 30:

if (str[i] != ' ') {

output(tmp, S, i, strCountr);

}

break;

case 31:

if (str[i] != ' ') {

output(tmp, S, i, strCountr);

}

break;

case 32:

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

output(tmp, S, i, strCountr);

}

break;

case 33:

if (str[i] == '&') {

S = 35;

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

break;

case 34:

if (str[i] == '|') {

S = 35;

tmp += str[i];

}

else {

output(tmp, S, i, strCountr);

}

break;

case 40:

flag = false;

for (int j = 0; j < fullAlph->Length; j++) {

if (str[i] == fullAlph[j]) flag = true;

}

if (flag) { tmp += str[i]; }

else { output(tmp, S, i, strCountr); }

flag = false;

break;

case 45:

output(tmp, S, i, strCountr);

break;

case 35:

output(tmp, S, i, strCountr);

break;

}

i++;

}

richTextBox3->Text = psevdocode;

richTextBox4->Text = dscode;

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

numcount = 1, keywordcount = 1, idcount = 1, opcount = 1, symcount = 1;

psevdocode = "";

dscode = "";

dataGridView1->Rows->Clear();

dataGridView2->Rows->Clear();

dataGridView3->Rows->Clear();

dataGridView4->Rows->Clear();

dataGridView5->Rows->Clear();

richTextBox2->Clear();

richTextBox3->Clear();

richTextBox4->Clear();

richTextBox5->Clear();

}

};

}