МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Институт (факультет)

Кафедра

КУРСОВАЯ РАБОТА

по дисциплине Теория автоматов и формальных языков

на тему Конструирование моделей лексического и синтаксического анализа

Выполнил студент группы

*группа*

направления подготовки (специальности)

*шифр, наименование*

*фамилия, имя, отчество*

Руководитель

*фамилия, имя, отчество*

*должность*

Дата представления работы

« » 20 г. Заключение о допуске к защите

Оценка

,

количество баллов

Подпись преподавателя

Череповец,

*год*

**Аннотация**

В данной курсовой работе рассмотрен процесс конструирования моделей лексического и синтаксического анализа. Проведено изучение и описание предметной области, для заданного задания. Помимо этого, детально рассматривается каждый этап выполнения.

Курсовая работа включает в себя четыре приложения: техническое задание, блок-схемы функций, код программы и руководство пользователя.

В техническом задании, в основном, говорится о непосредственной разработке программы. Руководство пользователя содержит подробные инструкции, необходимые пользователю.

Оглавление

1. Введение 4

2. Основная часть 7

2.1 Изучение и описание предметной области 7

2.2 Постановка задачи9

2.3 Описание конечного автомата10

2.4 Лексический анализ20

2.5 Синтаксический анализ27

2.6 КС-грамматика 27

2.7 Порождение грамматики28

2.8 Форма Бэкуса-Наура и диаграммы Вирта31

2.9 Логическое проектирование34

2.10 Проектирование интерфейса35

2.11 Кодирование и тестирование36

2.12 Проведение тестирования по таблице тестовых данных37

3. Заключение38

4. Источники информации39

5. Приложение 1. Техническое задание40

6. Приложение 2. Блок-схемы основных функций45

7. Приложение 3. Текст программы48

8. Приложение 4. Руководство пользователя66

**Введение**

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

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

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

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

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

Синтаксический разбор — это основная часть компилятора на этапе анализа. Без выполнения синтаксического разбора работа компилятора бессмысленна, в то время как лексический разбор в принципе является необязательной фазой. Все задачи по проверке синтаксиса входного языка могут быть решены на этапе синтаксического разбора.

Лексический анализатор только позволяет избавить сложный по структуре синтаксический анализатор от решения примитивных задач по выявлению и запоминанию лексем входной программы.

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

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

**Основная часть**

**Изучение и описание предметной области**

Лексический анализатор (или сканер) – это часть компилятора, которая читает исходную программу и выделяет в ее тексте лексемы входного

На вход лексического анализатора поступает текст исходной программы, а выходная информация передается для дальнейшей обработки компилятором на этапе синтаксического анализа и разбора. С теоретической точки зрения лексический анализатор не является обязательной частью компилятора. Все его функции могут выполняться на этапе синтаксического разбора. Однако лексический анализ включают в состав практически всех компиляторов по следующим причинам:

– применение лексического анализатора упрощает работу с текстом исходной программы на этапе синтаксического разбора и сокращает объем обрабатываемой информации;

– для выделения в тексте и разбора лексем применяется простая и эффективная техника анализа, в то время как на этапе синтаксического анализа конструкций исходного языка используются достаточно сложные алгоритмы разбора;

– при конструкции компилятора, когда лексический анализ реализован отдельно от синтаксического, для перехода от одной версии языка программирования к другой достаточно только перестроить относительно простой лексический анализатор.

Основные функции лексического анализатора:

1) исключение из текста исходной программы комментариев;

2) исключение из текста исходной программы незначащих пробелов, символов-табуляций и перевода строки;

3) выделение лексем следующих типов: идентификаторов, строковых, символьных и числовых констант, ключевых (служебных) слов входного языка, знаков операций и разделителей.

Лексический анализатор имеет дело с такими объектами, как различного рода константы и идентификаторы (к последним относятся и ключевые слова). Язык констант и идентификаторов в большинстве случаев является регулярным, т. е. может быть описан с помощью регулярных грамматик. Распознавателями для регулярных языков являются конечные автоматы. Существуют правила, с помощью которых для любой регулярной грамматики может быть построен недетерминированный конечный автомат, распознающий цепочки языка, заданного этой грамматикой. Конечный автомат для каждой входной цепочки языка дает ответ на вопрос о том, принадлежит или нет цепочка языку, заданному автоматом.

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

– четко определить границы лексемы, которые в исходном тексте явно не заданы;

– выполнить действия для сохранения информации об обнаруженной лексеме (или выдать сообщение об ошибке, если лексема неверна)

Для большинства входных языков границы лексем распознаются по заданным терминальным символам. Эти символы – пробелы, знаки операций, символы комментариев, а также разделители (запятые, точки с запятой и т.п.) Набор таких терминальных символов может варьироваться в зависимости от синтаксиса входного языка.

Как правило, лексические анализаторы действуют по следующему принципу:

1) очередной символ из входного потока данных добавляется в лексему всегда, когда он может быть туда добавлен;

2) как только символ не может быть добавлен в лексему, то считается, что он является границей лексемы и началом следующей лексемы.

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

**Постановка задачи**

I. Разработайте грамматику для моделирования работы компилятора согласно своему варианту. Для синтаксического анализа – КС-грамматику, для лексического анализа – регулярную грамматику.

II. Постройте лексический анализатор, который решает следующие задачи: выделяет из текста входной программы все лексемы, входящие в заданную языковую конструкцию оператора switch, удаляет лишние пробелы и комментарии из входной строки. Построение лексического анализатора выполнить на основе конечного автомата. По конечному автомату построить соответствующую ему регулярную грамматику.

III. Постройте КС-грамматику, описывающую синтаксис инструкции языка программирования для оператора switch.

IV. Докажите возможность вывода заданной синтаксической конструкции с помощью грамматики разработанной в п.III. на основе заданного варианта распознавателя. Результатом работы синтаксического анализатора должно быть дерево вывода и соответствующее ему порождение рассматриваемой цепочки.

V. Путем эквивалентных преобразований получите приведенную грамматику.

Примечание 1:

1. Все конструкции должны предусматривать наличие следующих операторов: составного оператора, оператора присваивания, операторов сравнения, операторы ввода – вывода, разделителей операторов, арифметические и (или) логические операции.

2. В задании I заданную грамматику представить в виде: правил КС-грамматики, в форме БНФ, на языке синтаксических диаграмм и регулярной грамматики.

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

Примечание 2:

1. Форма Бэкуса-Науэра и синтаксические диаграммы Вирта – см. уч. Гордеева, Молчанова «Системное ПО».

2. Выход лексического анализатора – дескрипторый код и псевдокод, таблица классов лексем, сообщения об ошибках: тип ошибки и в какой строке.

3. В ходе работы ЛА должны удаляться лишние пробелы и комментарии

4. Выполнение эквивалентных преобразований КС-грамматики выполнять по алгоритмам (см. уч Гордеева, Молчанова)

**Описание конечного автомата**

Конечный автомат - набор компонентов следующего вида:

A=(V, Q, F, q, I)

V: Входным алфавитом автомата являются выбранные лексемы:

- *ключевые слова языка C++* (int, if, case, then, switch, break, default, else) (Д.к. 10);

- *идентификаторы* (могут содержать буквы, а так же цифры, за исключением случая, что идентификатор начинается с цифры) (Д.к. 20);

- *константы* (числовые и текстовые) (Д.к. 30);

- *знаки отношений, разделители и знаки операций* (,.()[]:;+-\*/<>@={}==) (Д.к. 40);

Q=(q0,q1,…,q45);

F=Q (так как автомат является распознавателем);

q=q0;

I: Пусть

Letter={A,B,C…Z,a,b,c…z,\_};

Everything= все остальное.

Digit={0,1,2…9};

Illegal={А,Б,В…Я,а,б,в…я,|,~,`,@,№,^,?,&} и недопустимые символы для текущего состояния.

*Алгоритм работы автомата:*

q0 letter -> q1

q0 digit -> q2

q0 illegal -> q5

q0 i -> q6

q0 c -> q10

q0 s -> q19

q0 b -> q25

q0 d -> q30

q0 e -> q38

q0 t -> q42

q0 == -> q0

q0 , -> q0

q0 . -> q0

q0 ( -> q0

q0 ) -> q0

q0 [-> q0

q0 ] -> q0

q0 : -> q0

q0 ; -> q0

q0 + -> q0

q0 – -> q0

q0 \* -> q0

q0 / -> q0

q0 < -> q0

q0 > -> q0

q0 @ -> q0

q0 = -> q0

q0 { -> q0

q0 } -> q0

q1 letter -> q1

q1 digit -> q1

q1 illegal -> q1

q1 everything -> q0

q2 digit -> q2

q2 . -> q3

q2 illegal -> q5

q2 everything -> q0

q3 digit -> q4

q3 everything -> q0

q5 letter -> q5

q5 digit -> q5

q5 everything -> q5

q5 illegal -> q5

q6 n -> q7

q6 f -> q37

q6 letter -> q1

q6 digit -> q1

q6 everything -> q0

q6 illegal -> q5

q7 t -> q8

q7 letter -> q1

q7 digit -> q1

q7 everything -> q0

q7 illegal -> q5

q8 letter -> q1

q8 digit -> q1

q8 everything -> q0

q8 illegal -> q5

q10 a -> q11

q10 letter -> q1

q10 digit -> q1

q10 everything -> q0

q10 illegal -> q5

q11 a -> q12

q11 letter -> q1

q11 digit -> q1

q11 everything -> q0

q11 illegal -> q5

q12 s -> q13

q12 letter -> q1

q12 digit -> q1

q12 everything -> q0

q12 illegal -> q5

q13 e -> q14

q13 letter -> q1

q13 digit -> q1

q13 everything -> q0

q13 illegal -> q5

q14 letter -> q1

q14 digit -> q1

q14 everything -> q0

q14 illegal -> q5

q19 w -> q20

q19 letter -> q1

q19 digit -> q1

q19 everything -> q0

q19 illegal -> q5

q20 i -> q21

q20 letter -> q1

q20 digit -> q1

q20 everything -> q0

q20 illegal -> q5

q21 t -> q22

q21 letter -> q1

q21 digit -> q1

q21 everything -> q0

q21 illegal -> q5

q22 c -> q23

q22 letter -> q1

q22 digit -> q1

q22 everything -> q0

q22 illegal -> q5

q23 h -> q24

q23 letter -> q1

q23 digit -> q1

q23 everything -> q0

q23 illegal -> q5

q24 letter -> q1

q24 digit -> q1

q24 everything -> q0

q24 illegal -> q5

q25 r -> q26

q25 letter -> q1

q25 digit -> q1

q25 everything -> q0

q25 illegal -> q5

q26 e -> q27

q26 letter -> q1

q26 digit -> q1

q26 everything -> q0

q26 illegal -> q5

q27 a -> q28

q27 letter -> q1

q27 digit -> q1

q27 everything -> q0

q27 illegal -> q5

q28 k -> q29

q28 letter -> q1

q28 digit -> q1

q28 everything -> q0

q28 illegal -> q5

q29 letter -> q1

q29 digit -> q1

q29 everything -> q0

q29 illegal -> q5

q30 e -> q31

q30 letter -> q1

q30 digit -> q1

q30 everything -> q0

q30 illegal -> q5

q31 f -> q32

q31 letter -> q1

q31 digit -> q1

q31 everything -> q0

q31 illegal -> q5

q32 a -> q33

q32 letter -> q1

q32 digit -> q1

q32 everything -> q0

q32 illegal -> q5

q33 u -> q34

q33 letter -> q1

q33 digit -> q1

q33 everything -> q0

q33 illegal -> q5

q34 l -> q35

q34 letter -> q1

q34 digit -> q1

q34 everything -> q0

q34 illegal -> q5

q35 t -> q36

q35 letter -> q1

q35 digit -> q1

q35 everything -> q0

q35 illegal -> q5

q36 letter -> q1

q36 digit -> q1

q36 everything -> q0

q36 illegal -> q5

q37 letter -> q1

q36 digit -> q1

q36 everything -> q0

q36 illegal -> q5

q38 l -> q39

q38 letter -> q1

q38 digit -> q1

q38 everything -> q0

q38 illegal -> q5

q39 s -> q40

q39 letter -> q1

q39 digit -> q1

q39 everything -> q0

q39 illegal -> q5

q40 l -> q41

q40 letter -> q1

q40 digit -> q1

q40 everything -> q0

q40 illegal -> q5

q41 e -> q42

q41 letter -> q1

q41 digit -> q1

q41 everything -> q0

q41 illegal -> q5

q42 h -> q43

q42 letter -> q1

q42 digit -> q1

q42 everything -> q0

q42 illegal -> q5

q43 e -> q44

q43 letter -> q1

q43 digit -> q1

q43 everything -> q0

q43 illegal -> q5

q44 n -> q44

q44 letter -> q1

q44 digit -> q1

q44 everything -> q0

q44 illegal -> q5

q45 letter -> q1

q45 digit -> q1

q45 everything -> q0

q45 illegal -> q5

Граф автомата:

**Лексический анализ**

Конечный автомат лексического анализатора работает на основе следующей регулярной грамматики:

Регулярная грамматика

G=(T,N,P,S), где T-все возможные для ввода символы

N={q0, q1… q45}

S=q0

P – следующие продукции:

q0 -> letter q1

q0 -> digit q2

q0 -> illegal q5

q0 -> i q6

q0 -> c q10

q0 -> s q19

q0 -> b q25

q0 -> d q30

q0 -> e q38

q0 -> t q42

q0 -> == q0

q0 -> , q0

q0 -> . q0

q0 -> ( q0

q0 -> ) q0

q0 -> [ q0

q0 -> ] q0

q0 -> : q0

q0 -> ; q0

q0 -> + q0

q0 -> – q0

q0 -> \* q0

q0 -> / q0

q0 -> < q0

q0 -> > q0

q0 -> @ q0

q0 -> = q0

q0 -> { q0

q0 -> } q0

q1 -> letter q1

q1 -> digit q1

q1 -> illegal q1

q1 -> everything q0

q2 -> digit q2

q2 -> . q3

q2 -> illegal q5

q2 -> everything q0

q3 -> digit q4

q3 -> everything q0

q5 -> letter q5

q5 -> digit q5

q5 -> everything q5

q5 -> illegal q5

q6 -> n q7

q6 -> f q37

q6 -> letter q1

q6 -> digit q1

q6 -> everything q0

q6 -> illegal q5

q7 -> t q8

q7 -> letter q1

q7 -> digit q1

q7 -> everything q0

q7 -> illegal q5

q8 -> letter q1

q8 -> digit q1

q8 -> everything q0

q8 -> illegal q5

q10 -> a q11

q10 -> letter q1

q10 -> digit q1

q10 -> everything q0

q10 -> illegal q5

q11 -> a q12

q11 -> letter q1

q11 -> digit q1

q11 -> everything q0

q11 -> illegal q5

q12 -> s q13

q12 -> letter q1

q12 -> digit q1

q12 -> everything q0

q12 -> illegal q5

q13 -> e q14

q13 -> letter q1

q13 -> digit q1

q13 -> everything q0

q13 -> illegal q5

q14 -> letter q1

q14 -> digit q1

q14 -> everything q0

q14 -> illegal q5

q19 -> w q20

q19 -> letter q1

q19 -> digit q1

q19 -> everything q0

q19 -> illegal q5

q20 -> i q21

q20 -> letter q1

q20 -> digit q1

q20 -> everything q0

q20 -> illegal q5

q21 -> t q22

q21 -> letter q1

q21 -> digit q1

q21 -> everything q0

q21 -> illegal q5

q22 -> c q23

q22 -> letter q1

q22 -> digit q1

q22 -> everything q0

q22 -> illegal q5

q23 -> h q24

q23 -> letter q1

q23 -> digit q1

q23 -> everything q0

q23 -> illegal q5

q24 -> letter q1

q24 -> digit q1

q24 -> everything q0

q24 -> illegal q5

q25 -> r q26

q25 -> letter q1

q25 -> digit q1

q25 -> everything q0

q25 -> illegal q5

q26 -> e q27

q26 -> letter q1

q26 -> digit q1

q26 -> everything q0

q26 -> illegal q5

q27 -> a q28

q27 -> letter q1

q27 -> digit q1

q27 -> everything q0

q27 -> illegal q5

q28 -> k q29

q28 -> letter q1

q28 -> digit q1

q28 -> everything q0

q28 -> illegal q5

q29 -> letter q1

q29 -> digit q1

q29 -> everything q0

q29 -> illegal q5

q30 -> e q31

q30 -> letter q1

q30 -> digit q1

q30 -> everything q0

q30 -> illegal q5

q31 -> f q32

q31 -> letter q1

q31 -> digit q1

q31 -> everything q0

q31 -> illegal q5

q32 -> a q33

q32 -> letter q1

q32 -> digit q1

q32 -> everything q0

q32 -> illegal q5

q33 -> u q34

q33 -> letter q1

q33 -> digit q1

q33 -> everything q0

q33 -> illegal q5

q34 -> l q35

q34 -> letter q1

q34 -> digit q1

q34 -> everything q0

q34 -> illegal q5

q35 -> t q36

q35 -> letter q1

q35 -> digit q1

q35 -> everything q0

q35 -> illegal q5

q36 -> letter q1

q36 -> digit q1

q36 -> everything q0

q36 -> illegal q5

q37 -> letter q1

q36 -> digit q1

q36 -> everything q0

q36 -> illegal q5

q38 -> l q39

q38 -> letter q1

q38 -> digit q1

q38 -> everything q0

q38 -> illegal q5

q39 -> s q40

q39 -> letter q1

q39 -> digit q1

q39 -> everything q0

q39 -> illegal q5

q40 -> l q41

q40 -> letter q1

q40 -> digit q1

q40 -> everything q0

q40 -> illegal q5

q41 -> e q42

q41 -> letter q1

q41 -> digit q1

q41 -> everything q0

q41 -> illegal q5

q42 -> h q43

q42 -> letter q1

q42 -> digit q1

q42 -> everything q0

q42 -> illegal q5

q43 -> e q44

q43 -> letter q1

q43 -> digit q1

q43 -> everything q0

q43 -> illegal q5

q44 -> n q44

q44 -> letter q1

q44 -> digit q1

q44 -> everything q0

q44 -> illegal q5

q45 -> letter q1

q45 -> digit q1

q45 -> everything q0

q45 -> illegal q5

**Синтаксический анализ**

Текст программы:

int x=1;

int y=5;

switch (x) {

case 1: {x=x+x;}

case 2: {y=y+y;}}

**КС-грамматика. Восходящий синтаксический анализатор**

G=(T,N,P,S)

Где T — конечный алфавит терминальных символов (совпадает с алфавитом языка, задаваемого грамматикой);

N — конечный алфавит нетерминальных символов;

P — конечное множество правил порождения;

S — начальный нетерминал грамматики G.

T={id, 0,1,…,9, int, switch, case,+,=,:,(,),{,},;}

N={Q, K, I, R, D}

S=Q

Где K – ключевые слова;

I – идентификаторы;

R – разделители и знаки операций;

D – цифры.

P:

1. K-> Q
2. K-> KKK
3. K-> I
4. K-> switch
5. K-> case
6. K-> int
7. K-> K RIR
8. K-> RRI
9. K->KR
10. K->RI
11. K->KI
12. I-> id/id;
13. R-> +
14. R-> {
15. R-> }
16. R-> (
17. R-> )
18. R-> ;
19. R-> =
20. R-> :
21. R-> D
22. D-> 1/2/5

**Порождение грамматики**

Q=>K

=>K

=>KKK (2)

=>KKK KR (9)

=>KK K} (15)

=>KK }} (15)

=>K K RIR}} (7)

=>K K RI;}} (18)

=>K K R id;}} (12)

=>K K +id;}} (13)

=>K I +id;}} (3)

=>K id+id;}} (13)

=>KKK id+id;}} (2)

=>KK KR id+id;}} (9)

=>KK K =id+id;}} (19)

=>KK I =id+id;}} (3)

=>KK id=id+id;}} (12)

=>K KR id=id+id;}} (9)

=>K K {id=id+id;}} (14)

=>K KR {id=id+id;}} (9)

=>K K :{id=id+id;}} (20)

=>K KR :{id=id+id;}} (9)

=>K K D:{id=id+id;}} (21)

=>K K 2:{id=id+id;}} (22)

=>K case 2:{id=id+id;}} (5)

=>KKK case 2:{id=id+id;}} (2)

=>KK KR case 2:{id=id+id;}} (9)

=>KK K } case 2:{id=id+id;}} (15)

=>KK KR } case 2:{id=id+id;}} (9)

=>KK K ;} case 2:{id=id+id;}} (18)

=>KK RI ;} case 2:{id=id+id;}} (10)

=>KK R id;} case 2:{id=id+id;}} (12)

=>KK +id;} case 2:{id=id+id;}} (13)

=>K I +id;} case 2:{id=id+id;}} (3)

=>K id+id;} case 2:{id=id+id;}} (12)

=>KKK id+id;} case 2:{id=id+id;}} (2)

=>KK KR id+id;} case 2:{id=id+id;}} (9)

=>KK K =id+id;} case 2:{id=id+id;}} (19)

=>KK I =id+id;} case 2:{id=id+id;}} (3)

=>KK id=id+id;} case 2:{id=id+id;}} (12)

=>K KR id=id+id;} case 2:{id=id+id;}} (9)

=>K K {id=id+id;} case 2:{id=id+id;}} (14)

=>K KR {id=id+id;} case 2:{id=id+id;}} (9)

=>K K : {id=id+id;} case 2:{id=id+id;}} (20)

=>K KR : {id=id+id;} case 2:{id=id+id;}} (9)

=>K K D: {id=id+id;} case 2:{id=id+id;}} (21)

=>K K 1: {id=id+id;} case 2:{id=id+id;}} (22)

=>K case 1: {id=id+id;} case 2:{id=id+id;}} (5)

=>KKK case 1: {id=id+id;} case 2:{id=id+id;}} (2)

=>KK KR case 1: {id=id+id;} case 2:{id=id+id;}} (9)

=>KK K {case 1: {id=id+id;} case 2:{id=id+id;}} (14)

=>KK RIR {case 1: {id=id+id;} case 2:{id=id+id;}} (7)

=>KK RI ){case 1: {id=id+id;} case 2:{id=id+id;}} (17)

=>KK R id){case 1: {id=id+id;} case 2:{id=id+id;}} (12)

=>KK (id){case 1: {id=id+id;} case 2:{id=id+id;}} (16)

=>K switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (4)

=>KKK switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (2)

=>KK KR switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (9)

=>KK K ; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (18)

=>KK KR ; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (9)

=>KK K D; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (21)

=>KK K 5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (22)

=>KK KR 5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (9)

=>KK K =5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (19)

=>KK I =5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (3)

=>KK id=5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (12)

=>K int id=5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (6)

=>KR int id=5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (9)

=>K ; int id=5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (18)

=>KR ; int id=5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (9)

=>K D; int id=5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (21)

=>K 1; int id=5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (22)

=>KR 1; int id=5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (9)

=>K =1; int id=5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (19)

=>KI =1; int id=5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (11)

=>K id=1; int id=5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (12)

=>int id=1; int id=5; switch (id){case 1: {id=id+id;} case 2:{id=id+id;}} (6)

**Форма Бэкуса-Наура**

G=(T,N,P,S)

N={Q, K, I, R, D}

T={id, 0,1,…,9, int, switch, case,+,=,:,(,),{,},;}

P:

S=Q

K->int|switch|case|RIR|KK|KKK|RI|RRI|KR|KI

I-> id | id;

R->+|=|:|(|)|{|}|;|K|R|D|RR

D-> 0 | 9

**Диаграммы Вирта**

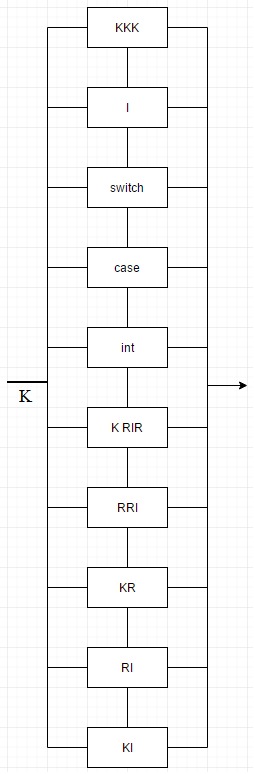
****

Рис. 1. Диаграмма для K

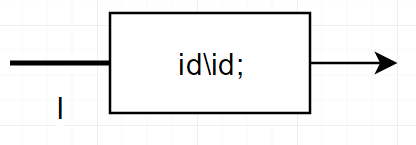
****

Рис. 2. Диаграмма для I

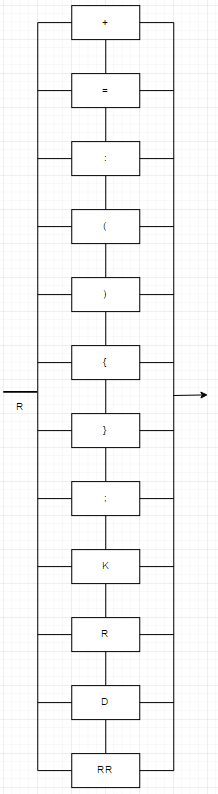
****

Рис. 3. Диаграмма для R

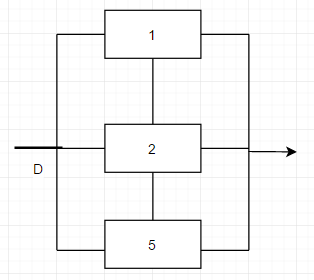
****

Рис. 4. Диаграмма для D

**Логическое проектирование**

Для написания программы, был разработан алгоритм, а по нему составлена упрощённая Блок-схема. Блок-схема программы представлена на Рис. 1.

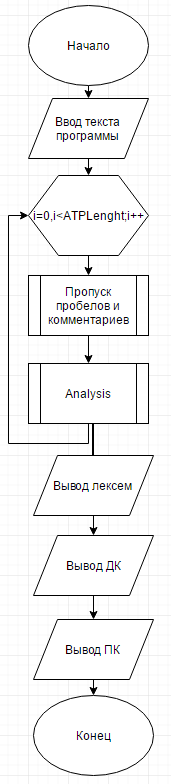


Рис. 5. Блок-схема

**Проектирование интерфейса**

Таблица 1

Таблица проектирования интерфейса

|  |  |  |
| --- | --- | --- |
| Элемент интерфейса | Реализация | Выполняемое действие |
| Поле для ввода текста программы | Поле listbox | Действий не выполняется |
| Кнопка ввод | Кнопка | Выполнение анализа текста введённого программы |
| Кнопка сброс | Кнопка | Возврат программы в изначальное состояние |
| Таблица ключевых слов | Таблица | Вывод результата |
| Таблица идентификаторов | Таблица | Вывод результата |
| Таблица числовых переменных | Таблица | Вывод результата |
| Таблица знаков пунктуации | Таблица | Вывод результата |
| Поле для вывода дескрипторного кода | Поле listbox | Вывод результата |
| Поле для вывода псевдокода | Поле listbox | Вывод результата |

**Кодирование**

Реализация разработанных алгоритмов и составленный по ним текст программы (с комментариями) в *Приложении 3*.

**Тестирование**

На этапе тестирования разработаны тестовые данные и оформлены в виде таблицы.

Таблица 2

Таблица тестовых данных

|  |  |  |  |
| --- | --- | --- | --- |
| № | Исходные данные | Тестируемый модуль или подпрограммы | Ожидаемый результат |
| 1 | int x=5; | MyForm | Выделение лексем |
| 2 | int xQx3=5.5; | MyForm | Выделение лексем |
| 3 | switch (i) { break; }; | MyForm | Выделение лексем |
| 4 | int x=5; //sadsad | MyForm | Пропуск |
| 5 | /\*int x=5;\*/ | MyForm | Пропуск |

**Проведение тестирования по таблице тестовых данных**

Таблица 3

Результаты выполнения тестирования

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Дата и время | Тестируемый модуль или подпрограмма | Кто проводил тестирование | Описание теста | Результаты тестирования |
| 1 | 9.01.2017 | MyForm | Кулагин А. | Ввод текста программы для проверки на определение лексем: int x=5; | Успех |
| 2 | 9.01.2017 | MyForm | Басистюк А.. | Ввод текста на проверку определения введённых переменных и констант: int xQx3=5.5; | Успех |
| 3 | 9.01.2017 | MyForm | Осипов А. | Проверка на определение конструкций оператора Switch:  switch (i) { break; }; | Успех |
| 4 | 9.01.2017 | MyForm | Борисов С. | Проверка на пропуск пробелов и комментариев:  int x=5; //sadsad | Успех |
| 5 | 9.01.2017 | MyForm | Стаськевич Л. | Проверка на пропуск комментария:  /\*int x=5;\*/ | Успех |

**Заключение**

В ходе выполнения курсовой работы была разработана грамматика для моделирования работы компилятора. Для синтаксического анализа – КС-грамматика, для лексического анализа – регулярную грамматика.

При написании курсового проекта была изучена специальная литература, включающая в себя статьи, учебники, руководства, описаны теоретические аспекты и раскрыты ключевые понятия исследования.

Во время написания курсовой работы был решён ряд поставленных задач:

* проанализировали задание на курсовую работу;
* определили требования к программному продукту;
* выбрали типы данных, необходимые для решения задачи;
* изучили и разработали алгоритмы, для решения задачи;
* разработали модульную структуру программы;
* написали код программы;
* выполнили тестирование.

В том числе выполнено техническое задание.

Результатом курсовой работы стала программа, написанная на C++.

**Источники информации**

1. Майкрософт [Электронный ресурс] //Microsoft. URL: <https://www.microsoft.com/> (дата обращения: 25.12.2016).

2. Майкрософт [Электронный ресурс] //Microsoft. URL: <https://msdn.microsoft.com/ru-ru/library/dn762121.aspx> (дата обращения: 25.12.2016).

*ПРИЛОЖЕНИЕ 1*

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

наименование института (факультета)

наименование кафедры

наименование дисциплины в соответствии с учебным планом

УТВЕРЖДАЮ

Зав. кафедрой ,

д.т.н., профессор Ершов Е.В.

« » 20 г.

Конструирование моделей лексического и синтаксического анализа

Техническое задание на курсовую работу

Листов

Руководитель

Ф.И.О. преподавателя

Исполнитель

студент

группа

Фамилия, имя, отчество

год

**Введение**

В процессе написания курсовой работы необходимо написать программу на языке программирования C++, моделирующую работу лексического и синтаксического анализатора. Средой для разработки служит Microsoft Visual Studio - набор инструментов для создания программного обеспечения: от планирования до разработки пользовательского интерфейса, написания кода, тестирования, отладки, анализа качества кода и производительности, развертывания в средах клиентов и сбора данных телеметрии по использованию[2].

**Основания для разработки**

Задание на Курсовую работу по Теории автоматов и формальных языков. Череповецкий Государственный Университет 2016 год.

**Назначение разработки**

Программа должна моделировать работу компилятора для оператора switch языка С++.

Требования к программе

*Требования к функциональным характеристикам:*

1. Программа должна выделять из текста входной программы все лексемы, входящие в заданную языковую конструкцию оператора switch языка С++;
2. Программа должна удалять лишние пробелы и комментарии из входной строки;
3. Программа должна преобразовывать исходный текст программы в псевдокод;
4. Программа должна преобразовывать исходный текст программы в дескрипторный код;
5. Программа должна выводить сообщения об ошибках (тип ошибки и в какой строке).

*Требования к надежности:*

* программа должна выполнять предписанные функциональные характеристики без сбоев;
* обеспечение контроля входной и выходной информации;
* защита при неверных действиях пользователя;
* контроль соответствия типов данных.

*Условия эксплуатации:*

* компьютер, поддерживающий операционную систему Windows XP;
* наличие на компьютере операционной системы Windows XP или более поздней версии;
* знание основ работы в операционной системе Windows.

*Требования к составу и параметрам технических средств:*

* для запуска программы требуется Windows XP или более поздняя версия и Super VGA видеоадаптер;
* процессор 233 МГц или лучше;
* как минимум 64 МБ Мб ОЗУ;
* не менее 1,5 ГБ свободного дискового пространства;
* для установки Windows требуется устройство для чтения компакт-дисков (или же поддержка других устройств, таких как флэш-накопителей);
* необходим монитор Super VGA с разрешением 800x600 или более высоким, отображающий 256 и более цветов;
* необходимы клавиатура и мышь.

*Требования к информационной и программной совместимости:*

* язык программирования (C++);
* среда для разработки программы (Visual Studio C++)
* операционная система (Windows XP);
* уровень защиты (без защиты).

*Требования к маркировке и упаковке:*

Распространение на электронных носителях или через средства коммуникации (для некоммерческого использования).

*Требования к транспортированию и хранению:*

Без специальных требований.

**Требования к программной документации**

*Наличие различной документации:*

* руководства пользователя;
* текст программы;
* расчетно-пояснительная записка с приложениями.

**Технико-экономические показатели**

Преимущества разработки по сравнению с существующими отечественными и зарубежными аналогами – программа должна быть простой в использовании и выполняться последовательно с описанием каждого этапа.

**Стадии и этапы разработки**

Таблица П1.1

Стадии и этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапа разработки ПО | Сроки разработки | Результат выполнения | Отметка  о выполнении |
| Получение задания | 9.10.2016 | Выполнен успешно | Выполнено |
| Анализ требований | 10.10.16-13.10.16 | Выполнен успешно | Выполнено |
| Регулярная грамматика | 13.10.16-14.10.16 | Выполнен успешно | Выполнено |
| КС грамматика | 14.10.16-19.10.16 | Выполнен успешно | Выполнено |
| Реализация | 14.10.16-19.10.16 | Выполнен успешно | Выполнено |
| Тестирование | 20.10.16-24.10.16 | Выполнен успешно | Выполнено |
| Внедрение и поддержка | 25.10.16-29.10.16 | Выполнен успешно | Выполнено |

**Порядок контроля и приемки**

Таблица П1.2

Порядок контроля и приемки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование контрольного этапа выполнения курсовой работы | Сроки контроля | Результат выполнения | Отметка о приемке результата контрольного этапа |
| Техническое задание | 20.12.16 | Успех | Выполнен |
| Написание программы | 24.12.16 | Успех | Выполнен |
| Расчётно-пояснительная записка | 25.12.16 | Успех | Выполнен |
| Руководство пользователя | 25.12.16 | Успех | Выполнен |

*ПРИЛОЖЕНИЕ 2*

**Блок-схемы основных функций**

Функции isdigit, isletter, islimiter, iserror – аналогичны, за исключением условия. В функции isdigit идёт проверка на принадлженость к цифре. Эта функция изображена на Рис. П2.1. В функции isletter идёт проверка на принадлженость к букве и знаку подчёркивания. В функции islimiter проверка на принадлженость к разделителю. В функции iserror проверка на запрещённый символ.

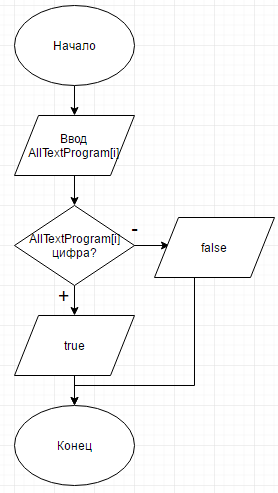


Рис. П2.1. Блок-схема функции isdigit

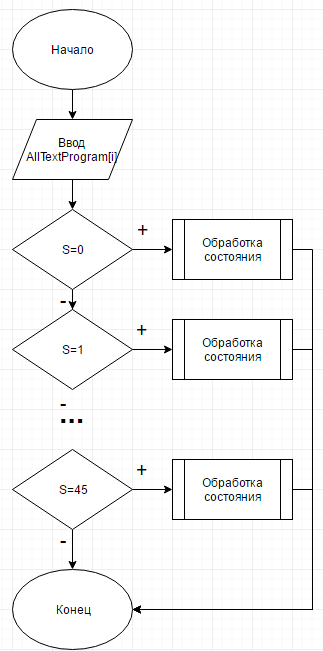


Рис. П2.2. Блок-схема функции Analys

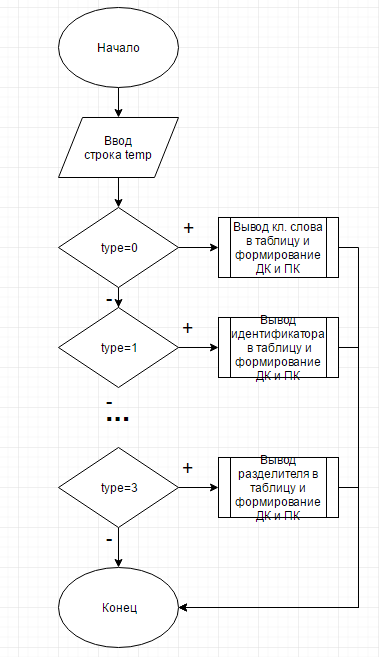


Рис. П2.3. Блок-схема функции Result

*ПРИЛОЖЕНИЕ 3*

**Текст программы**

//функции

static void Analys(char nextChar); //анализ

static void Result(std::string temp); //результат

bool isdigit(char l); //проверка на цифру

bool isletter(char l); //проверка на букву

bool islimiter(char l); //проверка на разделитель

bool iserror(char l); //проверка на ошибку

char\* SystemStringToChar(System::String^ string);

//массивы для записи результата

std::string mas0[100];

std::string mas1[100];

std::string mas2[100];

std::string mas3[100];

std::string DK; //строка для записи дескрипторного кода

std::string buffDK; //буфер для записи дескрипторного кода

std::string PK = ""; //строка для псевдокода

std::string other="Строка с ошибкой: "; //строка для неопознанных лексем

int i = 0; //счётчик

int sw0 = 0, sw1 = 0, sw2 = 0, sw3 = 0; // переменные счётчики для записи результата

int type=4; // переменная для определения типа лексемы

std::string temp=""; //буфер

std::string NULLstr; //пустая строка

std::string AllTextProgram; //строка для записи текста программы

int n, m = 0; //счётчики

int state = 0; //определение начального состояния

bool isdigit(char l) {

bool let = false;

std::string a = "0123456789";

for (int k = 0; k <= a.length(); k++)

{

if (a[k] == l)

let = true;

}

if (let == true)

return true;

else

return false;

}

bool isletter(char l) {

bool let = false;

std::string a = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz\_";

for (int k = 0; k <= a.length(); k++)

{

if (a[k] == l)

let = true;

}

if (let == true)

return true;

else

return false;

}

bool islimiter(char l) {

bool let = false;

std::string a = ",.()[]:;+-\*/<>@={} ";

for (int k = 0; k <= a.length(); k++)

{

if (a[k] == l)

let = true;

}

if (let == true)

return true;

else

return false;

}

bool iserror(char l) {

bool let = false;

std::string a = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЬЫЪЭЮЯабвгдеёжзийклмнопрстуфхцчшщьыъэюя|~`@№^?&!\_";

for (int k = 0; k <= a.length(); k++)

{

if (a[k] == l)

let = true;

}

if (let == true)

return true;

else

return false;

}

static void Analys(char nextChar) {

switch (state) {

case 0: {

if (AllTextProgram[i] == 'i') { state = 6; type = 0; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == 'c') { state = 10; type = 0; temp += AllTextProgram[i];}

else if (AllTextProgram[i] == 's') { state = 19; type = 0; temp += AllTextProgram[i];}

else if (AllTextProgram[i] == 'b' ) { state = 25; type = 0; temp += AllTextProgram[i];}

else if (AllTextProgram[i] == 'd') { state = 30; type = 0; temp += AllTextProgram[i];}

else if (AllTextProgram[i] == 'e') { state = 38; type = 0; temp += AllTextProgram[i];}

else if (AllTextProgram[i] == 't') { state = 42; type = 0; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 2; type = 2; temp += AllTextProgram[i]; }

else if (islimiter(AllTextProgram[i])) {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr; type = 4;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i];}

else { state = 0; m = i; }

break; }

case 1: {

if (isletter(AllTextProgram[i])) { state = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' '||AllTextProgram[i]=='\0'||AllTextProgram[i]=='\n') {

state = 0; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i];}

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 2: {

if (isdigit(AllTextProgram[i])) { state = 2; type = 2; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '.') { state = 3; type = 2; temp += AllTextProgram[i]; }

else if (islimiter(AllTextProgram[i])) {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i];}

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 3: {

if (isdigit(AllTextProgram[i])) { state = 4; temp += AllTextProgram[i];}

else { state = 5; }

break; }

case 4: {

if (isdigit(AllTextProgram[i])) { state = 4; temp += AllTextProgram[i];}

else if (islimiter(AllTextProgram[i])) {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state =5; temp += AllTextProgram[i];}

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 5: {

if (isletter(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i];}

else if (isdigit(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i];}

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; other += temp; other += " !!! "; temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; other += temp; other += " !!! "; temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else { state = 5; temp += AllTextProgram[i];}

break; }

case 6: {if (AllTextProgram[i] == 'n') { state = 7; type = 0; temp += AllTextProgram[i];}

else if (AllTextProgram[i] == 'f') { state = 37; type = 0; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 7: {if (AllTextProgram[i] == 't') { state = 8; temp += AllTextProgram[i];}

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 8: {if (isletter(AllTextProgram[i])) { state = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 10: {if (AllTextProgram[i] == 'a') { state = 11; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 11: {if (AllTextProgram[i] == 's') { state = 12; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 12: {if (AllTextProgram[i] == 'e') { state = 13; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 13: {if (isletter(AllTextProgram[i])) { state = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 19: {if (AllTextProgram[i] == 'w') { state = 20; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 20: {if (AllTextProgram[i] == 'i') { state = 21; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 21: {if (AllTextProgram[i] == 't') { state = 22; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 22: {if (AllTextProgram[i] == 'c') { state = 23; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 23: {if (AllTextProgram[i] == 'h') { state = 24; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 24: {if (isletter(AllTextProgram[i])) { state = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 25: {if (AllTextProgram[i] == 'r') { state = 26; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 26: {if (AllTextProgram[i] == 'e') { state = 27; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 27: {if (AllTextProgram[i] == 'a') { state = 28; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 28: {if (AllTextProgram[i] == 'k') { state = 29; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 29: {if (isletter(AllTextProgram[i])) { state = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 30: {if (AllTextProgram[i] == 'e') { state = 31; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 31: {if (AllTextProgram[i] == 'f') { state = 32; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 32: {if (AllTextProgram[i] == 'a') { state = 33; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 33: {if (AllTextProgram[i] == 'u') { state = 34; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 34: {if (AllTextProgram[i] == 'l') { state = 35; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 35: {if (AllTextProgram[i] == 't') { state = 36; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 36: {if (isletter(AllTextProgram[i])) { state = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 37: {if (isletter(AllTextProgram[i])) { state = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 38: {if (AllTextProgram[i] == 'l') { state = 39; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 39: {if (AllTextProgram[i] == 's') { state = 40; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 40: {if (AllTextProgram[i] == 'e') { state = 41; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 41: {if (isletter(AllTextProgram[i])) { state = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 42: {if (AllTextProgram[i] == 'h') { state = 43; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 43: {if (AllTextProgram[i] == 'e') { state = 44; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 44: {if (AllTextProgram[i] == 'n') { state = 45; temp += AllTextProgram[i]; }

else if (isletter(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; type = 1; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; type = 1; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

case 45: {if (isletter(AllTextProgram[i])) { state = 1; temp += AllTextProgram[i]; }

else if (isdigit(AllTextProgram[i])) { state = 1; type = 1; temp += AllTextProgram[i]; }

else if (AllTextProgram[i] == '=' && AllTextProgram[i + 1] == '=') {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; temp += AllTextProgram[i + 1]; type = 3; Result(temp); temp = NULLstr; i++;

}

else if (islimiter(AllTextProgram[i])) {

state = 0; Result(temp); temp = NULLstr;

temp += AllTextProgram[i]; type = 3; Result(temp); temp = NULLstr;

}

else if (AllTextProgram[i] == ' ' || AllTextProgram[i] == '\0' || AllTextProgram[i] == '\n') {

state = 0; Result(temp); temp = NULLstr;

}

else if (iserror(AllTextProgram[i])) { state = 5; temp += AllTextProgram[i]; }

else {

state = 0; other += AllTextProgram[i];

}

break; }

}

return;

}

static void Result(std::string temp) {

switch (type)

{

case 0:

for (int j = 0; j < 100; j++)

{

if (temp == mas0[j])

{

PK += temp;

buffDK = std::to\_string(j + 1);

DK += "(10,"; DK += buffDK; DK += ")";

return;

}

}

PK += temp;

buffDK = std::to\_string(sw0 + 1);

DK += "(10,"; DK += buffDK; DK += ")";

mas0[sw0] = temp;

sw0++;

return;

break;

case 1:

for (int j = 0; j < 100; j++)

{

if (temp == mas1[j])

{

PK += "id";

buffDK = std::to\_string(j + 1);

DK += "(20,"; DK += buffDK; DK += ")";

return;

}

}

PK += "id";

buffDK = std::to\_string(sw1 + 1);

DK += "(20,"; DK += buffDK; DK += ")";

mas1[sw1] = temp;

sw1++;

return;

break;

case 2:

for (int j = 0; j < 100; j++)

{

if (temp == mas2[j])

{

PK += "const";

buffDK = std::to\_string(j + 1);

DK += "(30,"; DK += buffDK; DK += ")";

return;

}

}

PK += "const";

buffDK = std::to\_string(sw2 + 1);

DK += "(30,"; DK += buffDK; DK += ")";

mas2[sw2] = temp;

sw2++;

return;

break;

case 3:

for (int j = 0; j < 100; j++)

{

if (temp == mas3[j])

{

PK += temp;

buffDK = std::to\_string(j + 1);

DK += "(40,"; DK += buffDK; DK += ")";

return;

}

}

PK += temp;

buffDK = std::to\_string(sw3 + 1);

DK += "(40,"; DK += buffDK; DK += ")";

mas3[sw3] = temp;

sw3++;

return;

break;

}

}

char\* SystemStringToChar(System::String^ string) {

return (char\*)(void\*)System::Runtime::InteropServices::Marshal::StringToHGlobalAnsi(string);

}

private: System::Void MyForm\_Load(System::Object^ sender, System::EventArgs^ e) {

//Создание таблицы ключевых слов

dataGridView1->ColumnCount = 3;

dataGridView1->RowCount = 1;

//

//заголовки столбцов

dataGridView1->Columns[0]->HeaderCell->Value = "10";

dataGridView1->Columns[1]->HeaderCell->Value = "Кл. сл.";

dataGridView1->Columns[2]->HeaderCell->Value = "п/к";

//

//выравнивание

dataGridView1->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);

dataGridView1->AutoResizeColumns();

//

//

//Создание таблицы индетификаторов

dataGridView2->ColumnCount = 3;

dataGridView2->RowCount = 1;

//

//заголовки столбцов

dataGridView2->Columns[0]->HeaderCell->Value = "20";

dataGridView2->Columns[1]->HeaderCell->Value = "Идент.";

dataGridView2->Columns[2]->HeaderCell->Value = "п/к";

//

//выравнивание

dataGridView2->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);

dataGridView2->AutoResizeColumns();

//

//

//Создание таблицы числовых переменных

dataGridView3->ColumnCount = 3;

dataGridView3->RowCount = 1;

//

//заголовки столбцов

dataGridView3->Columns[0]->HeaderCell->Value = "30";

dataGridView3->Columns[1]->HeaderCell->Value = "Числ. перем.";

dataGridView3->Columns[2]->HeaderCell->Value = "п/к";

//

//выравнивание

dataGridView3->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);

dataGridView3->AutoResizeColumns();

//

//

//Создание таблицы знаков

dataGridView4->ColumnCount = 3;

dataGridView4->RowCount = 1;

//

//заголовки столбцов

dataGridView4->Columns[0]->HeaderCell->Value = "40";

dataGridView4->Columns[1]->HeaderCell->Value = "Зн. оп. пункт.";

dataGridView4->Columns[2]->HeaderCell->Value = "п/к";

//

//выравнивание

dataGridView4->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);

dataGridView4->AutoResizeColumns();

//

//заполнение label

label1->Text = "Исходный текст программы";

label2->Text = "Дескрипторный код";

label3->Text = "Псевдокод";

label4->Text = "";

label5->Text = "";

}

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

String ^s="";

//s = textBox1->Text;

s = richTextBox3->Text;

std::string str;

char\* x = SystemStringToChar(s);

AllTextProgram = x;

//

//Тело программы

//

for (int j = 0; j < 100; j++)

{

mas0[j] = ""; mas1[j] = ""; mas2[j] = ""; mas3[j] = "";

};

for (i = 0; i < AllTextProgram.length(); i++)

{

char c = AllTextProgram[i];

if (AllTextProgram[i] == '\'')

{

temp += '\'';

i++;

while (AllTextProgram[i] != '\'')

{

temp += AllTextProgram[i];

i++;

}

temp += '\'';

type = 2;

Result(temp);

temp = NULLstr;

}

if (AllTextProgram[i] == '/'&&AllTextProgram[i + 1] == '\*')

{

int chet = 1;

while (chet != 0)

{

i++;

if (AllTextProgram[i] == '/'&&AllTextProgram[i + 1] == '\*')

chet++;

if (AllTextProgram[i] == '\*'&&AllTextProgram[i + 1] == '/')

chet--;

} i = i + 2;

}

if (AllTextProgram[i] == ' '&&AllTextProgram[i + 1] == ' ')

{

Analys(AllTextProgram[i]);

int chet = 1;

while (chet != 0)

{

i++;

if (AllTextProgram[i] != ' ')

chet--;

}

}

//

if (AllTextProgram[i] == '/'&&AllTextProgram[i + 1] == '/')

{

int chet = 1;

while (chet != 0)

{

i++;

if (AllTextProgram[i] == '\n'||AllTextProgram[i]=='\0')

chet--;

}

}

Analys(AllTextProgram[i]);//анализ кода

}

for (int j = 0; j < 100; j++)

{

String^ mas0converted = gcnew String(mas0[j].c\_str());

dataGridView1->Rows->Add(j+1, mas0converted, mas0converted);

String^ mas1converted = gcnew String(mas1[j].c\_str());

dataGridView2->Rows->Add(j + 1, mas1converted, "id");

String^ mas2converted = gcnew String(mas2[j].c\_str());

dataGridView3->Rows->Add(j + 1, mas2converted, "const");

String^ mas3converted = gcnew String(mas3[j].c\_str());

dataGridView4->Rows->Add(j + 1, mas3converted, mas3converted);

}

String^ DKconverted = gcnew String(DK.c\_str());

String^ PKconverted = gcnew String(PK.c\_str());

richTextBox1->Text = DKconverted;

richTextBox2->Text = PKconverted;

String^ label44 = gcnew String(other.c\_str());

label4->Text = label44;

}

*ПРИЛОЖЕНИЕ 4*

**Руководство пользователя**

**Общие сведения о программе**

Программа на C++, моделирует работу компилятора для оператора Switch.

**Описание установки**

Для использования программы потребуются:

1. Операционная система Windows Server 2008 SP2 (x86 и x64) или более поздняя версия;
2. Microsoft .NET Framework 3.0 или более поздняя версия.

**Инструкции по работе**

После запуска вы увидите интерфейс программы. Он изображён на Рис. П4.1.

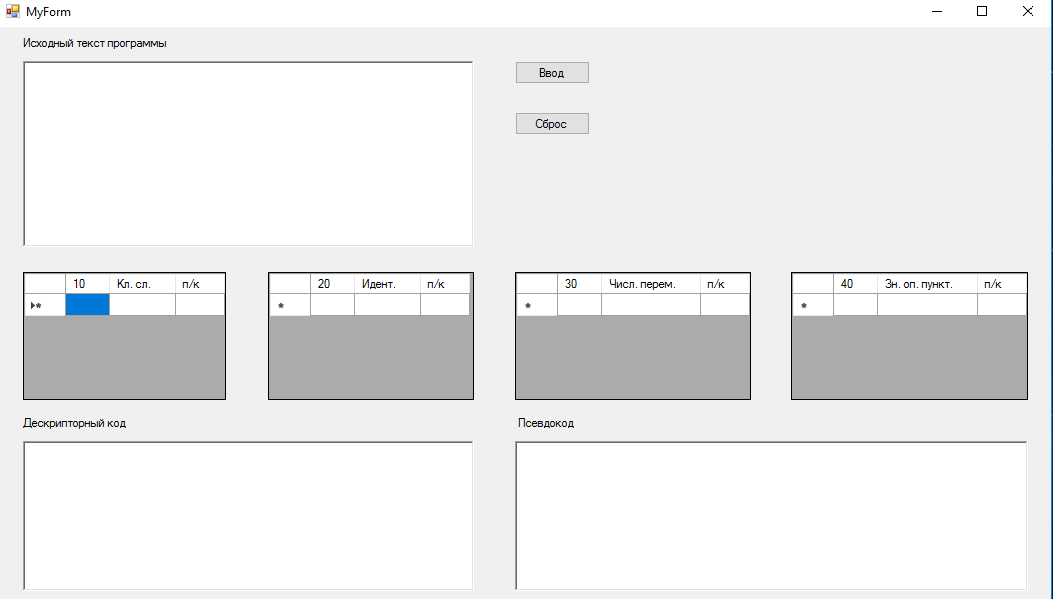


Рис. П4.1. Интерфейс программы

Ввод текста программы производится в соответствующее окно, находящееся в левом верхней части интерфейса. После нажатия на кнопку “Ввод” программа выделяет лексемы и выводит дескрипторный код и псевдокод в соответствующие поля. Результат представлен на Рис. П4.2.

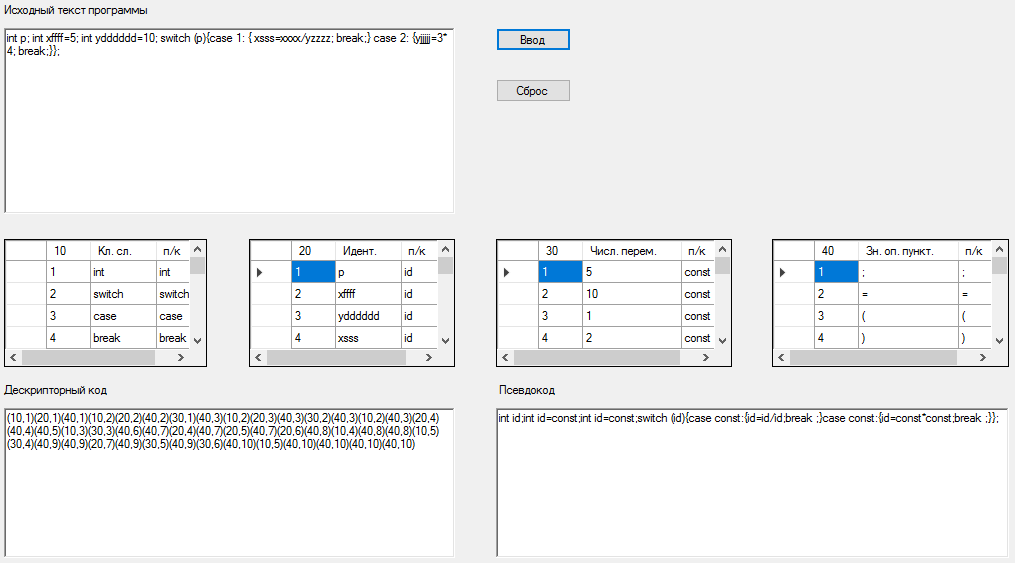


Рис. П4.2. Результат выполнения программы

Возврат программы в изначальное состояние производится при помощи нажатия на кнопку “Сброс”.