минобрнауки россии

федеральное государственное бюджетное

образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

|  |  |  |
| --- | --- | --- |
| Институт (факультет) | | Институт информационных технологий |
| Кафедра | Математического и программного обеспечения ЭВМ | |

КУРСОВАЯ РАБОТА

|  |  |
| --- | --- |
| по дисциплине | Модуль: Информатика |

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

|  |
| --- |
| Выполнил студент группы |
| 1ПИб-02-2оп-2 |
| направление подготовки (специальности) |
| 09.03.04., Программная инженерия |
| шифр, наименование |
| Зернов Владислав Александрович |
| фамилия, имя, отчество |

|  |
| --- |
| Руководитель  Ганичева Оксана Георгиевна |
|  |
| фамилия, имя, отчество  доцент, кандидат наук |
|  |
| должность |
| Руководитель  Пышницкий Константин Михайлович |
|  |
| фамилия, имя, отчество  старший преподователь |
|  |
| должность |

|  |
| --- |
| Дата представления работы |
| « » \_\_\_­­­­­\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |
| Заключение о допуске к защите |
|  |
|  |
|  |
| Оценка \_\_\_\_\_, \_\_\_\_\_\_ количество баллов |
|  |
| Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Череповец, 2024

Оглавление

[1. Введение 3](#_Toc138127092)

[2. Описание предметной области 5](#_Toc138127093)

[3. Основная часть 14](#_Toc138127094)

[4. Логическое проектирование 40](#_Toc138127095)

[5. Физическое проектирование 49](#_Toc138127096)

[6. Проектирование интерфейса 51](#_Toc138127097)

[7. Тестирование программного обеспечения 52](#_Toc138127098)

[8. Заключение. 57](#_Toc138127099)

[9. Список литературы. 58](#_Toc138127100)

[Приложение 1. Техническое задание 59](#_Toc138127101)

[Приложение 2. Текст программы 68](#_Toc138127102)

[Приложение 3. Руководство пользователя 122](#_Toc138127104)

[Приложение 4. Таблица переходов 124](#_Toc138127106)

[Приложение 5. Граф 125](#_Toc138127107)

Приложение 6. Дерево порождения..................................................................126

# 1. Введение

Каждая электронная-вычислительная машина имеет свой машинный язык, язык низкого уровня. Эта машина может исполнять программы, написанные только на этом машинном языке. Любой алгоритм можно написать на этом языке, но подобная задача довольно сложная, потому что для ее выполнения необходимо знать устройство и функционал этой электронно-вычислительной машины. Языки высокого уровня позволяют решить поставленную задачу. Они обладают развитой структурой данных и средствами их обработки. Языки высокого уровня упрощаю работу специалистов и разработчиков.

Язык высокого уровня – это входной язык, описанная на языке высокого уровня – исходная программа.

Для перевода программы с высокоуровневого языка на машинный язык нужен языковой процессор. Языковой процессор – это программа на машинном языке, которая позволяет электронно-вычислительной машине понимать и выполнять программы на входном языке. Языковые процессоры делятся на интерпретаторы и трансляторы.

Интерпретаторы – программа, на вход которой поступает код на входном языке, распознает и реализует конструкции этого входного языка, выдает на выходе результаты вычислений.

Транслятор – программа, которая на выход выдает программу на объектном(машинном) языке эквивалентную программе, которая подавалась на вход.

Компилятор – транслятор для языка высокого уровня.

Компиляция состоит из двух основных этапов: анализ и синтез. Результатом компиляции является объектный код.

Анализ включает в себя лексический, синтаксический и семантический анализы, а синтез – подготовку к генерации, саму генерацию и оптимизацию.

На этапе анализа происходит распознавание текста программы, создание и заполнение таблицы идентификаторов. Т.е. создание некоторого внутреннего представления, понятного компилятору.

Таблица идентификаторов (символов) – специальным образом организованные наборы данных, служащие для хранения информации об элементах исходной программы, которые потом используются для порождения текста результирующей программы.

На этапе синтеза на основании внутреннего представления программы и информации, содержащейся в таблице символов, порождается текс результирующей программы, т.е. объектный код.

# 2. Описание предметной области

В курсовой работе создается модель лексического анализатора.

Лексический анализ – это первая фаза компиляции, процесс предварительной обработки исходной программы, на котором лексические единицы программы – лексемы приводятся к единому формату и заменяются условными кодами или ссылками на соответствующие таблицы.

Лексема – это структурная единица языка, которая состоит из элементарных символов языка и не содержит в своем составе другие структурные единицы. Лексемами языка программирования являются идентификаторы, константы, ключевые слова, операторы и т.д.

Состав лексем определяется синтаксисом языка.

Цели лексического анализа:

1. Перевод исходной программы на язык, понятный для компилятора, приведение всех лексем к единому формату и замена этих лексем дескрипторами.
2. Лексический контроль – выявление в программе недопустимых слов.

Лексический анализатор – это часть компилятора, которая читает исходную программу, выделяет в ее тексте лексемы.

На выходе информация, которая поступает н вход лексического анализатора – промежуточное представление.

Дескрипторы – это условные коды, на которые заменяются лексемы. Дескрипторы делятся на два типа: числовой и символьный. Дескрипторы выглядят следующим образом: тип\_лексемы, указатель на лексему.

Теоретически лексический анализатор не является обязательной частью компилятора, т.к. его функции могут быть выполнены на этапе синтаксического анализатора. Но все-таки чаще всего лексический анализатор реализуется отдельно. На это есть следующие причины:

1. Отдельно реализованный лексический анализатор упрощает работу с текстом исходной программы на этапе синтаксического анализа, т.к. на этапе лексического анализатора код исходной программы очищается от лишних комментариев, пробелов и переносов строки. После лексического анализа взаимодействовать с текстом проще и удобнее.
2. Увеличение эффективности компилятора. Отдельный лексический анализатор позволяет создать специализированный и потенциально более эффективный процессор для решения поставленной задачи.
3. Увеличение переносимости компилятора. Особенности входного алфавита и другие специфичные характеристики используемых устройств могут ограничивать возможности лексического анализатора.

Набор функций лексического анализатора определяется разработчиком и могут меняться в зависимости от версии компилятора и нужд.

В стандартный набор функций лексического анализатора входят:

1. Удаление лишних пробелов;
2. удаление пустых строк;
3. удаление комментариев;
4. выделение лексем;
5. подсчет строк.

В большинстве компиляторов лексический анализатор и синтаксический анализатор являются взаимосвязанными частями. У них есть несколько вариантов взаимодействия:

1. Последовательное взаимодействие

Лексический анализатор просматривает весь текст исходной программы от начала и до конца, преобразует его структурированный набор данных (таблицу лексем). В таблице лексем все лексемы заменены на специально оговоренные коды.

В этом случае лексический анализатор просматривает текст исходной программы 1 раз и таблица лексем составляется сразу вся, целиком.

Дальнейшая обработка выполняется на следующих фазах компиляции.

1. Параллельное взаимодействие (рис.1)

Лексический анализ исходного текста выполняется поэтапно так, что синтаксический анализ выполняет разбиение очередной конструкции языка, обращается к сканеру за следующей лексемой. При этом он может передавать информацию о том, какую лексему ему необходимо ожидать. При ошибке может происходить «откат» чтобы попытаться выполнить анализ на другой основе.

Только после успешного синтаксического анализа, конструктор лексического анализатора помещает найденные лексемы в таблицу символов и продолжает разбор. Иногда лексический анализатор называют «сканером»

Параллельное взаимодействие анализаторов используется наиболее часто, так как является более эффективным и простым.

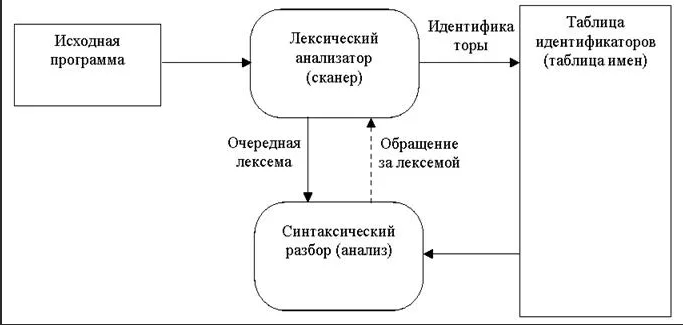


Рис.1. Параллельное взаимодействие лексического и синтаксического анализаторов

Определение границ лексемы – выделение тех строк в общем потоке, для которых нужно выполнить распознавание. Зачастую данная вызывает определенную проблему. В большинстве языков границы лексем задаются терминальными символами, ими могут быть пробелы, запятые, знаки операций, символы комментариев и т.д.

Лексический анализ, как правило, работает по следующему принципу:

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

При этом от пользователя требуется обозначить границы лексем пробелами. Данный подход возможен для большинства входных языков.

Способы реализации лексического анализатора:

1. Использование генератора лексических анализаторов (например LEX).

Строится на основе специфик, на основе регулярных выражений. В этом случае входная лексема сравнивается с шаблоном, подготовленным заранее. Генератор предоставляет функции для чтения и буферизации ввода.

1. Написание лексического анализатора на подходящем языке программирования с использованием возможностей ввода-вывода данного языка для чтения входной информации.
2. Написание лексического анализатора на языке ассемблера и явное управление процессом чтения входной информации.

Способы расположены в порядке усложнения их реализации.

Лексический анализатор и конечный автомат:

Разработка лексического анализатора выполняется с использованием математического аппарата, а именно: теории регулярных языков и конечного автомата.

Конечный автомат — 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.

Регулярные выражения – это способ записи, который позволяет точно определить множества слов, принадлежащих языку. Это формула(шаблон), точно показывающая, как было построено соответствующее ей регулярное множество с помощью операций. Эта формула конечна.

Построенная регулярная грамматика является источником, по которому в дальнейшем конструируется вычислительное устройство, реализующее функцию распознавания языка, порождаемого этой грамматикой.

Для регулярных языков таким устройством является конечный автомат. Язык называется автоматным (регулярным) и относится к 3 классу языков.

Формальными порождаемыми грамматиками называются 4 компонента вида:

G = (N, T, P, S), где

N – конечное непустое множество нетерминальных символов, образующих нетерминальный словарь грамматики. Нетерминалы представляют собой переменные, которые обозначают множество строк, которые определяют язык, порожденный грамматикой.

T – конечное не пустое множество терминальных символов, которые образуют терминальный словарь грамматики. Терминал представляют собой базовые символы, из которых формируются строки.

P – множество продукций грамматики. Продукция определяет способ, которым терминалы и нетерминалы могут объединяться для создания строк.

S – начальный/стартовый символ грамматики (аксиома грамматики). Стартовый символ всегда является нетерминалом.

Цепочки символов обозначаются греческими буквами. Цепочка – это некоторая упорядоченная последовательность символов из алфавита нетерминалов и терминалов.

Язык – это множество всех цепочек терминалов, вывод из аксиомы грамматики (язык, выводимый этой грамматикой).

Таким образом грамматика выводит или порождает строки, начиная с аксиомы, неоднократно замещая нетерминалы правыми частями продукций нетерминалов.

Конечный распознаватель – конечный автомат, результатом работы которого является указание на то, допустима ли входная последовательность символов.

Синтаксический анализатор (синтаксический разбор) - это часть компилятора, которая отвечает за выявление основных синтаксических конструкций входного языка.

В задачу синтаксического анализа входит:

1. найти и выделить основные синтаксические конструкции в тексте входной программы,

2. установить тип и проверить правильность каждой синтаксической конструкции,

3. представить синтаксические конструкции в виде, удобном для дальнейшей генерации текста результирующей программы.

В основе синтаксического анализатора лежит распознаватель текста входной программы на основе грамматики входного языка. Как правило, синтаксические конструкции языков программирования могут быть описаны с помощью КС-грамматик, реже встречаются языки, которые могут быть описаны с помощью регулярных грамматик.

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

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

Выходом лексического анализатора является таблица лексем (или цепочка лексем). Эта таблица образует вход синтаксического анализатора, который исследует только один компонент каждой лексемы — ее тип. Остальная информация о лексемах используется на более поздних фазах компиляции при семантическом анализе, подготовке к генерации и генерации кода результирующей программы.

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

Основу любого синтаксического анализатора всегда составляет распознаватель, построенный на основе какого-либо класса КС-грамматик. Поэтому главную роль в том, как функционирует синтаксический анализатор и какой алгоритм лежит в его основе, играют принципы построения распознавателей КС-языков. Без применения этих принципов невозможно выполнить эффективный синтаксический разбор предложений входного языка.

Синтаксис конструкций языка программирования может быть описан с помощью контекстно-свободных грамматик или нотации БНФ (форма Бэкуса Наура). Грамматики обеспечивают значительные преимущества разработчикам языков программирования и создателям компиляторов:

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

• Для некоторых классов грамматик можно автоматически построить эффективный синтаксический анализатор, который определяет, корректна ли структура исходной программы;

• Правильно построенная грамматика придает языку программирования структуру, которая способствует облегчению трансляции исходной программы в объектный код, выявлению ошибок.

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

# 3. Основная часть

1. Постановка задачи



Рис.2. Вариант задания

Строки в языке C++ могут быть представлены как массивы символов или объектами класса std::string, который является частью стандартной библиотеки. Класс std::string предоставляет множество функций для обработки строк.

Цикл for используется для многократного выполнения блока кода, пока выполняется заданное условие. Он особенно полезен при работе с массивами, векторами и другими коллекциями данных.

Тестовая программа принимает на вход строку и конвертирует обычную строку в строку типа camelCase. Также программа считает количество гласных в строке от пользователя. Результат программы выводится в консоль.

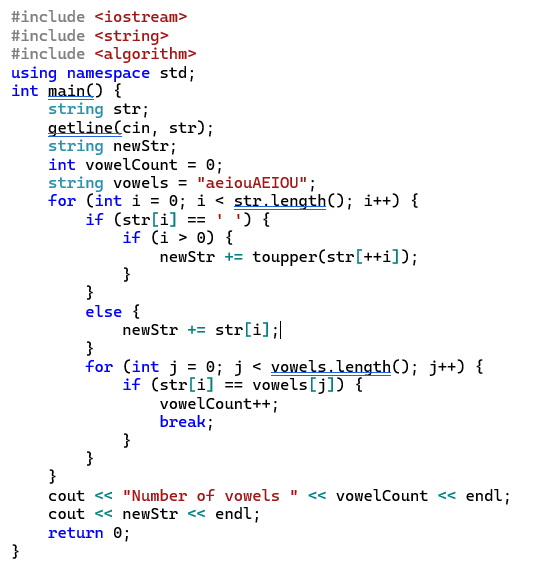


Рис.3. Код тестовой программы

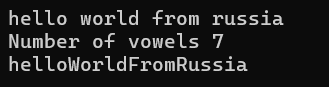


Рис. 4. Результат работы тестовой программы

1. Описание конечного автомата

Конечный автомат (КА) — математическая абстракция, модель дискретного устройства, имеющего один вход, один выход и в каждый момент времени, находящегося в одном состоянии из множества возможных. детерминированные КА — автоматы, в которых следующее состояние однозначно определяется текущим состоянием и входным символом и выход зависит только от текущего состояния и текущего входа.

Конечный автомат задается выражение A = (X, S, F, S0, ∂), где

X – конечное непустое множество входных символов

S – конечное множество состояний

F – множество выделенных заключительных состояний

S0 – начальное состояние автомата

∂ - отображение вида (S, X) -> S

A = (X, S, F, S0, ∂), где

X = {a..z, 0..9, (, ), {, }, [, ], <, >, -, =, +, /, :, ; }

S ={S0}

F = {S3, S35, S39, S45, S46, S47}, где

S3 – ключевые слова;

S45 –идентификаторы;

S47 – константы;

S46 – разделители;

S35 – знаки операций;

S39 – знаки отношений

∂ = {

|  |
| --- |
| (S1;i) -> S2 |
| (S1;e) -> S5 |
| (S1;f) -> S8 |
| (S1;w) -> S10 |
| (S1;s) -> S14 |
| (S1;r) -> S23 |
| (S1;c) -> S31 |
| (S1;d) -> S50 |
| (S1;u) -> S65 |
| (S1;n) -> S69 |
| (S1;p) -> S80 |
| (S1;a) -> S96 |
| (S1;t) -> S99 |
| (S1;b) -> S108 |
| (S1;g) -> S134 |
| (S1;l) -> S137 |
| (S1;o) -> S149 |
| (S1;v) -> S170 |
| (S1;m) -> S196 |
| (S1;x) -> S208 |
| (S1;>) -> S34 |
|  |
| (S1;<) -> S36 |  |
|  |
| (S1;=) -> S37 |  |
|  |
| (S1;!) -> S38 |  |
|  |
| (S1;+) -> S40 |  |
|  |
| (S1;-) -> S41 |  |
|  |
| (S1;\*) -> S42 |  |
|  |
| (S1;/) -> S43 |  |
|  |
| (S1;%) -> S44 |  |
|  |
| (S1;0..9) -> S47 |  |
|  |
| (S1;") -> S47 |  |
| (S1;') -> S47 |  |
| (S1;() -> S46 |  |
| (S1;)) -> S46 |  |
| (S1;#) -> S46 |  |
| (S1;.) -> S46 |  |
| (S1;,) -> S46 |  |
| (S1;[) -> S46 |  |
| (S1;]) -> S46 |  |
| (S1;{) -> S46 |  |
| (S1;}) -> S46 |  |
| (S1;:) -> S46 |  |
| (S1;;) -> S46 |  |
| (S1;пробел) -> S46 |  |
| (S2;f) -> S3 |  |
| (S2;n) -> S4 |  |
| (S4;t) -> S3 |  |
| (S4;l) -> S184 |  |
| (S184, i) -> S185 |  |
| (S185, n) -> S186 |  |
| (S186, e) -> S3 |  |
| (S5;l) -> S6 |  |
| (S6;s) -> S7 |  |
| (S7;e) -> S3 |  |
| (S5;n) -> S61 |  |
| (S61;u) -> S62 |  |
| (S62;m) -> S3 |  |
| (S5;x) -> S178 |  |
| (S178;p) -> S179 |  |
| (S179;l) -> S180 |  |
| (S180;i) -> S181 |  |
| (S181;c) -> S182 |  |
| (S182;i) -> S183 |  |
| (S183;t) -> S3 |  |
| (S178;t) -> S193 |  |
| (S193;e) -> S194 |  |
| (S194;r) -> S195 |  |
| (S195;n) -> S3 |  |
| (S8;o) -> S9 |  |
| (S9;r) -> S3 |  |
| (S8;l) -> S28 |  |
| (S28, o) -> S29 |  |
| (S29, a) -> S30 |  |
| (S30, t) -> S3 |  |
| (S9, a) -> S127 |  |
| (S127, l) -> S128 |  |
| (S128, s) -> S129 |  |
| (S129, e) -> S3 |  |
| (S9, r) -> S130 |  |
| (S130, i) -> S131 |  |
| (S131, e) -> S132 |  |
| (S132, n) -> S133 |  |
| (S133, d) -> S3 |  |
| (S10, h) -> S11 |  |
| (S11, i) -> S12 |  |
| (S12, l) -> S13 |  |
| (S13, e) -> S3 |  |
| (S10, c) -> S211 |  |
| (S211, h) -> S212 |  |
| (S213, a) -> S214 |  |
| (S214, r) -> S215 |  |
| (S215, \_) -> S216 |  |
| (S217, t) -> S3 |  |
| (S14, t) -> S15 |  |
| (S15, r) -> S16 |  |
| (S16, i) -> S17 |  |
| (S17, n) -> S18 |  |
| (S18, g) -> S3 |  |
| (S14, w) -> S19 |  |
| (S19, i) -> S20 |  |
| (S20, t) -> S21 |  |
| (S21, c) -> S22 |  |
| (S22, h) -> S3 |  |
| (S16, u) -> S63 |  |
| (S63, c) -> S64 |  |
| (S64, t) -> S3 |  |
| (S15, a) -> S77 |  |
| (S77, t) -> S78 |  |
| (S78, i) -> S79 |  |
| (S79, c) -> S3 |  |
| (S14, h) -> S140 |  |
| (S140, o) -> S141 |  |
| (S141, r) -> S142 |  |
| (S142, t) -> S3 |  |
| (S14, i) -> S164 |  |
| (S164, z) -> S165 |  |
| (S165, e) -> S166 |  |
| (S166, o) -> S167 |  |
| (S167, f) -> S3 |  |
| (S23, e) -> S24 |  |
| (S24, t) -> S25 |  |
| (S25, u) -> S26 |  |
| (S26, r) -> S27 |  |
| (S27, n) -> S3 |  |
| (S31, h) -> S32 |  |
| (S32, a) -> S33 |  |
| (S33, r) -> S3 |  |
| (S31, o) -> S55 |  |
| (S55, n) -> S56 |  |
| (S56, s) -> S57 |  |
| (S57, t) -> S3 |  |
| (S31, l) -> S58 |  |
| (S58, a) -> S59 |  |
| (S60, s) -> S61 |  |
| (S61, s) -> S3 |  |
| (S31, a) -> S101 |  |
| (S101, t) -> S102 |  |
| (S102, c) -> S103 |  |
| (S103, h) -> S3 |  |
| (S101, s) -> S107 |  |
| (S107, e) -> S3 |  |
| (S56, t) -> S114 |  |
| (S114, i) -> S115 |  |
| (S115, n) -> S116 |  |
| (S116, u) -> S117 |  |
| (S117, e) -> S118 |  |
| (S34, =) -> S35 |  |
| (S34, >) -> S39 |  |
| (S36, =) -> S35 |  |
| (S36, <) -> S39 |  |
| (S38, =) -> S35 |  |
| (S37, =) -> S35 |  |
| (S40, =) -> S39 |  |
| (S40, +) -> S390 |  |
| (S41, =) -> S39 |  |
| (S41, -) -> S390 |  |
| (S42, =) -> S39 |  |
| (S43, =) -> S39 |  |
| (S44, =) -> S39 |  |
| (S45, A..z) -> S45 |  |
|  |
| (S50, o) -> S51 |  |
|  |
| (S51, u) -> S52 |  |
| (S52, b) -> S53 |  |
| (S53, l) -> S54 |  |
| (S54, e) -> S3 |  |
| (S50, e) -> S118 |  |
| (S118, f) -> S119 |  |
| (S119, a) -> S120 |  |
| (S120, u) -> S121 |  |
| (S121, l) -> S122 |  |
| (S122, t) -> S3 |  |
| (S118, e) -> S123 |  |
| (S123, l) -> S124 |  |
| (S124, e) -> S125 |  |
| (S125, t) -> S126 |  |
| (S126, e) -> S3 |  |
| (S65, s) -> S66 |  |
| (S66, i) -> S67 |  |
| (S67, n) -> S68 |  |
| (S68, g) -> S3 |  |
| (S65, n) -> S216 |  |
| (S216, i) -> S217 |  |
| (S217, o) -> S218 |  |
| (S218, n) -> S3 |  |
| (S69, a) -> S70 |  |
| (S70, m) -> S71 |  |
| (S71, e) -> S72 |  |
| (S72, s) -> S73 |  |
| (S73, p) -> S74 |  |
| (S74, a) -> S75 |  |
| (S75, c) -> S76 |  |
| (S76, e) -> S3 |  |
| (S69, e) -> S168 |  |
| (S168, w) -> S3 |  |
| (S69, u) -> S187 |  |
| (S187, l) -> S188 |  |
| (S188, l) -> S189 |  |
| (S189, p) -> S190 |  |
| (S190, t) -> S191 |  |
| (S191, r) -> S3 |  |
| (S69, o) -> S202 |  |
| (S202, t) -> S3 |  |
| (S202, e) -> S203 |  |
| (S203, x) -> S204 |  |
| (S204, c) -> S205 |  |
| (S205, e) -> S206 |  |
| (S206, p) -> S207 |  |
| (S207, t) -> S3 |  |
| (S80, u) -> S81 |  |
| (S81, b) -> S82 |  |
| (S82, l) -> S83 |  |
| (S83, i) -> S84 |  |
| (S84, c) -> S3 |  |
| (S80, r) -> S85 |  |
| (S85, i) -> S86 |  |
| (S86, v) -> S87 |  |
| (S87, a) -> S88 |  |
| (S88, t) -> S89 |  |
| (S89, e) -> S3 |  |
| (S85, o) -> S90 |  |
| (S90, t) -> S91 |  |
| (S91, e) -> S92 |  |
| (S92, c) -> S93 |  |
| (S93, t) -> S94 |  |
| (S94, e) -> S95 |  |
| (S95, d) -> S3 |  |
| (S96, u) -> S97 |  |
| (S97, t) -> S98 |  |
| (S98, o) -> S3 |  |
| (S96, n) -> S192 |  |
| (S192, d) -> S3 |  |
| (S99, r) -> S100 |  |
| (S100, y) -> S3 |  |
| (S99, h) -> S104 |  |
| (S104, r) -> S105 |  |
| (S105, o) -> S106 |  |
| (S106, w) -> S3 |  |
| (S100, u) -> S126 |  |
| (S126, e) -> S3 |  |
| (S99, e) -> S143 |  |
| (S143, m) -> S144 |  |
| (S144, p) -> S145 |  |
| (S145, l) -> S146 |  |
| (S146, a) -> S147 |  |
| (S147, t) -> S148 |  |
| (S148, e) -> S3 |  |
| (S99, y) -> S156 |  |
| (S156, p) -> S157 |  |
| (S157, e) -> S158 |  |
| (S158, d) -> S159 |  |
| (S159, e) -> S160 |  |
| (S160, f) -> S3 |  |
| (S158, n) -> S161 |  |
| (S161, a) -> S162 |  |
| (S162, m) -> S163 |  |
| (S163, e) -> S3 |  |
| (S158, i) -> S210 |  |
| (S210, d) -> S3 |  |
| (S104, i) -> S169 |  |
| (S169, s) -> S3 |  |
| (S134, o) -> S135 |  |
| (S135, t) -> S136 |  |
| (S136, o) -> S3 |  |
| (S137, o) -> S138 |  |
| (S138, n) -> S139 |  |
| (S139, g) -> S3 |  |
| (S170, i) -> S171 |  |
| (S171, r) -> S172 |  |
| (S172, t) -> S173 |  |
| (S173, u) -> S174 |  |
| (S174, a) -> S175 |  |
| (S175, l) -> S3 |  |
| (S170, o) -> S176 |  |
| (S176, i) -> S177 |  |
| (S177, d) -> S3 |  |
| (S149, r) -> S3 |  |

}

1. Построение грамматики по конечному автомату

Понятие грамматики изначально было формализовано лингвистами при изучении естественных языков. Предполагалось, что это мюжет помочь при их автоматической трансляции. Однако, наилучшие результаты в этом направлении достигнуты при описании не естественных языков, а языков программирования.

Грамматика это четверка G=(N,T,P,S)

N-афлаваит нетерминальных символов

Нетерминал (нетерминальный символ) — объект, обозначающий какую-либо сущность языка (например: формула, арифметическое выражение, команда) и не имеющий конкретного символьного значения.

Т-афлаваит терминальных символов

Терминал (терминальный символ) — объект, непосредственно присутствующий в словах языка, соответствующего грамматике, и имеющий конкретное, неизменяемое значение (обобщение понятия «буквы»). Базовые символы, из которых формируются строки.

P - конечное множество правил вывода, называемых продукциями.

S - начальный символ грамматики (аксиома языка) (S∈N);

По иерархии Хомского, грамматики делятся на 4 типа, каждый последующий является более ограниченным подмножеством предыдущего (но и легче поддающимся анализу):тип 0. неограниченные грамматики — возможны любые правилатип 1. контекстно-зависимые грамматики — левая часть может содержать один нетерминал, окруженный «контекстом» (последовательности символов, в том же виде присутствующие в правой части); сам нетерминал заменяется непустой последовательностью символов в правой части. Тип 2. контекстно-свободные грамматики — левая часть состоит из одного нетерминала. Тип 3. регулярные грамматики — более простые, эквивалентны конечным автоматам.

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

G=(N,T,P,S)

T = {a..z, 0..9, (, ), {, }, [, ], <, >, -, =, +, /, :, ; }

N ={S0…S218}

S=S0

F = {S3, S45, S46, S47, S35, S39}, где

S3 – ключевые слова;

S45 –идентификаторы;

S47 – константы;

S46 – разделители;

S35 – знаки операций;

S39 – знаки отношений

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.

Таблица 1.

Формальное описание конечного автомата и регулярной грамматики по заданному конечному автомату

|  |  |
| --- | --- |
| Переход по конечному автомату | Переход по регулярной грамматике |
| 1 | 2 |
| (S1;i) -> S2 | S1 -> i |
| (S1;e) -> S5 | S1 -> e |
| (S1;f) -> S8 | S1 -> f |
| (S1;w) -> S10 | S1 -> w |
| (S1;s) -> S14 | S1 -> s |
| (S1;r) -> S23 | S1 -> r |
| (S1;c) -> S31 | S1 -> c |
| (S1;d) -> S50 | S1 -> d |
| (S1;u) -> S65 | S1 -> u |
| (S1;n) -> S69 | S1 -> n |
| (S1;p) -> S80 | S1 -> p |
| (S1;a) -> S96 | S1 -> a |
| (S1;t) -> S99 | S1 -> t |
| (S1;b) -> S108 | S1 -> b |
| (S1;g) -> S134 | S1 -> g |
| (S1;l) -> S137 | S1 -> l |
| (S1;o) -> S149 | S1 -> o |
| (S1;v) -> S170 | S1 -> v |
| (S1;m) -> S196 | S1 -> m |
| (S1;x) -> S208 | S1 -> x |
| (S1;>) -> S34 | S1 -> > |
| S1-> > S34 |
| (S1;<) -> S36 | S1 -> < |
| S1-> < S34 |

Продолжение табл. 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S1;<) -> S36 | S1 -> < |
| S1-> < S34 |
| (S1;=) -> S37 | S1 -> = |
| S1 -> = S37 |
| (S1;!) -> S38 | S1 -> ! |
| S1 -> ! S38 |
| (S1;+) -> S40 | S1 -> + |
| S1 -> + S40 |
| (S1;-) -> S41 | S1 -> - |
| S1 -> - S41 |
| (S1;\*) -> S42 | S1 -> \* |
| S1 -> \* S42 |
| (S1;/) -> S43 | S1 -> / |
| S1 -> / S43 |
| (S1;%) -> S44 | S1 -> % |
| S1 -> % S44 |
| (S1;0..9) -> S47 | S1 -> 0..9 |
| S1 -> 0..9 S47 |
| (S1;") -> S47 | S1 -> " |
| (S1;') -> S47 | S1 -> ' |
| (S1;() -> S46 | S1 -> ( |
| (S1;)) -> S46 | S1 -> ) |
| (S1;#) -> S46 | S1 -> # |
| (S1;.) -> S46 | S1 -> . |
| (S1;,) -> S46 | S1 -> , |
| (S1;[) -> S46 | S1 -> [ |
| (S1;]) -> S46 | S1 -> ] |

Продолжение табл. 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S1;{) -> S46 | S1 -> { |
| (S1;}) -> S46 | S1 -> } |
| (S1;:) -> S46 | S1 -> : |
| (S1;;) -> S46 | S1 -> ; |
| (S1;пробел) -> S46 | S1 -> пробел |
| (S2;f) -> S3 | S2 -> f |
| (S2;n) -> S4 | S2 -> n |
| (S4;t) -> S3 | S4 -> t |
| (S4;l) -> S184 | S4 -> l |
| (S184, i) -> S185 | S184 -> i |
| (S185, n) -> S186 | S185 -> n |
| (S186, e) -> S3 | S186 -> e |
| (S5;l) -> S6 | S5 -> l |
| (S6;s) -> S7 | S6 -> s |
| (S7;e) -> S3 | S7 -> e |
| (S5;n) -> S61 | S5 -> n |
| (S61;u) -> S62 | S61 -> u |
| (S62;m) -> S3 | S62 -> m |
| (S5;x) -> S178 | S5 -> x |
| (S178;p) -> S179 | S178 -> p |
| (S179;l) -> S180 | S179 -> l |
| (S180;i) -> S181 | S180 -> i |
| (S181;c) -> S182 | S181 -> c |
| (S182;i) -> S183 | S182 -> i |
| (S183;t) -> S3 | S183 -> t |
| (S178;t) -> S193 | S178 -> t |
| (S193;e) -> S194 | S193 -> e |

Продолжение табл. 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S194;r) -> S195 | S194 -> r |
| (S195;n) -> S3 | S195 -> n |
| (S8;o) -> S9 | S8 -> o |
| (S9;r) -> S3 | S9 -> r |
| (S8;l) -> S28 | S9 -> l |
| (S28, o) -> S29 | S28 -> o |
| (S29, a) -> S30 | S29 -> a |
| (S30, t) -> S3 | S30 -> t |
| (S9, a) -> S127 | S9 -> a |
| (S127, l) -> S128 | S127 -> l |
| (S128, s) -> S129 | S128 -> s |
| (S129, e) -> S3 | S129 -> e |
| (S9, r) -> S130 | S9 -> r |
| (S130, i) -> S131 | S130 -> i |
| (S131, e) -> S132 | S131 -> e |
| (S132, n) -> S133 | S132 -> n |
| (S133, d) -> S3 | S133 -> d |
| (S10, h) -> S11 | S10 -> h |
| (S11, i) -> S12 | S11 -> i |
| (S12, l) -> S13 | S12 -> l |
| (S13, e) -> S3 | S13 -> e |
| (S10, c) -> S211 | S10 -> c |
| (S211, h) -> S212 | S211 -> h |
| (S213, a) -> S214 | S213 -> a |
| (S214, r) -> S215 | S214 -> r |
| (S215, \_) -> S216 | S215 -> \_ |
| (S217, t) -> S3 | S217 -> t |

Продолжение табл. 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S14, t) -> S15 | S14 -> t |
| (S15, r) -> S16 | S15 -> r |
| (S16, i) -> S17 | S16 -> i |
| (S17, n) -> S18 | S17 -> n |
| (S18, g) -> S3 | S18 -> g |
| (S14, w) -> S19 | S14 -> w |
| (S19, i) -> S20 | S19 -> i |
| (S20, t) -> S21 | S20 -> t |
| (S21, c) -> S22 | S21 -> c |
| (S22, h) -> S3 | S22 -> h |
| (S16, u) -> S63 | S16 -> u |
| (S63, c) -> S64 | S63 -> c |
| (S64, t) -> S3 | S64 -> t |
| (S15, a) -> S77 | S15 -> a |
| (S77, t) -> S78 | S77 -> t |
| (S78, i) -> S79 | S78 -> i |
| (S79, c) -> S3 | S79 -> c |
| (S14, h) -> S140 | S14 -> h |
| (S140, o) -> S141 | S140 -> o |
| (S141, r) -> S142 | S141 -> r |
| (S142, t) -> S3 | S142 -> t |
| (S14, i) -> S164 | S14 -> i |
| (S164, z) -> S165 | S164 -> z |
| (S165, e) -> S166 | S165 -> e |
| (S166, o) -> S167 | S166 -> o |
| (S167, f) -> S3 | S167 -> f |
| (S23, e) -> S24 | S23 -> e |

Продолжение табл. 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S24, t) -> S25 | S24 -> t |
| (S25, u) -> S26 | S25 -> u |
| (S26, r) -> S27 | S26 -> r |
| (S27, n) -> S3 | S27 -> n |
| (S31, h) -> S32 | S31 -> h |
| (S32, a) -> S33 | S32 -> a |
| (S33, r) -> S3 | S33 -> r |
| (S31, o) -> S55 | S31 -> o |
| (S55, n) -> S56 | S55 -> n |
| (S56, s) -> S57 | S56 -> s |
| (S57, t) -> S3 | S57 -> t |
| (S31, l) -> S58 | S31 -> l |
| (S58, a) -> S59 | S58 -> a |
| (S60, s) -> S61 | S60 -> s |
| (S61, s) -> S3 | S61 -> s |
| (S31, a) -> S101 | S31 -> a |
| (S101, t) -> S102 | S101 -> t |
| (S102, c) -> S103 | S102 -> c |
| (S103, h) -> S3 | S103 -> h |
| (S101, s) -> S107 | S101 -> s |
| (S107, e) -> S3 | S107 -> e |
| (S56, t) -> S114 | S56 -> t |
| (S114, i) -> S115 | S114 -> i |
| (S115, n) -> S116 | S115 -> n |
| (S116, u) -> S117 | S116 -> u |
| (S117, e) -> S118 | S117 -> e |
| (S34, =) -> S35 | S34 -> = |

Продолжение табл. 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S34, >) -> S39 | S34 -> > |
| (S36, =) -> S35 | S36 -> = |
| (S36, <) -> S39 | S36 -> < |
| (S38, =) -> S35 | S38 -> = |
| (S37, =) -> S35 | S37 -> = |
| (S40, =) -> S39 | S40 -> = |
| (S40, +) -> S390 | S40 -> + |
| (S41, =) -> S39 | S41 -> = |
| (S41, -) -> S390 | S41 -> - |
| (S42, =) -> S39 | S42 -> = |
| (S43, =) -> S39 | S43 -> = |
| (S44, =) -> S39 | S44 -> = |
| (S45, A..z) -> S45 | S45 -> A..z |
| S45 -> A..z S45 |
| (S50, o) -> S51 | S50 -> o |
| S50 -> o S51 |
| (S51, u) -> S52 | S51 -> u |
| (S52, b) -> S53 | S52 -> b |
| (S53, l) -> S54 | S53 -> l |
| (S54, e) -> S3 | S54 -> e |
| (S50, e) -> S118 | S50 -> e |
| (S118, f) -> S119 | S118 -> f |
| (S119, a) -> S120 | S119 -> a |
| (S120, u) -> S121 | S120 -> u |
| (S121, l) -> S122 | S121 -> l |
| (S122, t) -> S3 | S122 -> t |
| (S118, e) -> S123 | S118 -> e |

Продолжение табл. 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S124, e) -> S125 | S124 -> e |
| (S125, t) -> S126 | S125 -> t |
| (S126, e) -> S3 | S126 -> e |
| (S65, s) -> S66 | S65 -> s |
| (S66, i) -> S67 | S66 -> i |
| (S67, n) -> S68 | S67 -> n |
| (S68, g) -> S3 | S68 -> g |
| (S65, n) -> S216 | S65 -> n |
| (S216, i) -> S217 | S216 -> i |
| (S217, o) -> S218 | S217 -> o |
| (S218, n) -> S3 | S218 -> n |
| (S69, a) -> S70 | S69 -> a |
| (S70, m) -> S71 | S70 -> m |
| (S71, e) -> S72 | S71 -> e |
| (S72, s) -> S73 | S72 -> s |
| (S73, p) -> S74 | S73 -> p |
| (S74, a) -> S75 | S74 -> a |
| (S75, c) -> S76 | S75 -> c |
| (S76, e) -> S3 | S76 -> e |
| (S69, e) -> S168 | S69 -> e |
| (S168, w) -> S3 | S168 -> w |
| (S69, u) -> S187 | S69 -> u |
| (S187, l) -> S188 | S187 -> l |
| (S188, l) -> S189 | S188 -> l |
| (S189, p) -> S190 | S189 -> p |
| (S190, t) -> S191 | S190 -> t |
| (S191, r) -> S3 | S191 -> r |

Продолжение табл. 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S69, o) -> S202 | S69 -> 0 |
| (S202, t) -> S3 | S202 -> t |
| (S202, e) -> S203 | S202 -> e |
| (S203, x) -> S204 | S203 -> x |
| (S204, c) -> S205 | S204 -> c |
| (S205, e) -> S206 | S205 -> e |
| (S206, p) -> S207 | S206 -> p |
| (S207, t) -> S3 | S207 -> t |
| (S80, u) -> S81 | S80 -> u |
| (S81, b) -> S82 | S81 -> b |
| (S82, l) -> S83 | S82 -> l |
| (S83, i) -> S84 | S83 -> i |
| (S84, c) -> S3 | S84 -> c |
| (S80, r) -> S85 | S80 -> r |
| (S85, i) -> S86 | S85 -> i |
| (S86, v) -> S87 | S86 -> v |
| (S87, a) -> S88 | S87 -> a |
| (S88, t) -> S89 | S88 -> t |
| (S89, e) -> S3 | S89 -> e |
| (S85, o) -> S90 | S85 -> o |
| (S90, t) -> S91 | S90 -> t |
| (S91, e) -> S92 | S91 -> e |
| (S92, c) -> S93 | S92 -> c |
| (S93, t) -> S94 | S93 -> t |
| (S94, e) -> S95 | S94 -> e |
| (S95, d) -> S3 | S95 -> d |
| (S96, u) -> S97 | S96 -> u |

Продолжение табл. 1

|  |  |
| --- | --- |
| 1 | 2 |
| (S97, t) -> S98 | S97 -> t |
| (S98, o) -> S3 | S98 -> o |
| (S96, n) -> S192 | S96 -> n |
| (S192, d) -> S3 | S192 -> d |
| (S99, r) -> S100 | S99 -> r |
| (S100, y) -> S3 | S100 -> y |
| (S99, h) -> S104 | S99 -> h |
| (S104, r) -> S105 | S104 -> r |
| (S105, o) -> S106 | S105 -> o |
| (S106, w) -> S3 | S106 -> w |
| (S100, u) -> S126 | S100 -> u |
| (S126, e) -> S3 | S126 -> e |
| (S99, e) -> S143 | S99 -> e |
| (S143, m) -> S144 | S143 -> m |
| (S144, p) -> S145 | S144 -> p |
| (S145, l) -> S146 | S145 -> l |
| (S146, a) -> S147 | S146 -> a |
| (S147, t) -> S148 | S147 -> t |
| (S148, e) -> S3 | S148 -> e |
| (S99, y) -> S156 | S99 -> y |
| (S156, p) -> S157 | S156 -> p |
| (S157, e) -> S158 | S157 -> e |
| (S158, d) -> S159 | S158 -> d |
| (S159, e) -> S160 | S159 -> e |
| (S160, f) -> S3 | S160 -> f |
| (S158, n) -> S161 | S158 -> n |
| (S161, a) -> S162 | S161 -> a |

Продолжение табл.1

|  |  |
| --- | --- |
| 1 | 2 |
| (S162, m) -> S163 | S162 -> m |
| (S163, e) -> S3 | S163 -> e |
| (S158, i) -> S210 | S158 -> i |
| (S210, d) -> S3 | S210 -> d |
| (S104, i) -> S169 | S104 -> i |
| (S169, s) -> S3 | S169 -> s |
| (S134, o) -> S135 | S134 -> o |
| (S135, t) -> S136 | S135 -> t |
| (S136, o) -> S3 | S136 -> o |
| (S137, o) -> S138 | S137 -> o |
| (S138, n) -> S139 | S138 -> n |
| (S139, g) -> S3 | S139 -> g |
| (S170, i) -> S171 | S170 -> i |
| (S171, r) -> S172 | S171 -> r |
| (S172, t) -> S173 | S172 -> t |
| (S173, u) -> S174 | S173 -> u |
| (S174, a) -> S175 | S174 -> a |
| (S175, l) -> S3 | S175 -> l |
| (S170, o) -> S176 | S170 -> o |
| (S176, i) -> S177 | S176 -> i |
| (S177, d) -> S3 | S177 -> d |
| (S149, r) -> S3 | S149 -> r |

Таблица переходов по конечному автомату (приложение 4)

1. Построение грамматики и дерева порождения

Контекстно-свободная грамматика — частный случай формальной грамматики (тип 2 по иерархии Хомского), у которой левые части всех продукций являются одиночными нетерминалами и не имеющими.

Смысл термина «контекстно-свободная» заключается в том, что есть возможность применить продукцию к нетерминалу, причём независимоот контекста этого нетерминала (в отличие от общего случая неограниченной грамматики

Хомского).

Язык, который может быть задан КС-грамматикой, называется контекстно-свободным языком или КС-языком.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.

КС-грамматика:

G = {N,T,P,S}

N- множество нетерминальных символов (S, NV, IF, SWITCH, FOR, TYPE, OPERATION, OPERAND, LOS, OS, CONDITION, CASE)

T – множество терминальных символов (id, const, if, for, switch, case, int, float, double, +,-,++;,--,==,!=,<=,>=,+=,-=,\*=,!,[,],{,],;,)

P – множество правил порождений

S – целевой символ грамматики

Правила:

S->NV; S| IF; S| SWITCH; S | FOR; S| OPERATION; S|{S}| NV;|IF|FOR| OPERATION;|break;

NV-> TYPE id;|TYPE id=id;|TYPE id=const;|TYPE id[id];

TYPE->int|double|float|string

IF-> if(CONDITION) S |if(CONDITION) S ELSE| if(CONDITION) S break;

ELSE-> else S

CONDITION-> OPERAND LOS OPERAND

OPERAND->id|const||id[id]|id[const]|id[OPERATION]|OPERAND.FUNCTION|FUNCTION

LOS-> >|>=|<|<=|!=|==

OPERATION->OPERAND OS OPERAND |OPERAND OS OPERATION| OS OPERAND| OPERAND OS

OS-> +|++;|-|--;|\*|=|+=

FOR-> for (NV; CONDITION; OPERATION) S

FUNCTION->length()|toupper(OPERAND)| toupper(OPERATION)|

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

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

if (i > 0) {

newStr += toupper(str[++i]);

}

}

else {

newStr += str[i];

}

for (int j = 0; j < vowels.length(); j++) {

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

vowelCount++;

break;

}

}

}

**S=>FOR=>for (NV; CONDITION; OPERATION) S=>**for (NV; CONDITION; OPERATION) **{S}=>** for (NV; CONDITION; OPERATION) **{IF; S}=>** for (NV; CONDITION; OPERATION){IF; **FOR**}**=>** for (NV; CONDITION; OPERATION){IF; **for (NV; CONDITION; OPERATION) S**}**=>** for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPERATION) **{S}**}=> for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPERATION) {**IF**}}=> for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPERATION) {**if (CONDITION) S** }}=> for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPERATION) {if (CONDITION) **{S}** }}=> for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPERATION) {if (CONDITION) **{OPERATION; S}** }}=> for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPERATION) {if (CONDITION) {OPERATION; **break;**}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPERATION) {if (CONDITION) {**OPERAND OS**; break;}}} => for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPERATION) {if (CONDITION) {OPERAND **++**; break;}}} => for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPERATION) {if (CONDITION) {**id1** ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPERATION) {if (**OPERAND LOS OPERAND**) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPERATION) {if (OPERAND LOS **id2[id3]**) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPERATION) {if (OPERAND **==** id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPERATION) {if (**id5[id4]** == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; **OPREAND OS**) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; OPREAND **++**) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; CONDITION; **id3** ++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; **OPERAND LOS OPERAND**; id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; OPERAND LOS **OPERAND.FUNCTION**; id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; OPERAND LOS OPERAND.**length()**; id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; OPERAND LOS **id2**.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; OPERAND **<** id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (NV; **id3** < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (**TYPE id3=const1**; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){IF;for (**int** id3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ **if(CONDITION) S ELSE**;for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) S **else S**;for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) S else **{S}**;for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) S else{**OPERATION**};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) S else{**OPERAND OS OPERAND**};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) S else{OPERAND OS **id5[id4]**};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) S else{OPERAND **+=** id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) S else{**id6** += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{S}** else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{IF}** else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{if(CONDITION) S}** else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(CONDITION) **{S}**} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(CONDITION) **{OPERATION}**} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(CONDITION) **{OPERAND OS OPERAND}**} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(CONDITION) **{**OPERAND OS **FUNCTION}**} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(CONDITION) **{**OPERAND OS **toupper(OPERATION)}**} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(CONDITION) **{**OPERAND OS toupper(**OS OPERAND**)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(CONDITION) **{**OPERAND OS toupper(OS **id5**)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(CONDITION) **{**OPERAND OS toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(CONDITION) **{**OPERAND **+=** toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(CONDITION) **{id6** += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(**OPERAND LOS OPERAND**) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(OPERAND LOS **const1**) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(OPERAND **<** const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(CONDITION) **{**if(**id4** < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(**OPERAND LOS OPERAND**) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(OPERAND LOS **const2**) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(OPERAND **==** const2) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERATION){ if(**id5[id4]** == const2) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; **OPERAND OS**){ if(id5[id4] == const2) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; OPERAND **++**){ if(id5[id4] == const2) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; CONDITION; **id4 ++**){ if(id5[id4] == const2) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; **OPERAND LOS OPERAND**; id4 **++**){ if(id5[id4] == const2) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; OPERAND LOS **OPERAND.FUNCTION**; id4 **++**){ if(id5[id4] == const2) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; OPERAND LOSOPERAND**.length()**; id4 **++**){ if(id5[id4] == const2) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; OPERAND LOS **id5.**length(); id4 **++**){ if(id5[id4] == const2) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV; OPERAND **<** id5**.**length(); id4 **++**){ if(id5[id4] == const2) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (NV **id4** **<** id5**.**length(); id4 **++**){ if(id5[id4] == const2) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (**TYPE id4=const1;** id4 **<** id5**.**length(); id4 **++**){ if(id5[id4] == const2) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}}=> for (**int** id4=const1; id4 **<** id5**.**length(); id4 **++**){ if(id5[id4] == const2) **{**if(id4 < const1) **{**id6 += toupper(**++** id5)}} else{id6 += id5[id4]};for (intid3=const1; id3 < id2.length(); id3++) {if (id5[id4] == id2[id3]) {id1 ++; break;}}

# 4. Логическое проектирование

1. Блок-схемы алгоритмов

С помощью блок-схем можно рассмотреть работу алгоритма программы в целом или в конкретные ее функции. Ниже представлена общая блок-схема алгоритма программы (рис. 6).

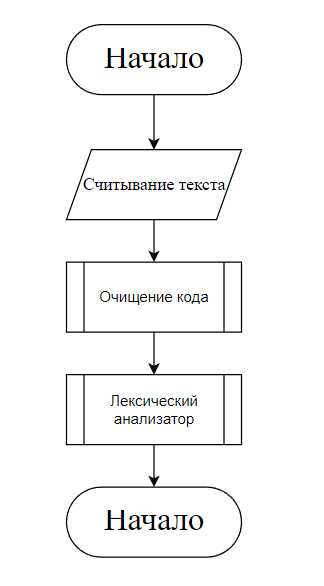


Рис.6. Общая блок-схема

Далее представлена блок-схема функции очистки кода от комментариев и лишних пробелов (рис. 7-8).

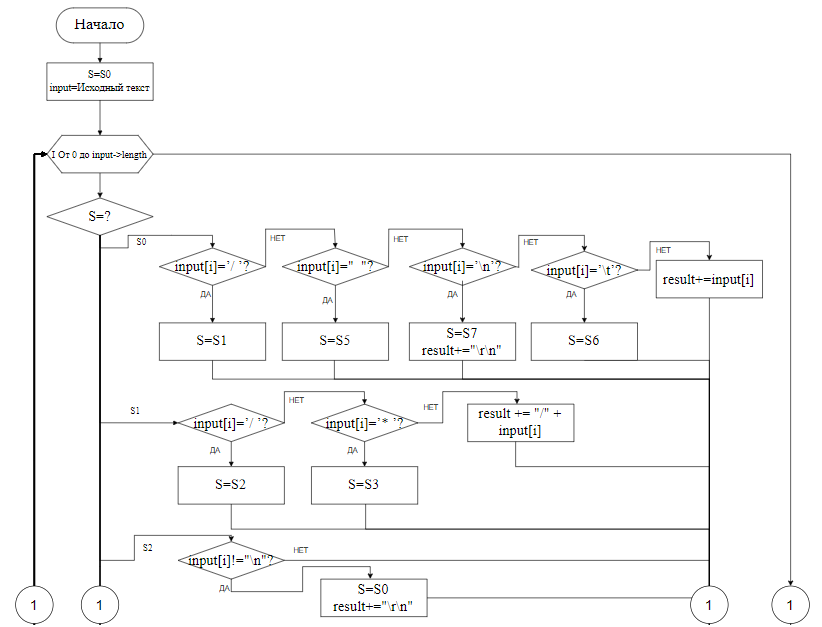


Рис.7. Блок-схема функции очищения кода. Часть 1

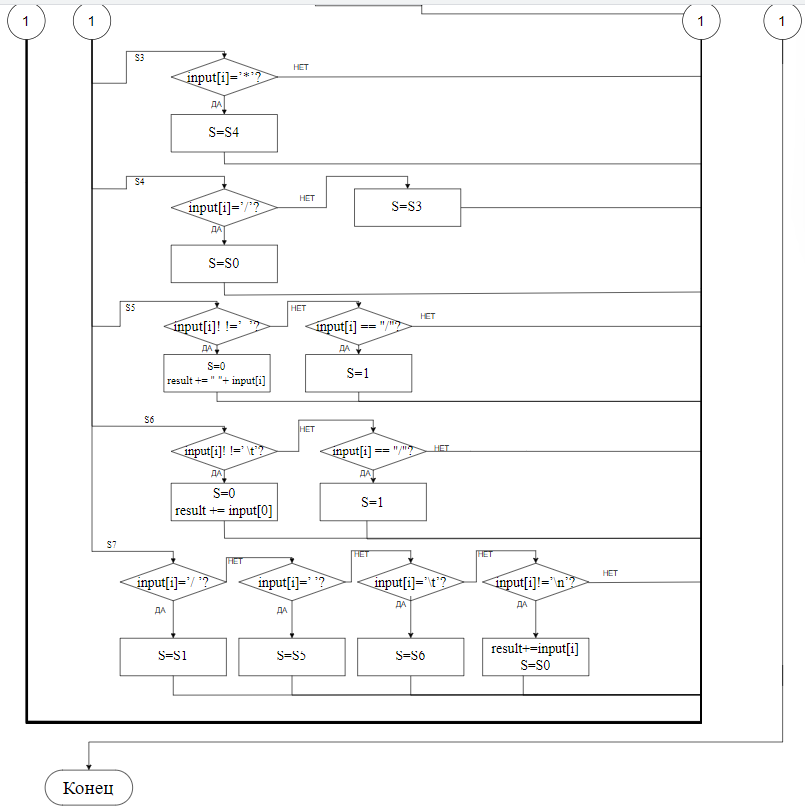


Рис.8. Блок-схема функции очищение кода. Часть 2

Далее представлена общая блок-схема лексического анализатора (рис.9).

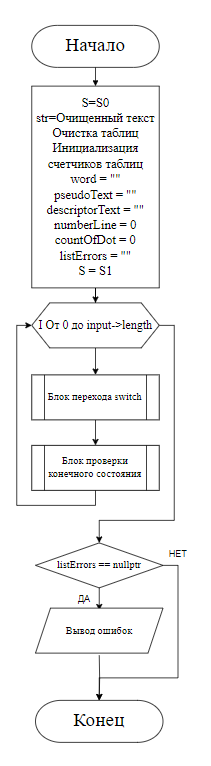


Рис.9. Общая блок-схема функции лексического анализатора

Далее представлена блок-схема switch(рис. 10-12) и блок-схема проверки конечного состояния (рис. 13-15).

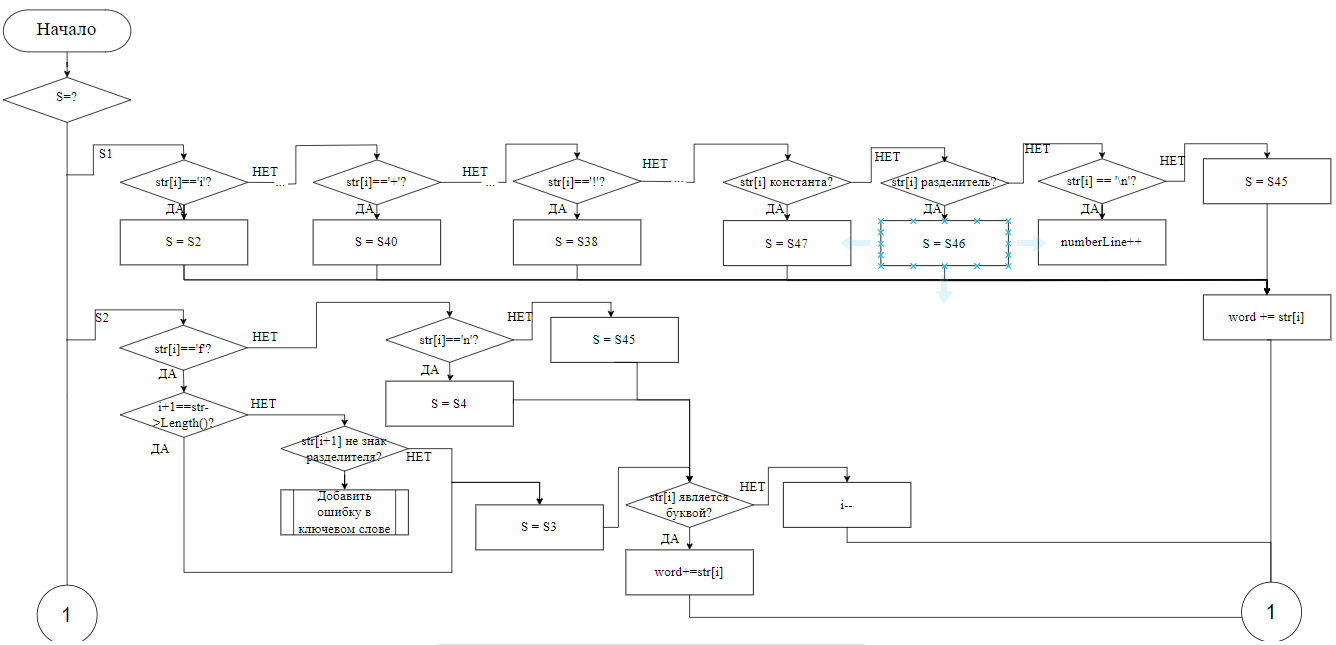


Рис.10. Блок-схема switch лексического анализатора. Часть 1

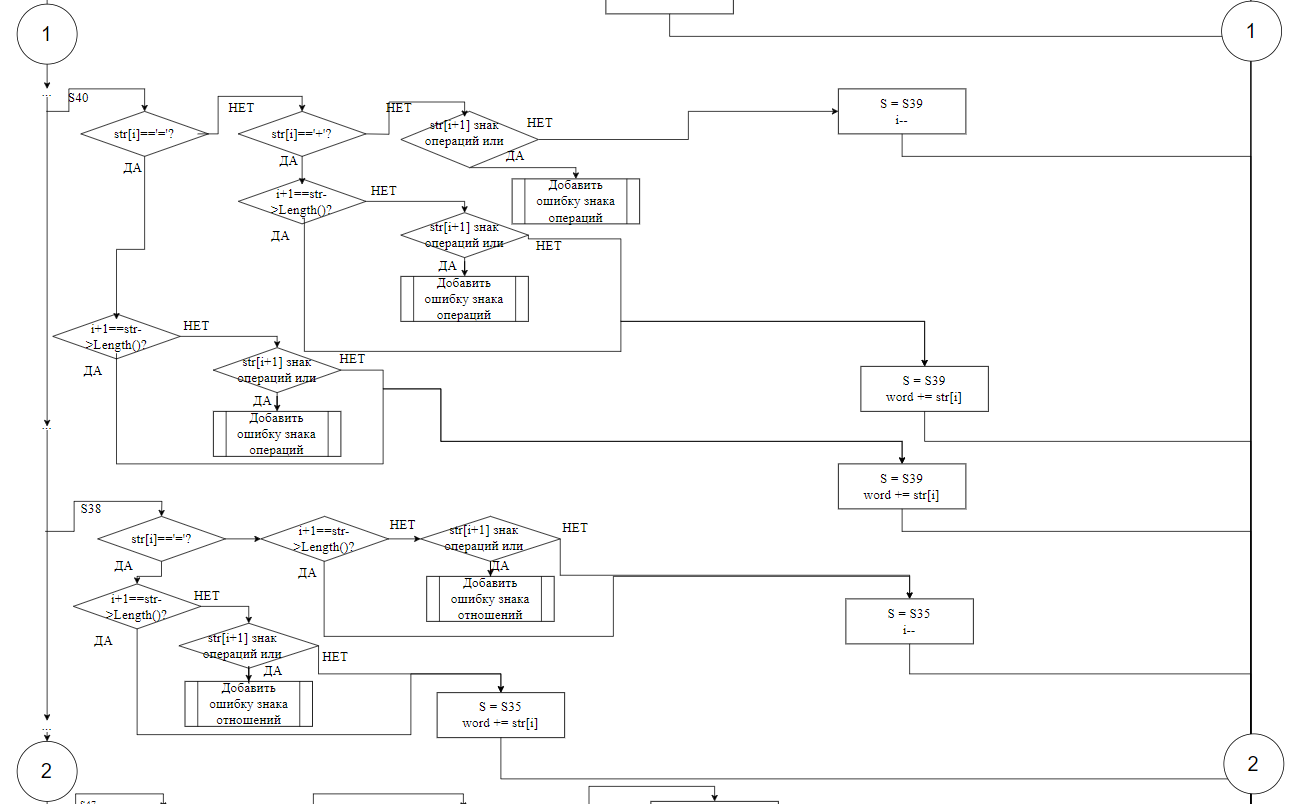


Рис. 11. Блок-схема switch лексического анализатора. Часть 2

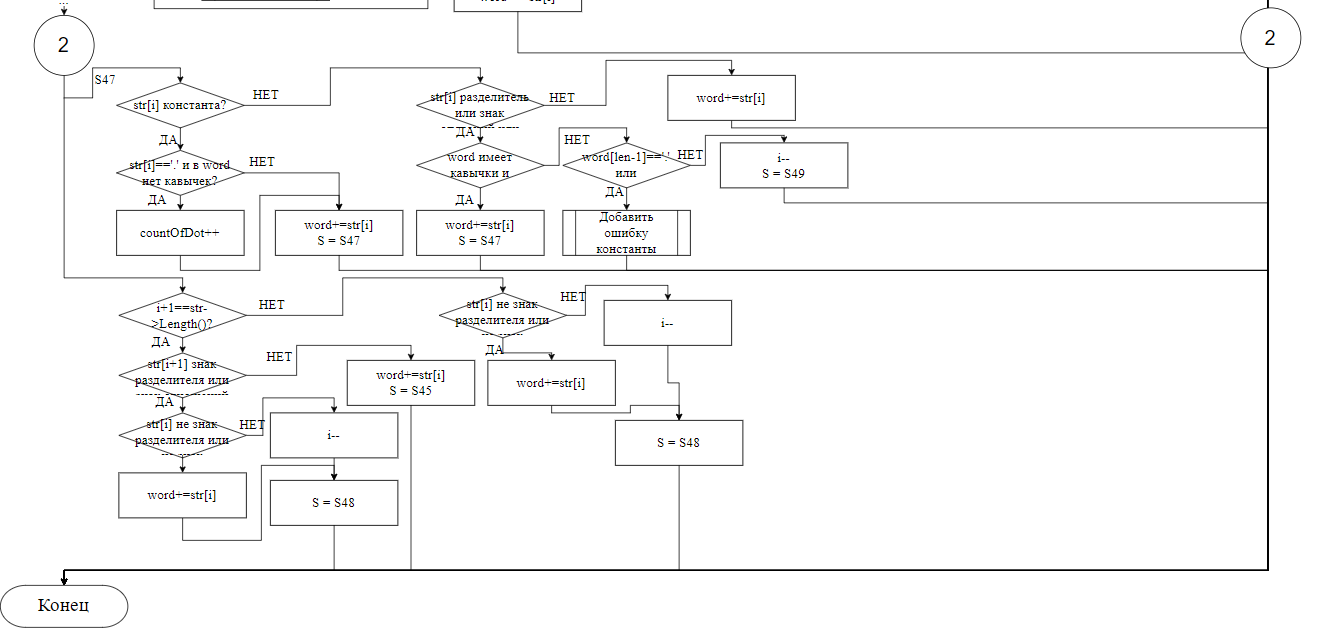


Рис. 12. Блок-схема switch лексического анализатора. Часть 3

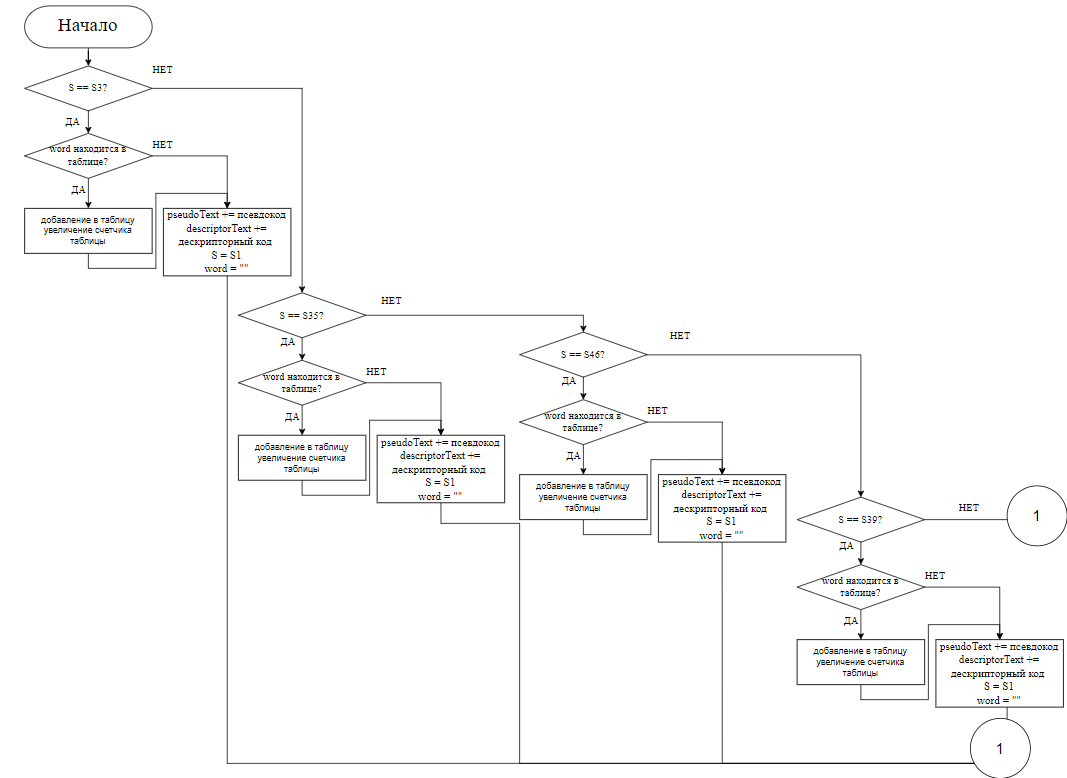


Рис. 12. Блок-схема проверки конечного состояния. Часть 1

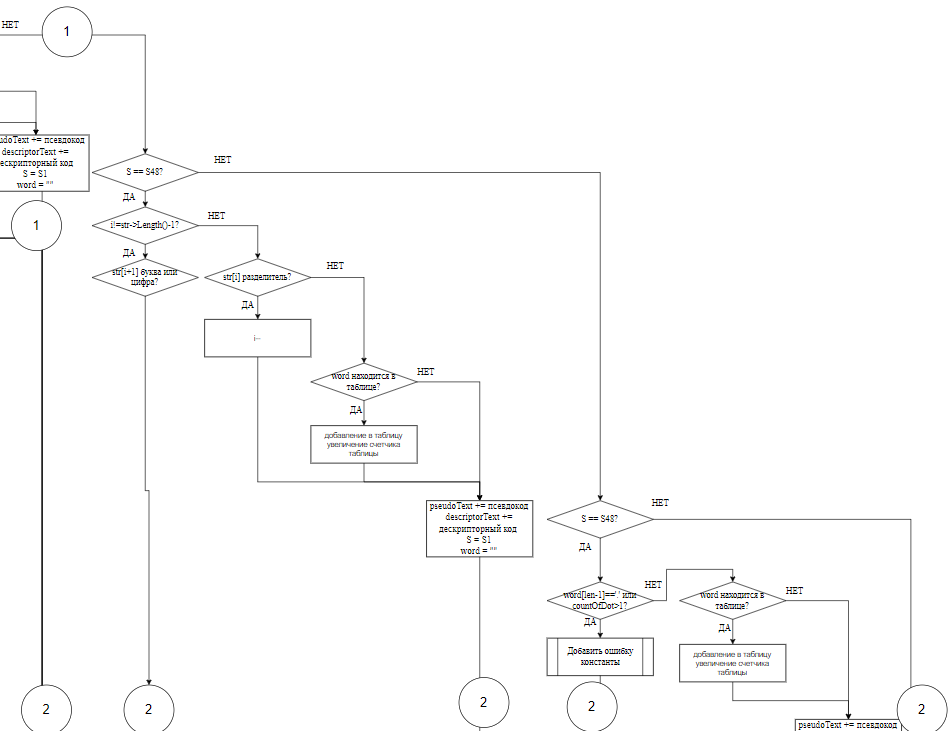


Рис. 13. Блок-схема проверки конечного состояния. Часть 2

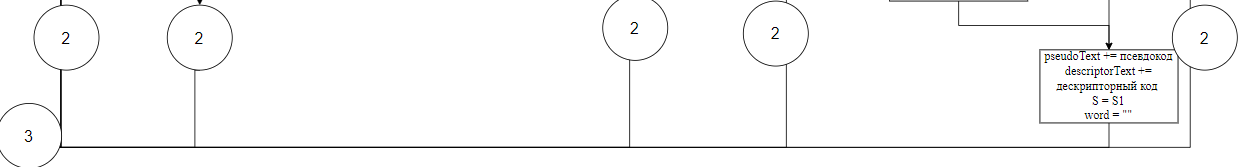


Рис. 14. Блок-схема проверки конечного состояния. Часть 3



Рис. 15. Блок-схема проверки конечного состояния. Часть 4

Далее представлена блок-схема добавления ошибки(рис. 16)

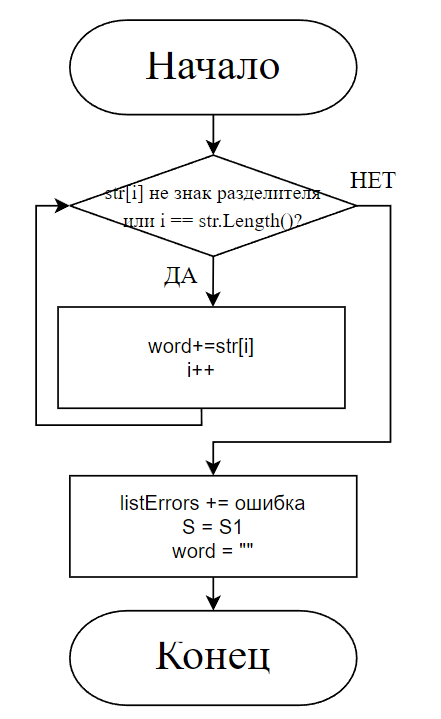


Рис. 16. Блок-схема добавления ошибки

1. Словесное описание алгоритмов

На вход программа получает строку символов, каждый из которых анализируется отдельно через цикл for, который продолжает работать до тех пор, пока не будет считан последний символ строки. От входного символа зависит дальнейшие действия программы, которые регулируются с помощью оператора switch. Программа реализует функцию удаления лишних пробелов, переносов строк и комментариев. Встретив символ, отвечающий за начало комментария, символы, следующие после не выводятся в поле «очищенный текст» до тех пор, пока не будет считан перенос строки, в случае однострочного комментария, или закрытие комментария в случае многострочного.

Для реализации функции лексического анализатора также используется оператор switch. Программа считывает символы до тех пор, пока не дочитает очищенный код до конца и если во время считывание нынешнее состояние становится конечным, то лексема добавляется в соответсвующую таблицу(которая определяется по конечному состоянию), если она до этого не была добавлена. А также в соответствующие поля выводится дескрипторный и псевдокод данной лексемы. Так программа может определять следующие лексемы: ключевые слова, идентификаторы, числовые константы, знаки операции и знаки отношений. При вводе нового текста автомат обнуляется до первого состояния, а так же очищаются все таблицы лексем.

Помимо определения лексем программа определяет ошибки во входном тексте, такие как ошибка в константе, в ключевом слове, в знаках операций и других символах. Ошибки выводятся в окне и не заносятся ни в таблицы лексем, ни в дескрипторный и псевдокод.

1. Оценка сложности алгоритм

Для определения сложности алгоритма необходимо проверить на сложность каждый этап вычислений (табл.2).

Таблица 2

Оценка сложности этапов

|  |  |  |
| --- | --- | --- |
| Название этапа | Описание этапа | Оценка сложности |
| Очистка кода от лишних пробелов, комментариев и пустых строк | Программа проходит каждый символ исходного текста единожды проходя по состояниям автомата. Если символ удовлетворяет условиям автомата, то он добавляется конец строки, где находиться очищенный код | O(n), где n – длина исходного кода |
| Лексический анализ очищенного текста | Программа проходит каждый символ 1 раз проходя по состояниям автомата. Если нашлась лексема или ошибка, она записывается в соответствующую таблицу/поле  Количество символов = длине строки очищенного кода | Запись нового значения в список занимает O(n), значит сложность данного этапа O(n), где n – длина очищенного кода |

Общая оценка сложности всей программы равна самой сложной оценки всех этапов. Так как длина исходного кода всегда либо равна, либо больше длины очищенного кода, тогда общая оценка сложности программы равна O(n), где n – длина строки исходного кода.

# 5. Физическое проектирование

1. Выбор структур данных

Таблица 3

Структуры данных

|  |  |  |
| --- | --- | --- |
| Назначение | Обозначение | Тип данных |
| Входная строка | input | String^ |
| Результат обработки | result | String^ |
| Номер строки | numberString | Int |
| Индекс по строке | i | int |
| Состояние очистки кода | currentLexerState | LexerState |
| Состояние чтения лексем | currentsKeyWordState | States |
| Собирающаяся автоматом лексема | word | String^ |
| Хранение информации об ошибках | listErrors | String^ |
| Номер строки | currentLineNumber | Int |
| Счетчик ключевых слов | keyWordIndex | Int |
| Счётчик знаков отношений | relationSignIndex | Int |
| Счетчик идентификаторов | idTableIndex | Int |
| Счётчик знаков операций | operationSignIndex | Int |
| Счетчик констант | constTableIndex | Int |

1. Спецификация функция

Таблица 4

Спецификации функции

|  |  |  |  |
| --- | --- | --- | --- |
| Назначение | Обозначение | Тип | Параметры |
| 1 | 2 | 3 | 4 |
| Лексического анализатора | ReadLexem | void |  |
| Очистка кода от комментариев, лишних пропусков и пустых строк | CleanCode | int |  |
| Проверка символа на разделитель | isSplitterForTable | bool | char ch |
| Проверка символа на знак операций | isOperationSign | bool | char ch |
| Проверка символа на знак отношений | isRelationSigns | bool | char ch |
| Проверка символа на констант | isConstant | bool | char ch |
| Проверка, есть уже лексема в таблице | isContainInTable | bool | String^ word, DataGridView^ table |
| Получение псевдокода лексемы | getPseudo | void | String^s, DataGridView^ table |
| Получение дескрипторного кода лексемы | getDescriptor | void | String^ s, DataGridView^ table |
| Запись лексемы в таблицу | WriteInTable | void | DataGridView^ wordsTable, String^ word, int wordIndex |
| Запись идентификатора в таблицу | WriteInIdTable | void | DataGridView^ wordsTable, String^ word, int wordIndex |
| Запись константы в таблицу | WriteInConstTable | void | DataGridView^ wordsTable, String^ word, int wordIndex |

# Продолжение табл. 4

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Добавление ошибки | addError | array<String^>^ | String^ listErrors, String^ word, String^ str, int i, int numberLine, String^ errorMessage |

# 6. Проектирование интерфейса

Интерфейс — это набор инструментов, который позволяет пользователю взаимодействовать с программой. В более широком смысле термин обозначает любые инструменты для соприкосновения между разными системами и сущностями. Часто говорят о графическом интерфейсе — это внешний вид сайта, программы или приложения.

**Графический пользовательский интерфейс** (GUI) – разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.), представленные пользователю на дисплее. Именно этот тип интерфейса разрабатывался в ходе работы(рис. ).

Интерфейс приложения (рис. 17) состоит из нескольких текстовых полей – «input», «result», «PseudoCode», «DescriptorCode», также на экране расположено несколько кнопок для взаимодействия – «Start», «add standart code». Помимо этого, на экране располагаются таблицы, предназначенные для заполнения их лексемами из исходного кода – «KeyWord», «Constant», «Operation Signs», «Relations Signs», «Splitter» «Id».

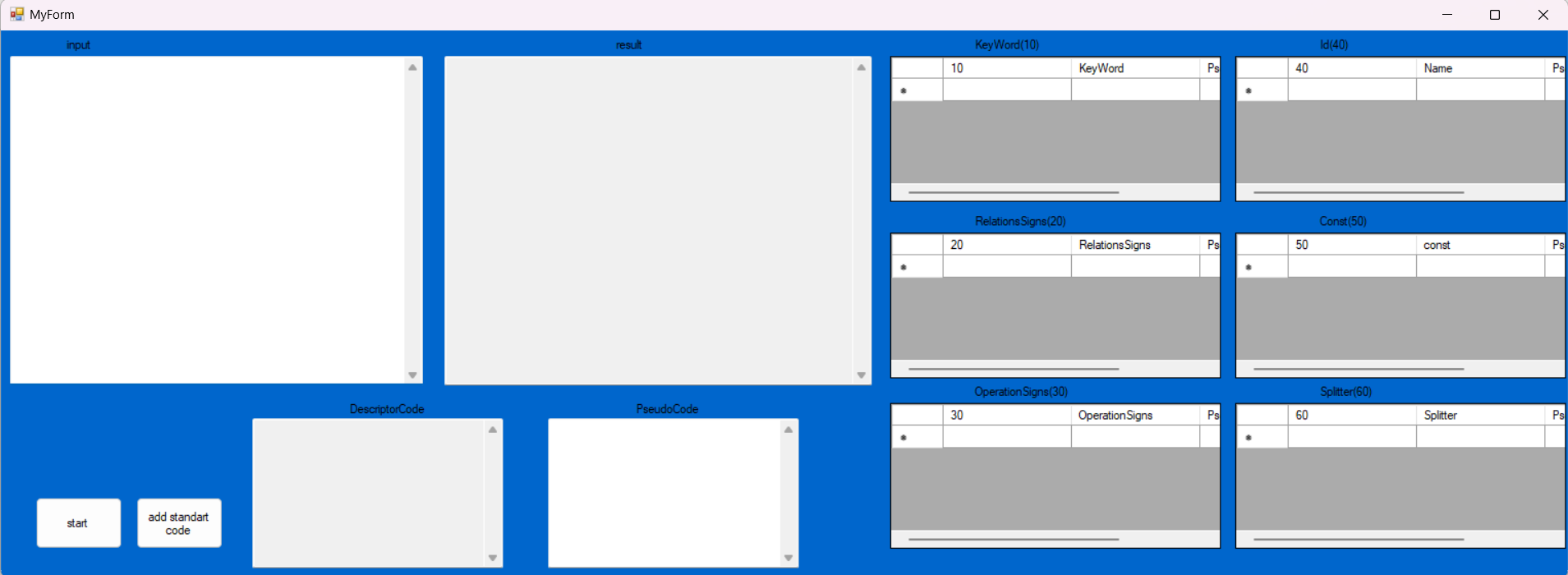


Рис.17. Интерфейс программы

# 7. Тестирование программного обеспечения

Во время тестирования была проанализирована работа программы при различных вариантах входных данных.

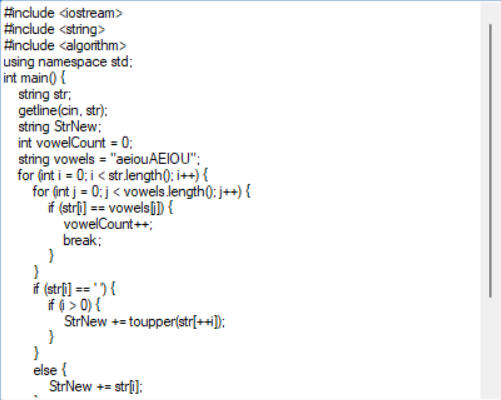


Рис.18. Вставка стандартного кода

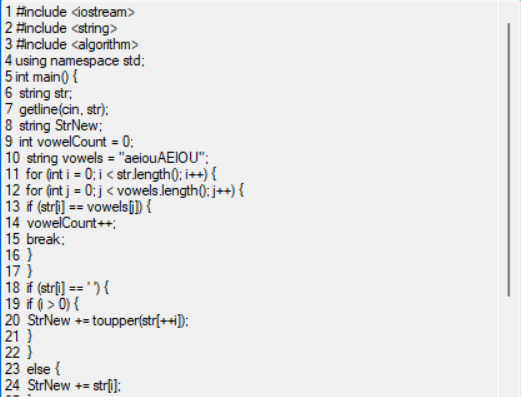


Рис.19. Удаление пробелов и комментариев

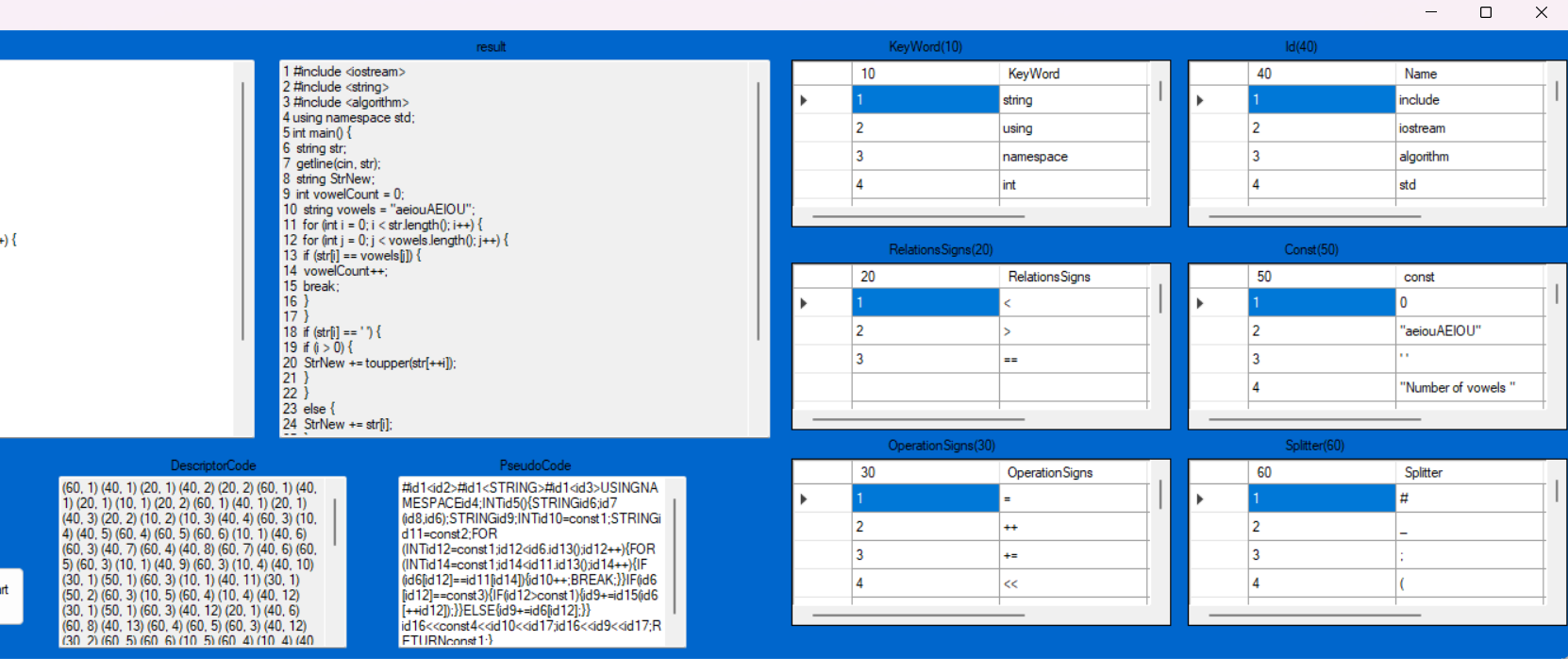


Рис. 20. Лексический анализ

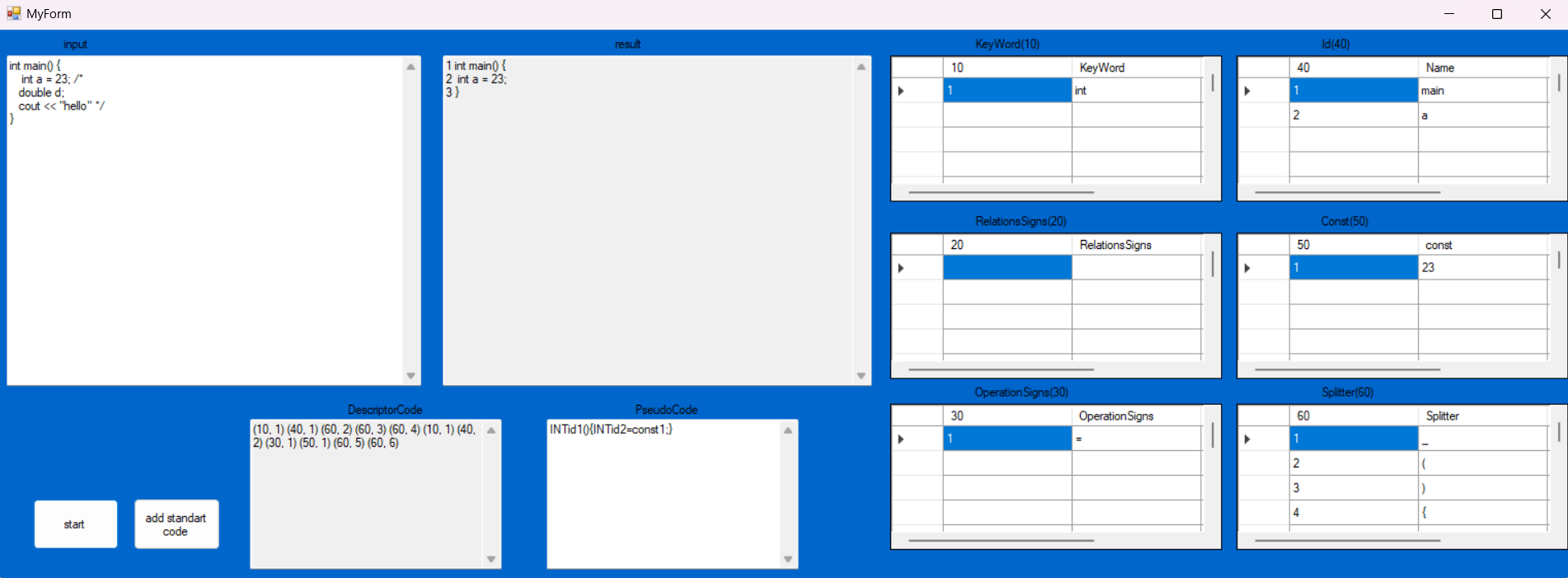


Рис. 21. Проверка удаления многострочного комментария

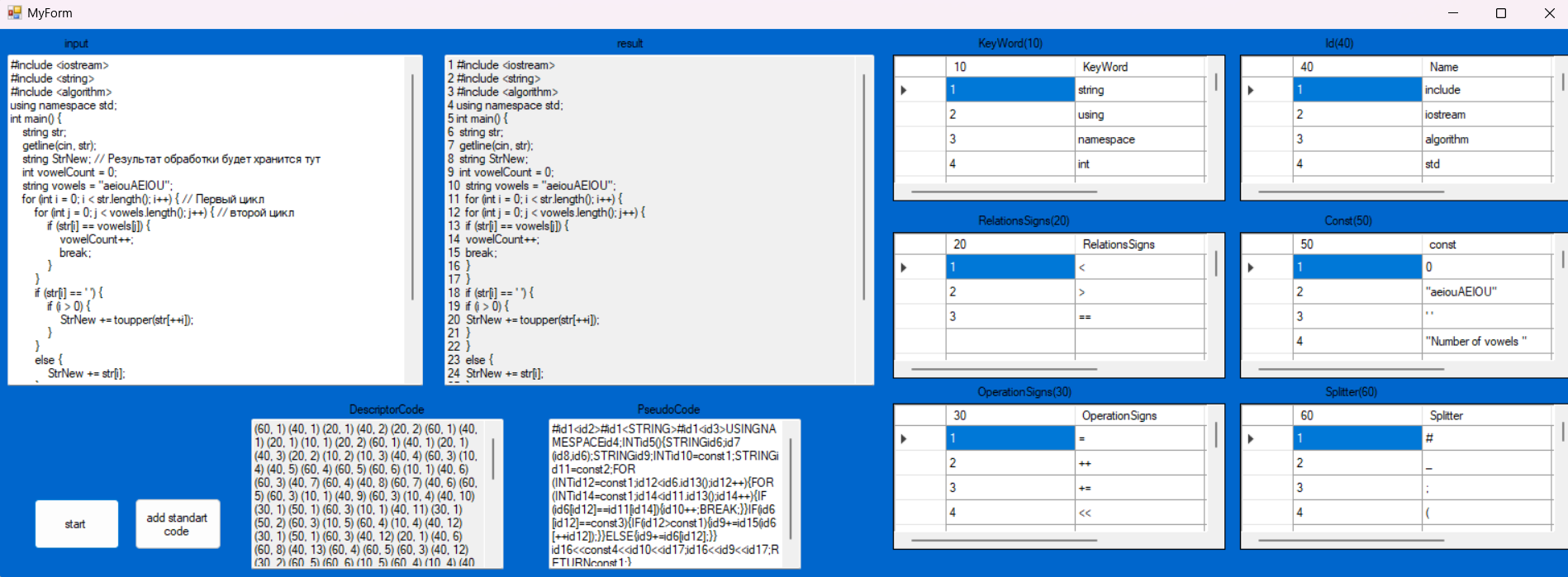


Рис. 22. Проверка удаления комментария

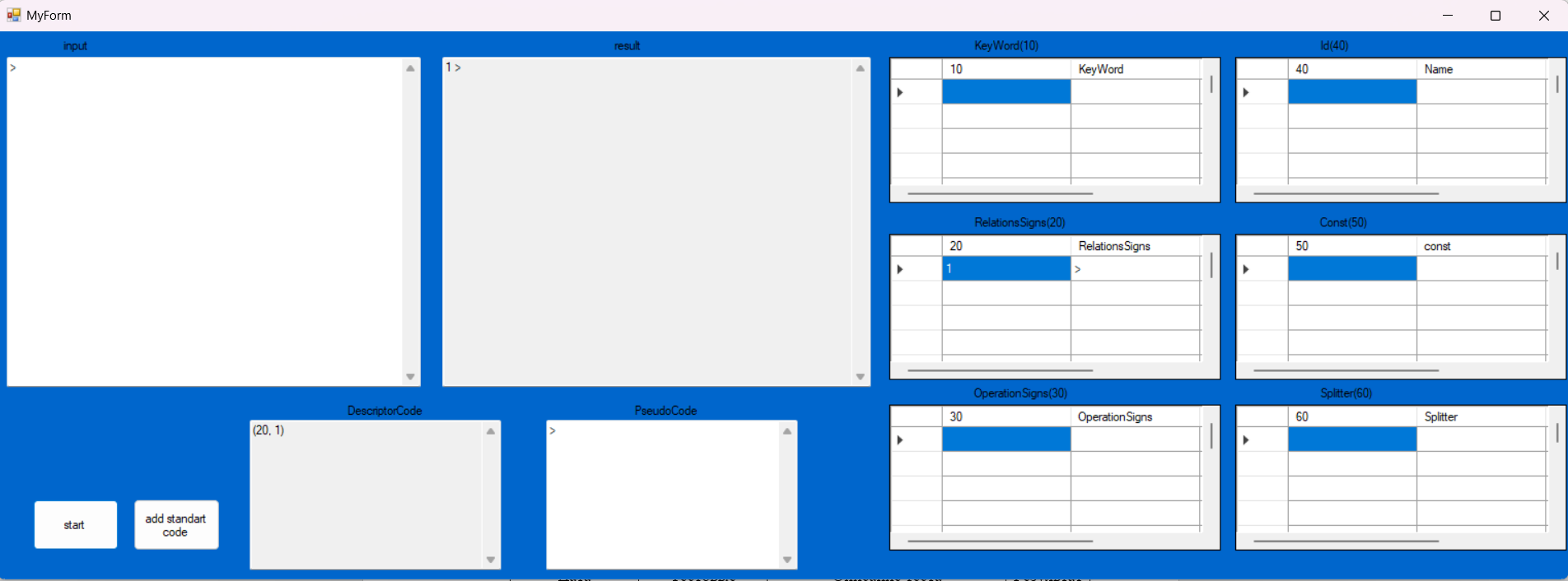


Рис. 23. Проверка чтения одной лексемы

Таблица 6

Тестирование данных

|  |  |  |  |
| --- | --- | --- | --- |
| Дата тестирования | Тестовые данные | Описание теста | Результат |
| 13.04.2024 | Тестовый код | Добавление тестового кода | Успех |
| 21.04.2024 | Тестовый код | Очищение считанного кода от лишних пробелов и комментариев | Успех |
| 11.05.2024 | Тестовый код | Определение ключевых слов | Успех |
| 13.05.2024 | Тестовый код | Определение констант с плавающей точкой | Успех |
| 20.05.2024 | Тестовый код | Определение идентификаторов | Успех |
| 25.05.2024 | Тестовый код | Определение знаков операций из кода | Успех |
| 30.05.2024 | Тестовый код | Определение прочих символов | Успех |
| 31.05.2024 | Тестовый код | Определение ошибок | Успех |

Таблица 7

Тестирование программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата тестирования | Модуль тестирования | Кто проводил тестирование | Описание теста | Результаты тестирования |
| 1 | 2 | 3 | 4 | 5 |
| 03.06.2024 | Lexer.exe | Симаньков А.Е | Тест функции очищения кода | Успех |
| 03.06.2024 | Lexer.exe | Зернов В.А | Тестирование ввода текста с клавиатуры | Успех |
| 04.06.2024 | Lexer.exe | Овичиников М.В | Тестирование функции лексического анализа | Ошибка при выявлении ключевых слов |
| 05.06.2024 | Lexer.exe | Гончаров Е.Д | Выявление ключевых слов | Ошибка в считывании по строкам |

Продолжение табл. 7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 07.06.2024 | Lexer.exe | Зернов В.А | Лексический анализатор | Успех |

# 8. Заключение.

В ходе выполнения данной курсовой работы были освоены знания и навыки работы с автоматами и лексическим анализатором. Было создано программное обеспечение выполняющая работу лексического анализатора на языке программирования C++ в соответствии с заданным вариантом(рис.2). Был смоделирован лексический анализатор компилятора на языке высокого уровня С++ с использованием Windows Forms. Данное приложение способно проводить обработку вводимого текста программы, удалять лишние пробелы, знаки табуляции, комментарии, излишние переходы на новую строку, выполнять подсчет количества строк в программе и выводить таблицу классов лексем, дескрипторный код и псевдокод.

Для программы созданы алгоритмы в виде блок-схем и словесной форме, подсчитана сложность алгоритма. Помимо этого, была разработана регулярная и КС-грамматики, построено дерево вывода и разобрано правостороннее порождение. Были построены таблицы переходов автомата и регулярная грамматика по нему.

Были получены знания в области построения автоматов, на основе которых строятся лексические анализаторы, компиляторы.

# 9. 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. унт", Ин-т информ. технологий. - Череповец : ГОУ ВПО "Череповецкий гос. унт", 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. Проектирование лексического анализатора для интерактивного конструктора задачи многокритериальной оптимизации Охотниченко А.В., Сибилева Н.С. Математическое и программное обеспечение систем в промышленной и социальной сферах. 2018. Т. 6. № 1. С. 40.
5. Сайт Microsoft с документация по языку программирования C# [эл. ресурс] https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=msvc-170

Приложение 1. Техническое задание

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное

образовательное учреждение высшего образования

ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт информационных технологий

Математического и программного обеспечения ЭВМ

Модуль «Информатика»

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ

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

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Разработка алгоритмического обеспечения и построение

лексического анализатора компилятора

Техническое задание на курсовую работу

Листов 8

Руководители: доцент Ганичева О. Г., Пышницкий К.М.

Исполнитель: студент гр. 1ПИб-02-2оп-22

Зернов В.А

2024 год

Введение

Курсовая работа ставит целью разработку алгоритмического обеспечения и построения лексического анализатора, которое позволит пользователю наглядно увидеть лексемы в коде и понять часть принципа работы компилятора.

Данный проект предназначен для формирования знаний и навыков по дисциплине Модуль «Информатика».

1. Основания для разработки:

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

Дата утверждения: 13 февраля 2024 года.

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

1. Назначение разработки

Учебная программа должна моделировать работу лексического анализатора, которая решает следующие задачи: выделить все лексемы языка C++, знаки операций, построить КС-грамматику, удалять лишние пробелы, отступы, переходы на следующие строки, однострочные и многострочные комментарии, поддерживать строки, подсчитывать количество строк в программе и выводить ошибки при наличии некорректных лексем.

1. Требования к программе

3.1. Требования к функциональным характеристикам

3.1.1. Модель лексического анализатора должна обрабатывать текст из выбранных пользователем файлов в графическом интерфейсе, подготовить текст к последующей обработке путем удаления лишних пробелов, отступов, однострочных и многострочных комментариев, излишних символов новой строки, подсчитывать строки кода.

3.1.2. Программа должна находить ключевые слова и знаки отношений, записывать их в таблицу без повторений.

3.1.3. Программа должна формировать таблицы с классами лексем, найденными во входном коде.

3.1.4. Программа должна реализовывать построение дескрипторного кода и псевдокода.

3.1.5. Программа должна обнаруживать ошибки, указывать номер строки и идентифицировать ошибку. Также программа должна построить КС-грамматику для заданного в курсовой работе варианта задания.

3.2 Требования к надежности

Для правильного функционирования программы необходимо:

1. Входной файл должен иметь текстовый формат.
2. Кодировка символов исходного файла должна быть ASCII или совместимая с ней и содержать только ASCII символы.
3. Вводимые лексемы должны удовлетворять возможным лексемам, которые определены с помощью состояний конечного автомата.
   1. Условия эксплуатации

Программа должна быть совместима с операционной системой пользователя. Условия эксплуатации программы зависят от условий эксплуатации персонального компьютера.

3.4 Требования к составу и параметрам технических средств

* Тактовая частота процессора более 1 GHz;
* Количество оперативной памяти 1Гб или более;
* Свободное место на системном накопителе от 512 Мб и более;
* Операционная система Windows 7 и выше;
* Процессорная архитектура x86 или x64;
* Монитор;
* Клавиатура;
* Мышь.

3.5 Требования к информационной и программной совместимости

Проект написан на языке C++ с использованием среды разработки Visual Studio 2022. Для редактирования или сборки программы необходимы инструменты версии v142 и пакет SDK для Windows 10.0. Если требуется поддержка более ранних версий Windows, необходимо настроить проект и установить соответствующий SDK для нужной версии Windows.

3.6 Требования к маркировке и упаковке

Распространяется передачей исходных файлов проекта как на физических носителях, так и через другие средства передачи. Также может передаваться уже собранная версия программы третьим лицам. Авторское право сохраняется за автором исходного кода до явной передачи его третьим лицам.

3.7 Требования к транспортировке и хранению

Транспортировку и хранение рекомендуется осуществлять на оптическом носителе DVD-диске. При соблюдении условий хранения выбранного носителя срок хранения ограничивается сроком хранения хотя бы одной копии программного обеспечения на каком-либо носителе.

3.8 Специальные требования

Наличие минимальных познаний о языке программирования C++, знания некоторых теоретических сведений о работе и устройстве компиляторов, лексических анализаторов.

4. Требования к программной документации

4.1 Программная документация должна содержать расчетно-пояснительную записку с содержанием:

* Титульный лист
* Оглавление
* Введение
  + - * + Теоретическая часть
        + Практическая часть
* Заключение
* Список литературы
* Приложения.

4.2. Требования к оформлению:

* Документ;
* Печать на отдельных листах формата А4 (210х297 мм); оборотная сторона не заполняется; листы нумеруются. Печать возможна ч/б;
* Файлы предъявляются на компакт-диске: РПЗ с ТЗ; программный код;
* Листы и диск в конверте вложены в пластиковую папку скоросшивателя;
* Страницы;
* Ориентация – книжная; отдельные страницы, при необходимости, альбомная;
* Поля: верхнее, нижнее – по 2 см, левое – 3 см, правое – 1 см;
* Абзацы;
* Межстрочный интервал – 1,5, перед и после абзаца – 0;
* Шрифты;
* Кегль – 14. В таблицах шрифт 12. Шрифт листинга – 10;
* Рисунки;
* Подписывается под ним по центру: Рис. Х. Название;
* В приложениях: Рис.П1.3. Название;
* Таблицы;
* Подписывается: над таблицей, выравнивание по правому: «Таблица Х»;
* В следующей строке по центру Название;
* Надписи в шапке (имена столбцов, полей) – по центру;
* В теле таблицы (записи) текстовые значения – выровнены по левому, числа, даты – по правому.

5. Стадии и этапы разработки

В данном пункте представлены этапы разработки, которые будут выполняться при разработке программы, в соответствии с табл. П1.1.

Таблица П1.1

Этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапа разработки | Сроки разработки | Результат выполнения | Отметка о выполнении |
| 1 | 2 | 3 | 4 |
| Сбор необходимой информации для выполнения данной курсовой работы | 12.03.2024 | Овладение необходимыми знаниями для написания кода программы |  |
| Анализ задачи и требований к проекту | 06.04.2024 | Выявление и составление четкого алгоритма программы |  |
| Реализация алгоритма по обработке кода | 15.03.2024 | Получение технически правильного кода программы |  |
| Написание функции для нахождения ключевых слов и знаков отношений | 08.03.2024 | Получение технически правильного кода программы |  |
| Написание функции для нахождения ключевых слов и всех знаков операций | 10.03.2024 | Получение технически правильного кода программы |  |
| Реализация построения дескрипторного кода и псевдокода | 13.03.2024 | Получение технически правильного кода программы |  |

Продолжение таблица П1.1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Написание функции по обнаружению ошибок и построению КС-грамматики | 18.03.2024 | Получение технически правильного кода программы |  |
| Тестирование | 25.05.2024 | Исправление ошибок и сравнение результатов работы программы и самостоятельных подсчетов |  |
| Разработка РПЗ | 18.03.2024-25.05.2024 | Разработанное РПЗ |  |

1. Порядок контроля и приемки

Контроль выполнения работы осуществляется преподавателем. В случае обнаружения недостатков, исполнитель работы обязан исправить их в срок, согласованный с преподавателем. Контроль разработки продукта осуществляется поэтапно, в соответствии с графиком, представленным в табл. П2.

Таблица П1.2

Порядок контроля и приемки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование контрольного этапа выполнения курсовой работы | Сроки контроля | Результат выполнения | Отметка о приемке результата контрольного этапа |
| 1 | 2 | 3 | 4 |
| Сдача технического задания | 10.04.2024 | Документ “Техническое задание” проверен |  |
| Демонстрация работы программы | 27.05.2024 | Работа программы проверена |  |

Продолжение табл. П1.2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Оформление РПЗ | 29.05.2024 | Оформление РПЗ принято |  |
| Сдача РПЗ |  | Оценка |  |

Приложение 2.

Текст программы

Файл MyForm.cpp

#include "MyForm.h"

#include <iostream>

using namespace System;

using namespace System::Windows::Forms;

[STAThreadAttribute]

int main(array<String^>^ args)

{

Application::SetCompatibleTextRenderingDefault(false);

Application::EnableVisualStyles();

Lexer::MyForm MyForm;

Application::Run(% MyForm);

}

Файл MyForm.h

#pragma once

#include "LexerStates.cpp"

#include "States.cpp"

#include <set>

#include <vector>

namespace Lexer {

using namespace System;

using namespace System::Text;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// ������ ��� MyForm

/// </summary>

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

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: �������� ��� ������������

//

}

protected:

/// <summary>

/// ���������� ��� ������������ �������.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::TextBox^ input;

protected:

private: System::Windows::Forms::TextBox^ result;

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

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

private: System::Windows::Forms::Label^ label2;

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

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::Label^ label4;

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

private: System::Windows::Forms::Label^ label5;

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

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

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

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

private: System::Windows::Forms::Label^ label7;

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

private: System::Windows::Forms::Label^ label8;

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

private: System::Windows::Forms::TextBox^ descriptorCode;

private: System::Windows::Forms::TextBox^ pseudoCodes;

private: System::Windows::Forms::Label^ label9;

private: System::Windows::Forms::Label^ label10;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ keyWord10;

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

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

private: System::Windows::Forms::DataGridViewTextBoxColumn^ relSigns20;

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

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

private: System::Windows::Forms::DataGridViewTextBoxColumn^ oper30;

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

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

private: System::Windows::Forms::DataGridViewTextBoxColumn^ split60;

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

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

private: System::Windows::Forms::DataGridViewTextBoxColumn^ const50;

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

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

private: System::Windows::Forms::DataGridViewTextBoxColumn^ id40;

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

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn13;

private:

/// <summary>

/// ������������ ���������� ������������.

/// </summary>

System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code

/// <summary>

/// ��������� ����� ��� ��������� ������������ � �� ���������

/// ���������� ����� ������ � ������� ��������� ����.

/// </summary>

void InitializeComponent(void)

{

this->input = (gcnew System::Windows::Forms::TextBox());

this->result = (gcnew System::Windows::Forms::TextBox());

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

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

this->label2 = (gcnew System::Windows::Forms::Label());

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

this->label3 = (gcnew System::Windows::Forms::Label());

this->label4 = (gcnew System::Windows::Forms::Label());

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

this->label5 = (gcnew System::Windows::Forms::Label());

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

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

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

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

this->label7 = (gcnew System::Windows::Forms::Label());

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

this->label8 = (gcnew System::Windows::Forms::Label());

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

this->descriptorCode = (gcnew System::Windows::Forms::TextBox());

this->pseudoCodes = (gcnew System::Windows::Forms::TextBox());

this->label9 = (gcnew System::Windows::Forms::Label());

this->label10 = (gcnew System::Windows::Forms::Label());

this->relSigns20 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

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

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

this->keyWord10 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

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

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

this->id40 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

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

this->dataGridViewTextBoxColumn13 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->const50 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

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

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

this->oper30 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

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

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

this->split60 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

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

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

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

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

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

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

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

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

this->SuspendLayout();

//

// input

//

this->input->Location = System::Drawing::Point(12, 31);

this->input->Margin = System::Windows::Forms::Padding(3, 2, 3, 2);

this->input->Multiline = true;

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

this->input->ScrollBars = System::Windows::Forms::ScrollBars::Both;

this->input->Size = System::Drawing::Size(535, 394);

this->input->TabIndex = 0;

//

// result

//

this->result->Location = System::Drawing::Point(576, 31);

this->result->Margin = System::Windows::Forms::Padding(3, 2, 3, 2);

this->result->Multiline = true;

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

this->result->ReadOnly = true;

this->result->ScrollBars = System::Windows::Forms::ScrollBars::Both;

this->result->Size = System::Drawing::Size(553, 394);

this->result->TabIndex = 1;

//

// startProcessing

//

this->startProcessing->Location = System::Drawing::Point(45, 560);

this->startProcessing->Margin = System::Windows::Forms::Padding(3, 2, 3, 2);

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

this->startProcessing->Size = System::Drawing::Size(112, 62);

this->startProcessing->TabIndex = 2;

this->startProcessing->Text = L"start";

this->startProcessing->UseVisualStyleBackColor = true;

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

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(83, 9);

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

this->label1->Size = System::Drawing::Size(35, 16);

this->label1->TabIndex = 3;

this->label1->Text = L"input";

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(796, 9);

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

this->label2->Size = System::Drawing::Size(39, 16);

this->label2->TabIndex = 4;

this->label2->Text = L"result";

//

// keyWordsTable

//

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

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

this->keyWord10,

this->KeyWord, this->Pseudocode

});

this->keyWordsTable->Location = System::Drawing::Point(1155, 31);

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

this->keyWordsTable->RowHeadersWidth = 51;

this->keyWordsTable->RowTemplate->Height = 24;

this->keyWordsTable->Size = System::Drawing::Size(430, 175);

this->keyWordsTable->TabIndex = 5;

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(1262, 9);

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

this->label3->Size = System::Drawing::Size(85, 16);

this->label3->TabIndex = 6;

this->label3->Text = L"KeyWord(10)";

//

// label4

//

this->label4->AutoSize = true;

this->label4->Location = System::Drawing::Point(1262, 220);

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

this->label4->Size = System::Drawing::Size(120, 16);

this->label4->TabIndex = 8;

this->label4->Text = L"RelationsSigns(20)";

//

// relationsSignsTable

//

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

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

this->relSigns20,

this->RelationsSigns, this->dataGridViewTextBoxColumn3

});

this->relationsSignsTable->Location = System::Drawing::Point(1155, 242);

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

this->relationsSignsTable->RowHeadersWidth = 51;

this->relationsSignsTable->RowTemplate->Height = 24;

this->relationsSignsTable->Size = System::Drawing::Size(430, 175);

this->relationsSignsTable->TabIndex = 7;

//

// label5

//

this->label5->AutoSize = true;

this->label5->Location = System::Drawing::Point(1262, 425);

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

this->label5->Size = System::Drawing::Size(122, 16);

this->label5->TabIndex = 10;

this->label5->Text = L"OperationSigns(30)";

//

// operationSignsTable

//

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

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

this->oper30,

this->OperationSigns, this->dataGridViewTextBoxColumn5

});

this->operationSignsTable->Location = System::Drawing::Point(1155, 447);

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

this->operationSignsTable->RowHeadersWidth = 51;

this->operationSignsTable->RowTemplate->Height = 24;

this->operationSignsTable->Size = System::Drawing::Size(430, 175);

this->operationSignsTable->TabIndex = 9;

//

// standartCode

//

this->standartCode->Location = System::Drawing::Point(176, 560);

this->standartCode->Margin = System::Windows::Forms::Padding(3, 2, 3, 2);

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

this->standartCode->Size = System::Drawing::Size(112, 62);

this->standartCode->TabIndex = 11;

this->standartCode->Text = L"add standart code";

this->standartCode->UseVisualStyleBackColor = true;

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

//

// label6

//

this->label6->AutoSize = true;

this->label6->Location = System::Drawing::Point(1710, 425);

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

this->label6->Size = System::Drawing::Size(70, 16);

this->label6->TabIndex = 17;

this->label6->Text = L"Splitter(60)";

//

// splitterTable

//

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

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

this->split60,

this->Splitter, this->dataGridViewTextBoxColumn7

});

this->splitterTable->Location = System::Drawing::Point(1603, 447);

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

this->splitterTable->RowHeadersWidth = 51;

this->splitterTable->RowTemplate->Height = 24;

this->splitterTable->Size = System::Drawing::Size(430, 175);

this->splitterTable->TabIndex = 16;

//

// label7

//

this->label7->AutoSize = true;

this->label7->Location = System::Drawing::Point(1710, 220);

this->label7->Name = L"label7";

this->label7->Size = System::Drawing::Size(63, 16);

this->label7->TabIndex = 15;

this->label7->Text = L"Const(50)";

//

// constTable

//

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

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

this->const50, this->Const,

this->dataGridViewTextBoxColumn10

});

this->constTable->Location = System::Drawing::Point(1603, 242);

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

this->constTable->RowHeadersWidth = 51;

this->constTable->RowTemplate->Height = 24;

this->constTable->Size = System::Drawing::Size(430, 175);

this->constTable->TabIndex = 14;

//

// label8

//

this->label8->AutoSize = true;

this->label8->Location = System::Drawing::Point(1710, 9);

this->label8->Name = L"label8";

this->label8->Size = System::Drawing::Size(40, 16);

this->label8->TabIndex = 13;

this->label8->Text = L"Id(40)";

//

// idTable

//

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

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

this->id40, this->Name,

this->dataGridViewTextBoxColumn13

});

this->idTable->Location = System::Drawing::Point(1603, 31);

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

this->idTable->RowHeadersWidth = 51;

this->idTable->RowTemplate->Height = 24;

this->idTable->Size = System::Drawing::Size(430, 175);

this->idTable->TabIndex = 12;

//

// descriptorCode

//

this->descriptorCode->Location = System::Drawing::Point(327, 465);

this->descriptorCode->Multiline = true;

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

this->descriptorCode->ReadOnly = true;

this->descriptorCode->ScrollBars = System::Windows::Forms::ScrollBars::Both;

this->descriptorCode->Size = System::Drawing::Size(324, 179);

this->descriptorCode->TabIndex = 18;

//

// pseudoCodes

//

this->pseudoCodes->Location = System::Drawing::Point(711, 465);

this->pseudoCodes->Multiline = true;

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

this->pseudoCodes->ScrollBars = System::Windows::Forms::ScrollBars::Both;

this->pseudoCodes->Size = System::Drawing::Size(324, 179);

this->pseudoCodes->TabIndex = 19;

//

// label9

//

this->label9->AutoSize = true;

this->label9->Location = System::Drawing::Point(449, 446);

this->label9->Name = L"label9";

this->label9->Size = System::Drawing::Size(102, 16);

this->label9->TabIndex = 20;

this->label9->Text = L"DescriptorCode";

//

// label10

//

this->label10->AutoSize = true;

this->label10->Location = System::Drawing::Point(822, 446);

this->label10->Name = L"label10";

this->label10->Size = System::Drawing::Size(87, 16);

this->label10->TabIndex = 21;

this->label10->Text = L"PseudoCode";

//

// relSigns20

//

this->relSigns20->HeaderText = L"20";

this->relSigns20->MinimumWidth = 6;

this->relSigns20->Name = L"relSigns20";

this->relSigns20->Width = 125;

//

// RelationsSigns

//

this->RelationsSigns->HeaderText = L"RelationsSigns";

this->RelationsSigns->MinimumWidth = 6;

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

this->RelationsSigns->Width = 125;

//

// dataGridViewTextBoxColumn3

//

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

this->dataGridViewTextBoxColumn3->MinimumWidth = 6;

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

this->dataGridViewTextBoxColumn3->Width = 125;

//

// keyWord10

//

this->keyWord10->HeaderText = L"10";

this->keyWord10->MinimumWidth = 6;

this->keyWord10->Name = L"keyWord10";

this->keyWord10->Width = 125;

//

// KeyWord

//

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

this->KeyWord->MinimumWidth = 6;

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

this->KeyWord->Width = 125;

//

// Pseudocode

//

this->Pseudocode->HeaderText = L"Pseudocode";

this->Pseudocode->MinimumWidth = 6;

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

this->Pseudocode->Width = 125;

//

// id40

//

this->id40->HeaderText = L"40";

this->id40->MinimumWidth = 6;

this->id40->Name = L"id40";

this->id40->Width = 125;

//

// Name

//

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

this->Name->MinimumWidth = 6;

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

this->Name->Width = 125;

//

// dataGridViewTextBoxColumn13

//

this->dataGridViewTextBoxColumn13->HeaderText = L"Pseudocode";

this->dataGridViewTextBoxColumn13->MinimumWidth = 6;

this->dataGridViewTextBoxColumn13->Name = L"dataGridViewTextBoxColumn13";

this->dataGridViewTextBoxColumn13->Width = 125;

//

// const50

//

this->const50->HeaderText = L"50";

this->const50->MinimumWidth = 6;

this->const50->Name = L"const50";

this->const50->Width = 125;

//

// Const

//

this->Const->HeaderText = L"const";

this->Const->MinimumWidth = 6;

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

this->Const->Width = 125;

//

// dataGridViewTextBoxColumn10

//

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

this->dataGridViewTextBoxColumn10->MinimumWidth = 6;

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

this->dataGridViewTextBoxColumn10->Width = 125;

//

// oper30

//

this->oper30->HeaderText = L"30";

this->oper30->MinimumWidth = 6;

this->oper30->Name = L"oper30";

this->oper30->Width = 125;

//

// OperationSigns

//

this->OperationSigns->HeaderText = L"OperationSigns";

this->OperationSigns->MinimumWidth = 6;

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

this->OperationSigns->Width = 125;

//

// dataGridViewTextBoxColumn5

//

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

this->dataGridViewTextBoxColumn5->MinimumWidth = 6;

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

this->dataGridViewTextBoxColumn5->Width = 125;

//

// split60

//

this->split60->HeaderText = L"60";

this->split60->MinimumWidth = 6;

this->split60->Name = L"split60";

this->split60->Width = 125;

//

// Splitter

//

this->Splitter->HeaderText = L"Splitter";

this->Splitter->MinimumWidth = 6;

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

this->Splitter->Width = 125;

//

// dataGridViewTextBoxColumn7

//

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

this->dataGridViewTextBoxColumn7->MinimumWidth = 6;

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

this->dataGridViewTextBoxColumn7->Width = 125;

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

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

this->AutoSize = true;

this->BackColor = System::Drawing::SystemColors::HotTrack;

this->ClientSize = System::Drawing::Size(1924, 656);

this->Controls->Add(this->label10);

this->Controls->Add(this->label9);

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

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

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

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

this->Controls->Add(this->label7);

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

this->Controls->Add(this->label8);

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

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

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

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

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

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

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

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

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

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

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

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

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

this->Margin = System::Windows::Forms::Padding(3, 2, 3, 2);

this->Text = L"MyForm";

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

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

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

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

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

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

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private: LexerStates currentLexerState = LexerStates::S0;

private: String^ resultWithoutNumbers = "";

private: States currentsKeyWordState = States::S1;

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

int numbersString = LexerAnalyze();

if (currentLexerState == LexerStates::S3) {

this->result->Text = "";

MessageBox::Show("Error in line " + numbersString.ToString() + ". A multiline comment is not closed." "\n");

return;

}

ReadingKeyWords();

}

// if\_ int\_ else\_ for\_ while\_ string\_ switch\_ return\_ float\_ char\_

// double\_ const\_ do\_ class\_ enum\_ struct\_ using\_ namespace\_ static\_ public\_

// private\_ protected\_ auto\_ try\_ catch\_ throw\_ case\_ bool\_ break\_ continue\_

// default\_ delete\_ true\_ false\_ friend\_ goto\_ long\_ short\_ template\_ operator\_

// typedef\_ typename\_ sizeof\_ new\_ this\_ virtual\_ void\_ explicit\_ inline\_ nullptr\_

// and\_ extern\_ mutable\_ noexcept\_ xor\_ or\_ typeid\_ wchar\_t\_ union\_ not

// + - \* / % ++ -- = += -= \*= /= %= >> <<

// > < >= <= == !=

private: void ReadingKeyWords() {

keyWordsTable->Rows->Clear();

relationsSignsTable->Rows->Clear();

operationSignsTable->Rows->Clear();

splitterTable->Rows->Clear();

idTable->Rows->Clear();

constTable->Rows->Clear();

keyWordsTable->RowCount = 60;

idTable->RowCount = 1000;

splitterTable->RowCount = 20;

constTable->RowCount = 100;

relationsSignsTable->RowCount = 15;

operationSignsTable->RowCount = 15;

this->pseudoCodes->Text = "";

this->descriptorCode->Text = "";

currentsKeyWordState = States::S1;

String^ str = resultWithoutNumbers;

String^ word = "";

int countOfDot = 0;

int numberLine = 1;

String^ listErrors;

String^ ERROR\_IN\_KEY\_WORD = "Error in key word.";

String^ ERROR\_IN\_RELATIONS\_SIGNS = "Error in relations signs.";

String^ ERROR\_IN\_OPERATIONS\_SIGNS = "Error in operations signs.";

String^ ERROR\_IN\_CONST = "Error in constant.";

String^ ERROR\_IN\_ID = "Error in id.";

int keyWordIndex = 0;

if (str->Length < 3) str += " ";

int relationsSignsIndex = 0;

int operationsSignsIndex = 0;

int idTableIndex = 0;

int splitterTableIndex = 0;

int constTableIndex = 0;

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

switch (currentsKeyWordState)

{

case States::S1:

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

currentsKeyWordState = States::S2;

}

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

currentsKeyWordState = States::S5;

}

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

currentsKeyWordState = States::S8;

}

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

currentsKeyWordState = States::S10;

}

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

currentsKeyWordState = States::S14;

}

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

currentsKeyWordState = States::S23;

}

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

currentsKeyWordState = States::S31;

}

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

currentsKeyWordState = States::S50;

}

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

currentsKeyWordState = States::S65;

}

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

currentsKeyWordState = States::S69;

}

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

currentsKeyWordState = States::S80;

}

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

currentsKeyWordState = States::S96;

}

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

currentsKeyWordState = States::S99;

}

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

currentsKeyWordState = States::S108;

}

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

currentsKeyWordState = States::S134;

}

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

currentsKeyWordState = States::S137;

}

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

currentsKeyWordState = States::S149;

}

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

currentsKeyWordState = States::S170;

}

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

currentsKeyWordState = States::S196;

}

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

currentsKeyWordState = States::S208;

}

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

currentsKeyWordState = States::S34;

}

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

currentsKeyWordState = States::S36;

}

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

currentsKeyWordState = States::S37;

}

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

currentsKeyWordState = States::S38;

}

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

currentsKeyWordState = States::S40;

}

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

currentsKeyWordState = States::S41;

}

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

currentsKeyWordState = States::S42;

}

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

currentsKeyWordState = States::S43;

}

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

currentsKeyWordState = States::S44;

}

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

if(str[i] == '\n') numberLine++;

continue;

}

else if (isSplitterForTable(str[i])) {

currentsKeyWordState = States::S46;

}

else if (isConstant(str[i])) {

currentsKeyWordState = States::S47;

str += " ";

}

else {

currentsKeyWordState = States::S45;

}

word += str[i];

break;

case States::S2: // if int inline

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

if(i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i+1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

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

currentsKeyWordState = States::S4;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S4:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

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

currentsKeyWordState = States::S184;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S5: // else enum explicit extern

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

currentsKeyWordState = States::S6;

}

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

currentsKeyWordState = States::S61;

}

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

currentsKeyWordState = States::S178;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S6:

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

currentsKeyWordState = States::S7;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S7:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S8: // for float false friend

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

currentsKeyWordState = States::S9;

}

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

currentsKeyWordState = States::S28;

}

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

currentsKeyWordState = States::S127;

}

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

currentsKeyWordState = States::S130;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S9:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S10: // while wchar\_t

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

currentsKeyWordState = States::S11;

}

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

currentsKeyWordState = States::S211;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S11:

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

currentsKeyWordState = States::S12;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S12:

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

currentsKeyWordState = States::S13;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S13:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S14: // string switch struct static short sizeof

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

currentsKeyWordState = States::S15;

}

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

currentsKeyWordState = States::S19;

}

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

currentsKeyWordState = States::S140;

}

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

currentsKeyWordState = States::S164;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S15:

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

currentsKeyWordState = States::S16;

}

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

currentsKeyWordState = States::S77;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S16:

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

currentsKeyWordState = States::S17;

}

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

currentsKeyWordState = States::S63;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S17:

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

currentsKeyWordState = States::S18;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S18:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1]) && str[i+1] != '>') {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S19:

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

currentsKeyWordState = States::S20;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S20:

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

currentsKeyWordState = States::S21;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S21:

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

currentsKeyWordState = States::S22;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S22:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S23: // return

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

currentsKeyWordState = States::S24;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S24:

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

currentsKeyWordState = States::S25;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S25:

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

currentsKeyWordState = States::S26;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S26:

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

currentsKeyWordState = States::S27;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S27:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S28:

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

currentsKeyWordState = States::S29;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S29:

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

currentsKeyWordState = States::S30;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S30:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S31: // char const class catch case continue

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

currentsKeyWordState = States::S32;

}

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

currentsKeyWordState = States::S55;

}

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

currentsKeyWordState = States::S58;

}

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

currentsKeyWordState = States::S101;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S32:

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

currentsKeyWordState = States::S33;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S33:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S34: // > >= >>

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

if (i == str->Length - 1) {

word += str[i];

currentsKeyWordState = States::S35;

}

else if (isRelationsSigns(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_RELATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

word += str[i];

currentsKeyWordState = States::S35;

}

}

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

if (i == str->Length - 1) {

word += str[i];

currentsKeyWordState = States::S39;

}

else if (isOperationSigns(str[i + 1]) || isRelationsSigns(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

word += str[i];

currentsKeyWordState = States::S39;

}

}

else {

if (isRelationsSigns(str[i])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_RELATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

i--;

currentsKeyWordState = States::S35;

}

}

break;

case States::S36: // < <= <<

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

if (i == str->Length - 1) {

word += str[i];

currentsKeyWordState = States::S35;

}

else if (isRelationsSigns(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_RELATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

word += str[i];

currentsKeyWordState = States::S35;

}

}

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

if (i == str->Length - 1) {

word += str[i];

currentsKeyWordState = States::S39;

}

else if (isOperationSigns(str[i + 1]) || isRelationsSigns(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

word += str[i];

currentsKeyWordState = States::S39;

}

}

else {

if (isRelationsSigns(str[i])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_RELATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

i--;

currentsKeyWordState = States::S35;

}

}

break;

case States::S37: // = ==

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

if (i == str->Length - 1) {

word += str[i];

currentsKeyWordState = States::S35;

}

else if (isRelationsSigns(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_RELATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

word += str[i];

currentsKeyWordState = States::S35;

}

}

else {

if (isOperationSigns(str[i]) || isRelationsSigns(str[i])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

i--;

currentsKeyWordState = States::S39;

}

break;

case States::S38: // ! !=

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

if (i == str->Length - 1) {

word += str[i];

currentsKeyWordState = States::S35;

}

else if (isRelationsSigns(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_RELATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

word += str[i];

currentsKeyWordState = States::S35;

}

}

else {

if (isRelationsSigns(str[i])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_RELATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

i--;

currentsKeyWordState = States::S35;

}

}

break;

case States::S40: // + += ++

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

if (i == str->Length - 1) {

word += str[i];

currentsKeyWordState = States::S39;

}

else if (isOperationSigns(str[i+1]) || isRelationsSigns(str[i+1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

word += str[i];

currentsKeyWordState = States::S39;

}

}

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

if (i == str->Length - 1) {

word += str[i];

currentsKeyWordState = States::S39;

}

else if (isOperationSigns(str[i+1]) || isRelationsSigns(str[i+1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

word += str[i];

currentsKeyWordState = States::S39;

}

}

else {

if (isOperationSigns(str[i]) || isRelationsSigns(str[i])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

i--;

currentsKeyWordState = States::S39;

}

break;

case States::S41: // - -= --

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

if (i == str->Length - 1) {

word += str[i];

currentsKeyWordState = States::S39;

}

else if (isOperationSigns(str[i+1]) || isRelationsSigns(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

word += str[i];

currentsKeyWordState = States::S39;

}

}

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

if (i == str->Length - 1) {

word += str[i];

currentsKeyWordState = States::S39;

}

else if (isOperationSigns(str[i + 1]) || isRelationsSigns(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

word += str[i];

currentsKeyWordState = States::S39;

}

}

else {

if (isOperationSigns(str[i]) || isRelationsSigns(str[i])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

i--;

currentsKeyWordState = States::S39;

}

break;

case States::S42: // \* \*=

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

if (i == str->Length - 1) {

word += str[i];

currentsKeyWordState = States::S39;

}

else if (isOperationSigns(str[i + 1]) || isRelationsSigns(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

word += str[i];

currentsKeyWordState = States::S39;

}

}

else {

if (isOperationSigns(str[i]) || isRelationsSigns(str[i])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

i--;

currentsKeyWordState = States::S39;

}

break;

case States::S43: // / /=

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

if (i == str->Length - 1) {

word += str[i];

currentsKeyWordState = States::S39;

}

else if (isOperationSigns(str[i + 1]) || isRelationsSigns(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

word += str[i];

currentsKeyWordState = States::S39;

}

}

else {

if (isOperationSigns(str[i]) || isRelationsSigns(str[i])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

i--;

currentsKeyWordState = States::S39;

}

break;

case States::S44: // % %=

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

if (i == str->Length - 1) {

word += str[i];

currentsKeyWordState = States::S39;

}

else if (isOperationSigns(str[i + 1]) || isRelationsSigns(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

word += str[i];

currentsKeyWordState = States::S39;

}

}

else {

if (isOperationSigns(str[i]) || isRelationsSigns(str[i])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_OPERATIONS\_SIGNS);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

i--;

currentsKeyWordState = States::S39;

}

break;

case States::S45: // id

if (i + 1 != str->Length) {

if (isSplitterForTable(str[i + 1]) ||

isSplitterForTable(str[i]) ||

isSplitter(str[i + 1]) ||

isSplitter(str[i]) ||

isRelationsSigns(str[i]) ||

isOperationSigns(str[i])) {

if (!isSplitterForTable(str[i]) && !isRelationsSigns(str[i]) && !isOperationSigns(str[i]))

word += str[i];

else i--;

currentsKeyWordState = States::S48;

}

else {

word += str[i];

currentsKeyWordState = States::S45;

}

}

else {

if (isSplitter(str[i]) ||

isRelationsSigns(str[i]) ||

isOperationSigns(str[i])) {

i--;

}

else word += str[i];

currentsKeyWordState = States::S48;

}

break;

case States::S47: // constants

if (isConstant(str[i])) {

if (str[i] == '.' && !word->Contains("\'") && !word->Contains("\"")) countOfDot++;

currentsKeyWordState = States::S47;

word += str[i];

}

else if (isSplitterForTable(str[i]) || isOperationSigns(str[i]) || isRelationsSigns(str[i])) {

if ((word->Contains("\'") || word->Contains("\"")) && !isEndStrOrChar(word))

{

word += str[i];

continue;

}

else if (word[word->Length - 1] == '.' || countOfDot > 1) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_CONST);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

currentsKeyWordState = States::S49;

i--;

}

else word += str[i];

break;

case States::S50: // double do default delete

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

if (i == str->Length - 1) {

currentsKeyWordState = States::S3;

}

else {

if (str[i + 1] == 'u') {

currentsKeyWordState = States::S51;

}

else if(isWord(str[i+1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else {

currentsKeyWordState = States::S3;

}

}

}

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

currentsKeyWordState = States::S118;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S51:

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

currentsKeyWordState = States::S52;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S52:

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

currentsKeyWordState = States::S53;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S53:

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

currentsKeyWordState = States::S54;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S54:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S55:

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

currentsKeyWordState = States::S56;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S56:

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

currentsKeyWordState = States::S57;

}

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

currentsKeyWordState = States::S114;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S57:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S58:

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

currentsKeyWordState = States::S59;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S59:

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

currentsKeyWordState = States::S60;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S60:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S61:

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

currentsKeyWordState = States::S62;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S62:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S63:

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

currentsKeyWordState = States::S64;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S64:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S65: // using union

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

currentsKeyWordState = States::S66;

}

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

currentsKeyWordState = States::S216;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S66:

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

currentsKeyWordState = States::S67;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S67:

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

currentsKeyWordState = States::S68;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S68:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S69: // namespace new nullptr noexcept not

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

currentsKeyWordState = States::S70;

}

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

currentsKeyWordState = States::S168;

}

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

currentsKeyWordState = States::S187;

}

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

currentsKeyWordState = States::S202;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S70:

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

currentsKeyWordState = States::S71;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S71:

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

currentsKeyWordState = States::S72;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S72:

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

currentsKeyWordState = States::S73;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S73:

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

currentsKeyWordState = States::S74;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S74:

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

currentsKeyWordState = States::S75;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S75:

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

currentsKeyWordState = States::S76;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S76:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S77:

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

currentsKeyWordState = States::S78;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S78:

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

currentsKeyWordState = States::S79;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S79:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S80: // public private protected

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

currentsKeyWordState = States::S81;

}

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

currentsKeyWordState = States::S85;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S81:

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

currentsKeyWordState = States::S82;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S82:

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

currentsKeyWordState = States::S83;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S83:

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

currentsKeyWordState = States::S84;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S84:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S85:

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

currentsKeyWordState = States::S86;

}

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

currentsKeyWordState = States::S90;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S86:

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

currentsKeyWordState = States::S87;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S87:

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

currentsKeyWordState = States::S88;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S88:

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

currentsKeyWordState = States::S89;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S89:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S90:

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

currentsKeyWordState = States::S91;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S91:

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

currentsKeyWordState = States::S92;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S92:

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

currentsKeyWordState = States::S93;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S93:

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

currentsKeyWordState = States::S94;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S94:

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

currentsKeyWordState = States::S95;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S95:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S96: // auto and

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

currentsKeyWordState = States::S97;

}

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

currentsKeyWordState = States::S192;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S97:

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

currentsKeyWordState = States::S98;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S98:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S99: // try throw true template typedef typename this typeid

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

currentsKeyWordState = States::S100;

}

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

currentsKeyWordState = States::S104;

}

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

currentsKeyWordState = States::S143;

}

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

currentsKeyWordState = States::S156;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S100:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

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

currentsKeyWordState = States::S126;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S101:

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

currentsKeyWordState = States::S102;

}

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

currentsKeyWordState = States::S107;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S102:

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

currentsKeyWordState = States::S103;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S103:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S104:

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

currentsKeyWordState = States::S105;

}

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

currentsKeyWordState = States::S169;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S105:

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

currentsKeyWordState = States::S106;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S106:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S107:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S108:

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

currentsKeyWordState = States::S109;

}

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

currentsKeyWordState = States::S111;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S109:

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

currentsKeyWordState = States::S110;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S110:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S111:

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

currentsKeyWordState = States::S112;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S112:

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

currentsKeyWordState = States::S113;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S113:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S114:

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

currentsKeyWordState = States::S115;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S115:

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

currentsKeyWordState = States::S116;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S116:

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

currentsKeyWordState = States::S117;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S117:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S118:

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

currentsKeyWordState = States::S119;

}

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

currentsKeyWordState = States::S123;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S119:

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

currentsKeyWordState = States::S120;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S120:

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

currentsKeyWordState = States::S121;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S121:

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

currentsKeyWordState = States::S122;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S122:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S123:

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

currentsKeyWordState = States::S124;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S124:

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

currentsKeyWordState = States::S125;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S125:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S126:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S127:

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

currentsKeyWordState = States::S128;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S128:

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

currentsKeyWordState = States::S129;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S129:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S130:

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

currentsKeyWordState = States::S131;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S131:

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

currentsKeyWordState = States::S132;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S132:

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

currentsKeyWordState = States::S133;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S133:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S134: // goto

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

currentsKeyWordState = States::S135;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S135:

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

currentsKeyWordState = States::S136;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S136:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S137: // long

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

currentsKeyWordState = States::S138;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S138:

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

currentsKeyWordState = States::S139;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S139:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S140:

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

currentsKeyWordState = States::S141;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S141:

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

currentsKeyWordState = States::S142;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S142:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S143:

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

currentsKeyWordState = States::S144;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S144:

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

currentsKeyWordState = States::S145;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S145:

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

currentsKeyWordState = States::S146;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S146:

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

currentsKeyWordState = States::S147;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S147:

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

currentsKeyWordState = States::S148;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S148:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S149:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

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

currentsKeyWordState = States::S150;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S150:

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

currentsKeyWordState = States::S151;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S151:

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

currentsKeyWordState = States::S152;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S152:

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

currentsKeyWordState = States::S153;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S153:

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

currentsKeyWordState = States::S154;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S154:

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

currentsKeyWordState = States::S155;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S155:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S156:

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

currentsKeyWordState = States::S157;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S157:

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

currentsKeyWordState = States::S158;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S158:

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

currentsKeyWordState = States::S159;

}

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

currentsKeyWordState = States::S161;

}

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

currentsKeyWordState = States::S210;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S159:

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

currentsKeyWordState = States::S160;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S160:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S161:

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

currentsKeyWordState = States::S162;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S162:

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

currentsKeyWordState = States::S163;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S163:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S164:

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

currentsKeyWordState = States::S165;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S165:

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

currentsKeyWordState = States::S166;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S166:

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

currentsKeyWordState = States::S167;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S167:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S168:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S169:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S170: // virtual void

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

currentsKeyWordState = States::S171;

}

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

currentsKeyWordState = States::S176;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S171:

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

currentsKeyWordState = States::S172;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S172:

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

currentsKeyWordState = States::S173;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S173:

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

currentsKeyWordState = States::S174;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S174:

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

currentsKeyWordState = States::S175;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S175:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S176:

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

currentsKeyWordState = States::S177;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S177:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S178:

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

currentsKeyWordState = States::S179;

}

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

currentsKeyWordState = States::S193;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S179:

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

currentsKeyWordState = States::S180;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S180:

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

currentsKeyWordState = States::S181;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S181:

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

currentsKeyWordState = States::S182;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S182:

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

currentsKeyWordState = States::S183;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S183:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S184:

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

currentsKeyWordState = States::S185;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S185:

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

currentsKeyWordState = States::S186;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S186:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S187:

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

currentsKeyWordState = States::S188;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S188:

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

currentsKeyWordState = States::S189;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S189:

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

currentsKeyWordState = States::S190;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S190:

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

currentsKeyWordState = States::S191;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S191:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S192:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S193:

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

currentsKeyWordState = States::S194;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S194:

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

currentsKeyWordState = States::S195;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S195:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S196:

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

currentsKeyWordState = States::S197;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S197:

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

currentsKeyWordState = States::S198;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S198:

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

currentsKeyWordState = States::S199;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S199:

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

currentsKeyWordState = States::S200;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S200:

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

currentsKeyWordState = States::S201;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S201:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S202:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

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

currentsKeyWordState = States::S203;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S203:

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

currentsKeyWordState = States::S204;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S204:

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

currentsKeyWordState = States::S205;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S205:

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

currentsKeyWordState = States::S206;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S206:

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

currentsKeyWordState = States::S207;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S207:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S208:

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

currentsKeyWordState = States::S209;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S209:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S210:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S211:

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

currentsKeyWordState = States::S212;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S212:

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

currentsKeyWordState = States::S213;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S213:

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

currentsKeyWordState = States::S214;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S214:

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

currentsKeyWordState = States::S215;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S215:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S216:

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

currentsKeyWordState = States::S217;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S217:

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

currentsKeyWordState = States::S218;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

case States::S218:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_KEY\_WORD);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

str += " ";

}

if (!isSplitterForTable(str[i]) && !isOperationSigns(str[i]) && !isRelationsSigns(str[i]))

word += str[i];

else i--;

break;

default:

break;

}

if (currentsKeyWordState == States::S3) {

if (!isContainInTable(word, keyWordsTable)) {

WriteInTable(keyWordsTable, word, keyWordIndex);

keyWordIndex++;

}

this->descriptorCode->Text += getDescriptor(keyWordsTable, word);

this->pseudoCodes->Text += getPseudo(keyWordsTable, word);

word = "";

currentsKeyWordState = States::S1;

}

else if (currentsKeyWordState == States::S35) {

if (!isContainInTable(word, relationsSignsTable)) {

WriteInTable(relationsSignsTable, word, relationsSignsIndex);

relationsSignsIndex++;

}

this->descriptorCode->Text += getDescriptor(relationsSignsTable, word);

this->pseudoCodes->Text += getPseudo(relationsSignsTable, word);

word = "";

currentsKeyWordState = States::S1;

}

else if (currentsKeyWordState == States::S46) {

if (!isContainInTable(word, splitterTable)) {

WriteInTable(splitterTable, word, splitterTableIndex);

splitterTableIndex++;

}

this->descriptorCode->Text += getDescriptor(splitterTable, word);

this->pseudoCodes->Text += getPseudo(splitterTable, word);

word = "";

currentsKeyWordState = States::S1;

}

else if (currentsKeyWordState == States::S39) {

if (!isContainInTable(word, operationSignsTable)) {

WriteInTable(operationSignsTable, word, operationsSignsIndex);

operationsSignsIndex++;

}

this->descriptorCode->Text += getDescriptor(operationSignsTable, word);

this->pseudoCodes->Text += getPseudo(operationSignsTable, word);

word = "";

currentsKeyWordState = States::S1;

}

else if (currentsKeyWordState == States::S48) {

if (i != str->Length - 1) {

if (isWord(str[i + 1]) && !isOperationSigns(str[i+1]) && !isRelationsSigns(str[i+1])) continue;

}

if (isSplitterForTable(str[i])) {

i--;

}

if (!isContainInTable(word, idTable)) {

WriteInIdTable(idTable, word, idTableIndex);

idTableIndex++;

}

this->descriptorCode->Text += getDescriptor(idTable, word);

this->pseudoCodes->Text += getPseudo(idTable, word);

word = "";

currentsKeyWordState = States::S1;

}

else if (currentsKeyWordState == States::S49) {

if (word[word->Length - 1] == '.' || countOfDot > 1) {

array<String^>^ res = addError(listErrors, word, str, i, numberLine, ERROR\_IN\_CONST);

listErrors += res[0];

i = Convert::ToInt32(res[1]);

word = "";

currentsKeyWordState = States::S1;

continue;

}

else if (!isContainInTable(word, constTable)) {

WriteInConstTable(constTable, word, constTableIndex);

constTableIndex++;

}

this->descriptorCode->Text += getDescriptor(constTable, word);

this->pseudoCodes->Text += getPseudo(constTable, word);

word = "";

currentsKeyWordState = States::S1;

}

}

if(listErrors != nullptr)

MessageBox::Show(listErrors);

}

private: bool isEndStrOrChar(String^ word) {

return (word[0] == '\'' || word[0] == '\"') && ((word[word->Length - 1] == '\'' || word[word->Length - 1] == '\"')) && word->Length >= 2;

}

private: array<String^>^ addError(String^ listErrors, String^ word, String^ str, int i, int numberLine, String^ errorMessage) {

while (!isSplitter(str[i]) || i == str->Length) {

word += str[i];

i++;

}

array<String^>^ res = { ("Error in line " + numberLine.ToString() + " in word " + word + ". " + errorMessage + "\r\n") , i.ToString()};

return res;

}

private: String^ getDescriptor(DataGridView^ wordsTable, String^ word) {

if (word == " ") return "";

String^ id = "";

for (int i = 0; i < wordsTable->Rows->Count; i++) {

if (wordsTable->Rows[i]->Cells[1]->Value->ToString() == word) {

id = wordsTable->Rows[i]->Cells[0]->Value->ToString();

break;

}

}

return "(" + wordsTable->Columns[0]->HeaderText + ", " + id + ") ";

}

private: String^ getPseudo(DataGridView^ wordsTable, String^ word) {

if (word == " ") return "";

for (int i = 0; i < wordsTable->Rows->Count; i++) {

if (wordsTable->Rows[i]->Cells[1]->Value->ToString() == word) {

return wordsTable->Rows[i]->Cells[2]->Value->ToString();

}

}

return "";

}

private: void WriteInTable(DataGridView^ wordsTable, String^ word, int wordIndex) {

if (word != " ") {

wordsTable->Rows[wordIndex]->Cells[0]->Value = wordIndex + 1;

wordsTable->Rows[wordIndex]->Cells[1]->Value = word;

wordsTable->Rows[wordIndex]->Cells[2]->Value = word->ToUpper();

}

else if (resultWithoutNumbers->Contains(" ")) {

wordsTable->Rows[wordIndex]->Cells[0]->Value = wordIndex + 1;

wordsTable->Rows[wordIndex]->Cells[1]->Value = "\_";

wordsTable->Rows[wordIndex]->Cells[2]->Value = "\_";

}

}

private: void WriteInIdTable(DataGridView^ wordsTable, String^ word, int wordIndex) {

wordsTable->Rows[wordIndex]->Cells[0]->Value = wordIndex + 1;

wordsTable->Rows[wordIndex]->Cells[1]->Value = word;

wordsTable->Rows[wordIndex]->Cells[2]->Value = "id" + (wordIndex+1).ToString();

}

private: void WriteInConstTable(DataGridView^ wordsTable, String^ word, int wordIndex) {

wordsTable->Rows[wordIndex]->Cells[0]->Value = wordIndex + 1;

wordsTable->Rows[wordIndex]->Cells[1]->Value = word;

wordsTable->Rows[wordIndex]->Cells[2]->Value = "const" + (wordIndex + 1).ToString();

}

private: bool isWord(char ch) {

return !isSplitterForTable(ch);

}

private: bool isSplitter(char ch) {

std::set<char> st = { ' ', '(', ')', '[', ']', '{', '}', ':', '\r', '\n', '\t'};

bool bl = st.count(ch);

return bl;

}

private: bool isRelationsSigns(char ch) {

std::set<char> st = { '>', '<', '=', '!' };

bool bl = st.count(ch);

return bl;

}

private: bool isOperationSigns(char ch) {

std::set<char> st = { '+', '-', '=', '\*', '/', '%' };

bool bl = st.count(ch);

return bl;

}

private: bool isSplitterForTable(char ch) {

std::set<char> st = { ' ', '(', ')', '[', ']', '{', '}', ':', '.', ';', '#', ','};

bool bl = st.count(ch);

return bl;

}

private: bool isNewWord(String^ str, int i) {

if (i == 0) return true;

return isSplitter(str[i - 1]);

}

private: bool isConstant(char ch) {

std::set<char> st = { '1', '2', '3','4', '5', '6', '7', '8', '9', '0', '\'', '.', '\"' };

return st.count(ch);

}

private: bool isContainInTable(String^ word, DataGridView^ table) {

if (word == " ") word = "\_";

for (int i = 0; i < table->Rows->Count; i++) {

if (table->Rows[i]->Cells[1]->Value == nullptr) return false;

if (table->Rows[i]->Cells[1]->Value->ToString() == word) return true;

}

return false;

}

private: int LexerAnalyze() {

this->result->Text = "";

String^ str = "";

currentLexerState = LexerStates::S0;

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

switch (currentLexerState) {

case LexerStates::S0:

if (this->input->Text[i] == '/') {

currentLexerState = LexerStates::S1;

}

else if (this->input->Text[i] == ' ') {

currentLexerState = LexerStates::S5;

}

else if (this->input->Text[i] == '\t') {

currentLexerState = LexerStates::S6;

}

else if (this->input->Text[i] == '\n') {

str += this->input->Text[i];

currentLexerState = LexerStates::S7;

}

else {

str += this->input->Text[i];

}

break;

case LexerStates::S1:

if (this->input->Text[i] == '/') {

currentLexerState = LexerStates::S2;

}

else if (this->input->Text[i] == '\*') {

currentLexerState = LexerStates::S3;

}

else {

str += "/" + this->input->Text[i];

currentLexerState = LexerStates::S0;

}

break;

case LexerStates::S2:

if (this->input->Text[i] == '\n') {

str += "\r\n";

currentLexerState = LexerStates::S0;

}

break;

case LexerStates::S3:

if (this->input->Text[i] == '\*') {

currentLexerState = LexerStates::S4;

}

break;

case LexerStates::S4:

if (this->input->Text[i] == '/') {

currentLexerState = LexerStates::S0;

}

else {

currentLexerState = LexerStates::S3;

}

break;

case LexerStates::S5:

if (this->input->Text[i] == '/') {

currentLexerState = LexerStates::S1;

}

else if (this->input->Text[i] != ' ') {

currentLexerState = LexerStates::S0;

str += " ";

str += this->input->Text[i];

}

break;

case LexerStates::S6:

if (this->input->Text[i] == '/') {

currentLexerState = LexerStates::S1;

}

else if (this->input->Text[i] != '\t') {

currentLexerState = LexerStates::S0;

str += this->input->Text[i];

}

break;

case LexerStates::S7:

if (this->input->Text[i] == '\n' || this->input->Text[i] == '\r') {

currentLexerState = LexerStates::S7;

}

else if (this->input->Text[i] == '/')

currentLexerState = LexerStates::S1;

else if (this->input->Text[i] == '\t') {currentLexerState = LexerStates::S6;

}

else if (this->input->Text[i] == ' ') {currentLexerState = LexerStates::S5;}

else {

str += this->input->Text[i];

currentLexerState = LexerStates::S0;

}

break;

}

}

int numberString = 1;

resultWithoutNumbers = str;

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

if (i == 0) {

this->result->Text += numberString.ToString() + " " + str[i];

numberString++;

}

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

this->result->Text += str[i] + numberString.ToString() + " ";

numberString++;

}

else {this->result->Text += str[i];}

}

return numberString;

}

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

this->input->Text = "#include <iostream>\r\n#include <string>\r\n#include <algorithm>\r\nusing namespace std;\r\nint main() {\r\n string str;\r\n getline(cin, str);\r\n string StrNew;\r\n int vowelCount = 0;\r\n string vowels = \"aeiouAEIOU\";\r\n for (int i = 0; i < str.length(); i++) {\r\n for (int j = 0; j < vowels.length(); j++) {\r\n if (str[i] == vowels[j]) {\r\n vowelCount++;\r\n break;\r\n }\r\n }\r\n if (str[i] == \' \') {\r\n if (i > 0) {\r\n StrNew += toupper(str[++i]);\r\n }\r\n }\r\n else {\r\n StrNew += str[i];\r\n }\r\n }\r\n cout << \"Number of vowels \" << vowelCount << endl;\r\n cout << StrNew << endl;\r\n return 0;\r\n}\r\n";

}

};

};

Приложение 3.

Руководство пользователя

1. Общие сведения о программе

Исполнительный файл: Lexer.exe. Данная программа предназначена для анализа, введенного пользователем кода.

1. Описание установки

Установка программы не требуется.

1. Описание запуска

Двойным нажатием левой кнопки мыши запустить файл смотрим.exe

1. Инструкция по работе

4.1 Описание возможностей программы

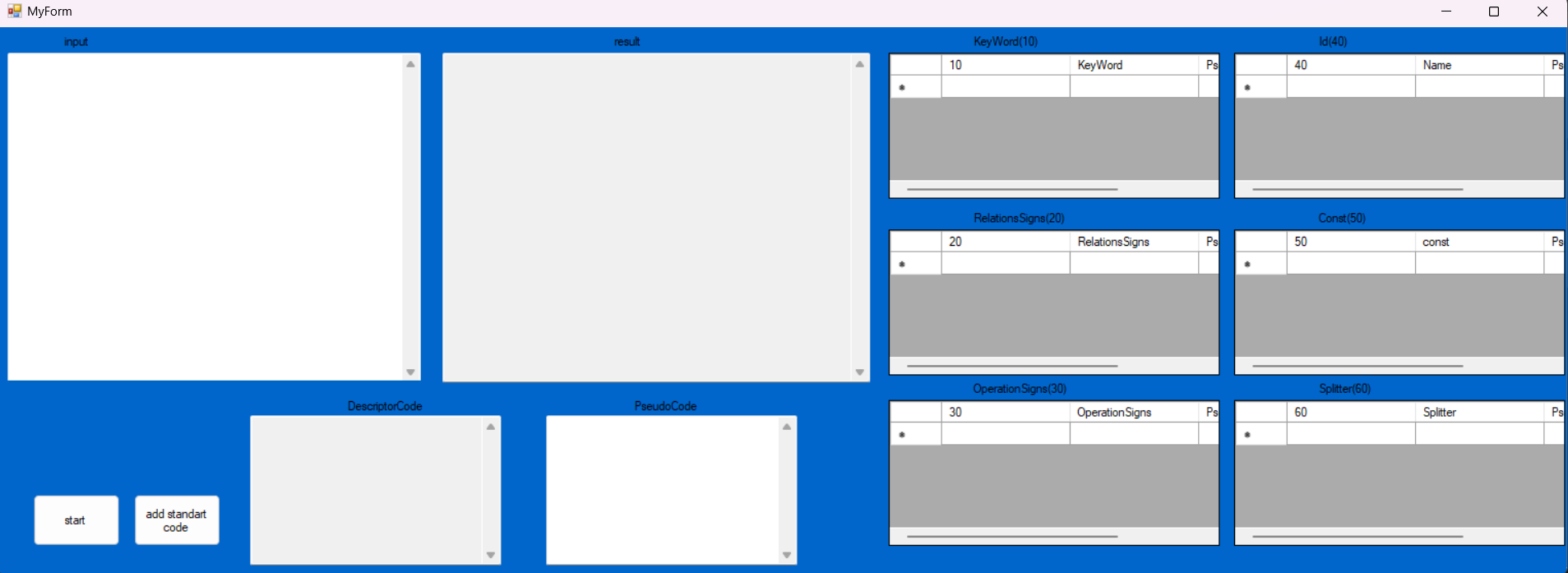


Рис. П3.1. Запуск приложения

Пользователь может взаимодействовать с тремя элементами, расположенными на окне программы: кнопкой обработки (слева внизу на рис. П3.1), кнопкой вставки стандартного кода (справа внизу на рис. П3.1) и полем для входного текста (слева сверху на рис. П3.1). Для работы с программой пользователю необходимо ввести желаемый текст (код), который следует обработать, после чего нажать кнопку «start». Все таблицы и текстовые поля заполнятся информацией, полученной при обработке входного текста (рис. П3.2).

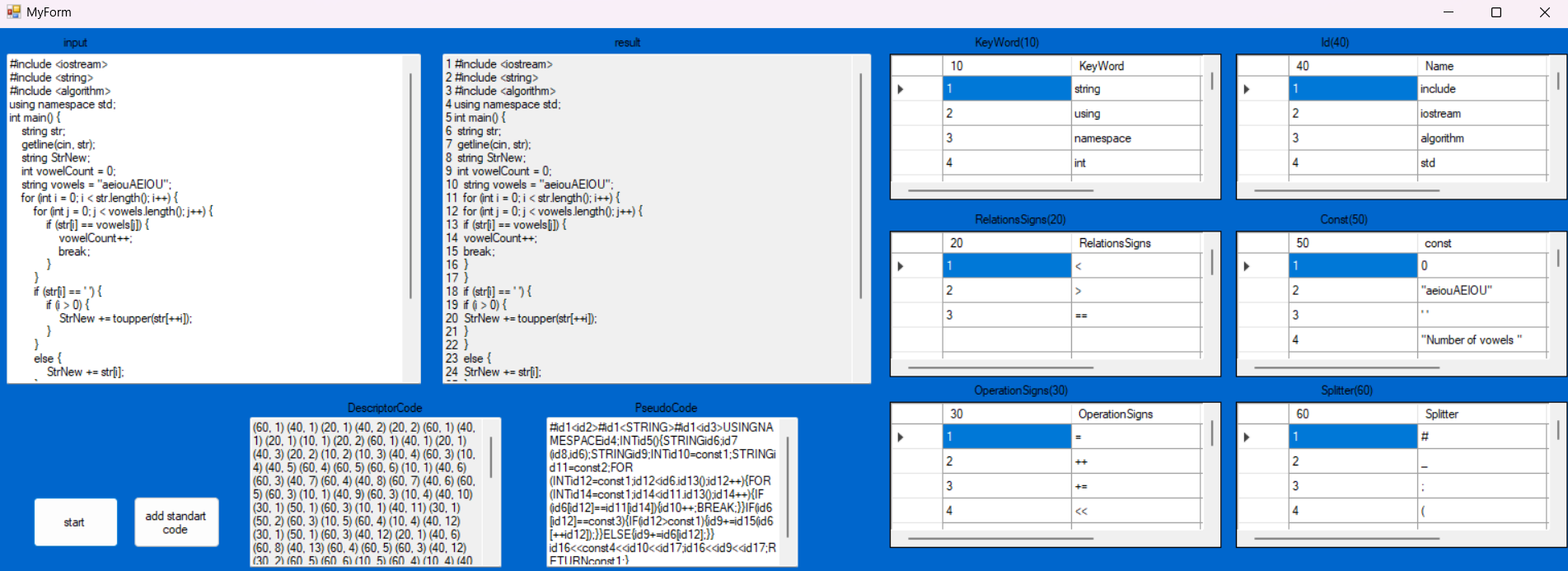


Рис. П3.2. Анализ

Чтобы очистить все поля пользователю необходимо очистить поле input и нажать снова кнопку start. Чтобы прекратить работу программы нужно закрыть рабочее окно.

# Приложение 4.

Таблица переходов

Таблица П4.1

Таблица переходов

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Входной символ/  состояние | a | b | c | d | e | f | g | i | l |
| S0 | S10 | S29 | S44 | S117 | S118 | S219 | S120 | S121 | S122 |
| S1 | - | - | - | - | - | - | - | - | - |
| S2 | - | - | - | - | - | - | - | - | - |
| S3 | - | - | - | - | - | - | - | - | - |
| S4 | - | - | - | - | - | - | - | - | - |
| S5 | - | - | - | - | - | - | - | - | - |
| S6 | - | - | - | - | - | - | - | - | - |
| S7 | - | - | - | - | - | - | - | - | - |
| S8 | - | - | - | - | - | - | - | - | - |
| S10 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | S11 |
| S11 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | S15 | **S217** |
| S15 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | S16 | **S217** | **S217** |
| S16 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** |
| S17 | S18 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** |
| S18 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** |
| S20 | - | - | - | - | - | - | - | - | - |
| S19 | **S217** | **S217** | **S217** | **S217** | **S217** | S21 | **S217** | **S217** | **S217** |
| S21 | - | - | - | - | - | - | - | - | - |
| S12 | **S217** | **S217** | **S217** | S22 | **S217** | **S217** | **S217** | **S217** | **S217** |
| S22 | - | - | - | - | - | - | - | - | - |
| S23 | **S217** | **S217** | **S217** | **S217** | S24 | **S217** | **S217** | **S217** | **S217** |
| S24 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** |
| S25 | - | - | - | - | - | - | - | - | - |
| S13 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** |
| S26 | - | - | - | - | - | - | - | - | - |
| S14 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** |
| S27 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** |
| S28 | - | - | - | - | - | - | - | - | - |
| S29 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | S30 | **S217** |
| S30 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** |
| S33 | S34 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** |
| S34 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** |
| S36 | **S217** | **S217** | **S217** | S37 | **S217** | **S217** | **S217** | **S217** | **S217** |
| S37 | - | - | - | - | - | - | - | - | - |
| S35 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** |
| S38 | - | - | - | - | - | - | - | - | - |
| S39 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | S40 |
| S40 | - | - | - | - | - | - | - | - | - |
| S32 | **S217** | **S217** | **S217** | **S217** | S41 | **S217** | **S217** | **S217** | **S217** |
| S41 | S42 | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** | **S217** |

# Приложение 5.

# Граф лексического анализатора

Приложение 6

Дерево порождений

Тут будет дерево..