Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
федерального государственного бюджетного образовательного учреждения
высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(МИ ВлГУ)**

КУРСОВАЯ РАБОТА

по дисциплине *Теория автоматов и формальных языков*
(наименование дисциплины)
на тему *Транслятор с подмножества языка Рапира* .
(тема курсовой работы)
Выполнил(-а) студент(-ка) группы *ПИНз-121*
_____
(фамилия, имя, отчество)
Допущена к защите
Руководитель работы (нормоконтролёр) _____
(подпись, дата, расшифровка подписи)
Защищена_____ Оценка_____
(дата)
Члены комиссии_____
_____
_____
_____
(подпись, дата, расшифровка подписи)

Муром 2024

Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
федерального государственного бюджетного образовательного учреждения высшего образования
«Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых» (МИ ВлГУ)

Факультет ФИТР
«УТВЕРЖДАЮ»
Зав. кафедрой: _____
(подпись)
«___» _____ 2024 г.

ЗАДАНИЕ

На курсовую работу по курсу *Теория автоматов и формальных языков*

Студенту Сорокин И.К. *гр. ПИНз – 121*

1. Тема проекта *Транслятор с подмножества языка Рапира*
2. Сроки сдачи студентом законченного проекта *18 мая 2024*
3. Исходные данные к проекту
   3.1 *Язык для трансляции –* ***Рапира***
   3.2 *Обеспечить развернутую диагностику ошибок (лексических, синтаксических, семантических)*
   3.3 *Разработать формальную грамматику подмножества языка*
   3.4 *Синтаксический разбор на основе LL-грамматик*
   3.5 *СПодготовить несколько тестовых программ, содержащих все заданные элементы языка*
   3.6 *В языке поддерживаются*
   – *идентификаторы произвольной длины, значащие первые 8 символов;*
   – *2-байтные десятичные целые числа;*
   – *директива описания переменных типа* ***целое****;*
   – *простой* ***арифметический*** *оператор;*
   – *оператор цикла* **Для от до ::** *с предусловием;*
   3.7 *Пример программы на заданном языке*

```
целые k, i, j;
Для i от 1 до 10 ::
   Для j от 1 до 5 ::
      k = k * j;
   все;
все;
```

кнц;

4. Содержание расчетно-пояснительной записки (перечень полежащих разработке вопросов)
   Анотация (на двух языках)
   4.1 *Введение*
   4.2 *Анализ технического задания*
   4.3 *Описание грамматики языка*
   4.4 *Разработка архитектуры системы и алгоритмов*
   4.5 *Тестирование*
   4.6 *Руководство программиста*
   4.7 *Руководство пользователя*
   Заключение
5. Перечень графического материала (с точным указанием обязательных чертежей и графиков
   5.1 *Структурная и функциональная схема программы*
   5.2 *Текст программы с комментариями*
   5.3 *Скриншоты окон программы*
6. Рекомендуемая литература
   6.1 *Холкина, Н.Е. Введение в формальные грамматики и методы трансляции: учебное пособие. – Муром: ИПЦ МИ ВлГУ, 2012. – 145 с.*
   6.2 *Малявко, А.А. Формальные языки и компиляторы: учебник: – Новосибирск: НГТУ, 2014. – 431 с. – Текст: электронный [сайт]. – URL: htpps://www.iprbookshop.ru/*
   6.3 *Пентус, А.Е. Математическая теория формальных языков: учебное пособие. – М.: ИНТУИТ, Ай Пи Ар Медиа, 2020. – 218 с. – URL: https://www.iprbookshop.ru/97548*
7. Дата выдачи задания *17.02.2024*
8. Календарный график работы над проектом (на весь период проектирования)
   8.1 *Введение, анализ ТЗ, описание грамматики языка* 10%, 6 нед.
   8.2 *Разработка архитектуры системы и алгоритмов* 20%, 8 нед.
   8.3 *Реализация алгоритмов* 60%, 12 нед.
   8.4 *Тестирование* 80%, 16 нед.
   8.5 *Оформление пояснительной записки* 100%, 17 нед.

Руководитель_____
(подпись)

Задание принял к исполнению (дата) _____

Подпись студента_____

Аннотация

В курсовой работе рассматривается проблема синтаксического анализа при помощи формальных *LL*(1)-грамматик. Приведена краткая теория *LL*(1)-грамматик с подробным описанием всех алгоритмов построения соответствующих отношений и множеств. Кроме того, представлен альтернативный способ синтаксического анализа *LL*(1)-грамматик при помощи матричных преобразований, описанный в [2]. Именно такой метод реализован в данной работе.

Результатом выполнения курсовой работы является оконное приложение, созданное в среде разработки C# 2019, которое позволяет контролировать грамматики на принадлежность их к классу *LL*(1)-грамматик и выполнять синтаксический разбор соответствующей цепочки символов. Существенно, что описание грамматики одновременно является и описанием лексики языка при помощи встроенных лексических элементов типа комментарий, идентификатор, число или строка.

В работе описана грамматика подмножества языка *Panupa* с учетом семантических ограничений. Работа содержит 94 страницы и 26 рисунков.

The course work examines the problem of parsing using formal LL(1) grammars. A brief theory of LL(1) grammars is given with a detailed description of all algorithms for constructing the corresponding relations and sets. In addition, an alternative method for parsing LL(1) grammars using matrix transformations, described in [2], is presented. This is exactly the method implemented in this work.

The result of the course work is a window application created in the C# 2019 development environment, which allows you to control grammars to determine if they belong to the class of LL(1) grammars and parse the corresponding string of characters. It is important that the description of the grammar is also a description of the vocabulary of the language using built-in lexical elements such as comment, identifier, number or string.

The paper describes the grammar of a subset of the Rapier language, taking into account semantic restrictions. The work contains 94 pages and 26 pictures.

Оглавление

1. ВВЕДЕНИЕ

Сейчас даже трудно представить себе программирование в машинном коде, которое было единственным способом написания программ до появления языков программирования высокого уровня. Самым знаменитым пионером такой серии языков можно считать Фортран. При конструировании компилятора Фортран (да и при определении самого языка) практически не применялась теория формальных грамматик. Поэтому методы разработки алгоритмов трансляции были скорее искусством, а не обоснованной научной теорией.

Все изменилось с появлением Алголо-подобных языков программирования. Для более строгого определения языка стала применяться формулировка Бэкуса-Наура, семантика конструкций языка также подробно описывалась. К этому времени уже были определенные достижения по теории контекстно свободных *LL*(1)-грамматик, которыми можно было описать практически все конструкции языка программирования. Имея в своем распоряжении программу, реализующую синтаксический разбор языка программирования на базе *LL*(1)-грамматик, можно было считать половину работы создания компилятора более-менее автоматизированной.

Поэтому в данной работе предпринята попытка создания автоматического синтаксического и лексического разбора именно при помощи *LL*(1)-грамматик. Практически для этого обычно выполняются также автоматические предварительные действия по преобразованию грамматик к виду, пригодных для алгоритма работы с *LL*(1)-грамматиками (в частности, удаление левой рекурсии). Такие преобразования могут быть темой отдельной курсовой работы и здесь не рассматриваются. Однако контроль грамматики на принадлежность ее к типу *LL*(1) в любом случае выполняется.

В описание грамматики введены массово используемые терминальные символы типа целых и вещественных констант, идентификаторов и служебных слов. В результате лексический разбор получается автоматическим, включая популярно используемые виды однострочных и многострочных комментариев в стиле языка C++. Все это помогает упростить описание синтаксиса языка с применением укрупненных лексем. В противном случае при прямом определении синтаксиса того же вещественного числа пришлось бы писать не одно синтаксическое правило. Более подробно описание структуры грамматики изложено в соответствующем пункте описания.

2. АНАЛИЗ ТЕХНИЧЕСКОГО ЗАДАНИЯ

2.1 Формулировка задания

Разработать программу для автоматического лексического и синтаксического разбора модельного языка программирования, определенного при помощи *LL*(1)-грамматики. Программа должна транслировать любые программы, для которых существует *LL*(1)-грамматика.

Если грамматика не является *LL*(1)-грамматикой, программа должна сообщать конкретную причину такой ситуации.

Программа должна отображать все этапы построения необходимых таблиц для создания автомата разбора конструкций языка.

2.2 Краткая теория

Для начала предположим, что **G**=(**N**,**E**,**P**,**S**) – однозначная грамматика и $w=a_1,a_2...a_n$ – цепочка из **L**(**G**). Тогда существует единственная последовательность левовыводимых цепочек $b_0,b_1..b_m$, для которой **S**=$b_0,b_i,p_i…b_{i+1}$ при 0<=i<m и a_m=w. Последовательность $p_0p_1..p_{m-1}$ – левый разбор цепочки w.

Допустим, что мы хотим найти этот левый разбор, просматривая w один раз слева направо. Можно попытаться сделать это, строя последовательность левовыводимых цепочек $b_0,b_1..b_m$. Если $b_i=a_1,a_2...a_jAB$, то к данному моменту анализа мы уже прочли первые j входных символов и сравнили их с первыми j символами цепочки b_i. Было бы желательно определить b_{i+1}, зная только $a_1,a_2...a_j$ (часть входной цепочки, считанную к данному моменту), несколько следующих входных символов ($a_{j+1}a_{j+2}...a_{j+k}$ для некоторого фиксированного k) и нетерминал A. Если эти три фактора однозначно определяют, какое правило надо применить для развертки нетерминала A, то a_{i+1} точно определяется по a_i и k входным символам $a_{j+1}a_{j+2}...a_{j+k}$.

Грамматика, в которой каждый левый вывод обладает этим свойством, называется *LL*(k)-грамматикой. Доказано, что для каждой *LL*(k)-грамматики можно построить детерминированный левый анализатор, работающий линейное время.

Определение 1. КС-грамматика обладает свойством *LL*(*k*) для некоторого *k*>0, если на каждом шаге вывода для однозначного выбора очередной альтернативы МП-автомату достаточно знать символ на верхушке

стека и рассмотреть первые k символов от текущего положения считывающей головки во входной строке.

Определение 2. КС-грамматика называется $LL(k)$-грамматикой, если она обладает свойством $LL(k)$ для некоторого $k>0$.

В основе распознавателя $LL(k)$-грамматик лежит левосторонний разбор строки языка. Исходной сентенциальной формой является начальный символ грамматики, а целевой – заданная строка языка. На каждом шаге разбора правило грамматики применяется к самому левому нетерминалу сентенции. Данный процесс соответствует построению дерева разбора цепочки сверху вниз (от корня к листьям). Отсюда и произошла аббревиатура $LL(k)$: первая «L» (от слова «*left*») означает левосторонний ввод исходной цепочки символов, вторая «L» – левосторонний вывод в процессе работы распознавателя.

Определение 3. Для построения распознавателей для $LL(k)$-грамматик используются два множества:

- $FIRST(k, \alpha)$ – множество терминальных цепочек, выводимых из цепочки $\alpha \in (V_T \cup V_N)^*$, укороченных до k символов;
- $FOLLOW(k, A)$ – множество укороченных до k символов терминальных цепочек, которые могут следовать непосредственно за $A \in V_N$ в цепочках вывода.

Формально эти множества можно определить следующим образом:

- $FIRST(k, \alpha) = \{\omega \in V_T^* \mid \exists$ вывод $\alpha \Rightarrow^* \omega$ и $|\omega| \le k$ или $\exists$ вывод $\alpha \Rightarrow^* \omega x$ и $|\omega| = k; x, \alpha \in (V_T \cup V_N)^*, k > 0\}$;
- $FOLLOW(k, A) = \{\omega \in V_T^* \mid \exists$ вывод $S \Rightarrow^* \alpha A \gamma$ и $\omega \in FIRST(k, \gamma); \alpha, \gamma \in V^*, A \in V_N, k>0\}$.

Теорема 1. Необходимое и достаточное условие *LL*(1)-грамматики

Для того чтобы грамматика $G(V_N, V_T, P, S)$ была $LL(1)$-грамматикой необходимо и достаточно, чтобы для каждого символа $A \in V_N$, у которого в грамматике существует более одного правила вида $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, выполнялось требование:

$FIRST(1, \alpha_i FOLLOW(1, A)) \cap FIRST(1, \alpha_j FOLLOW(1, A)) = \varnothing$,

$\forall\ i \ne j,\ 0<i \le n,\ 0<j \le n$.

Т.е. если для символа A отсутствует правило вида $A \to \varepsilon$, то все множества $FIRST(1, \alpha_1)$, $FIRST(1, \alpha_2)$,…, $FIRST(1, \alpha_n)$ должны попарно не пересекаться, если же присутствует правило $A \to \varepsilon$, то они не должны также пересекаться с множеством $FOLLOW(1, A)$.

Для построения распознавателей для $LL(1)$-грамматик необходимо построить множества $FIRST(1, x)$ и $FOLLOW(1, A)$. Причем, если строка x будет начинаться с терминального символа a, то $FIRST(1, x)=a$, и если она будет начинаться с нетерминального символа A, то $FIRST(1, x)=FIRST(1, A)$. Следовательно, достаточно рассмотреть алгоритмы построения множеств $FIRST(1, A)$ и $FOLLOW(1, A)$ для каждого нетерминального символа A.

Алгоритм 1. Построение множества *FIRST*(1, *A*)

Для выполнения алгоритма необходимо предварительно преобразовать исходную грамматику G в грамматику G', не содержащую ε-правил. Алгоритм построения множества $FIRST(1, A)$ использует грамматику G'.

Шаг 1. Первоначально внести во множество первых символов для каждого нетерминального символа A все символы, стоящие в начале правых частей правил для этого нетерминала, т.е.

$$\forall\, A \in V_N\ FIRST_0(1, A) = \{X \mid A \to X\alpha \in P, X \in (V_T \cup V_N), \alpha \in (V_T \cup V_N)^*\}.$$

Шаг 2. Для всех $A \in V_N$ положить:

$$FIRST_{i+1}(1, A) = FIRST_i(1, A) \cup FIRST_i(1, B),\ \forall\, B \in (FIRST(1, A) \cap V_N).$$

Шаг 3. Если существует $A \in V_N$, такой что $FIRST_{i+1}(1, A) \neq FIRST_i(1, A)$, то присвоить $i=i+1$ и вернуться к шагу 2, иначе перейти к шагу 4.

Шаг 4. Исключить из построенных множеств все нетерминальные символы, т.е.

$$\forall\, A \in V_N\ FIRST(1, A) = FIRST_i(1, A)_{\setminus N}.$$

Алгоритм 2. Построение множества *FOLLOW*(1, *A*)

Алгоритм основан на использовании правил вывода грамматики G.

Шаг 1. Первоначально внести во множество последующих символов для каждого нетерминального символа A все символы, которые в правых частях правил вывода встречаются непосредственно за символом A, т.е.

$$\forall\, A \in V_N\ FOLLOW_0(1, A) = \{X \mid \exists\, B \to \alpha A X \beta \in P, B \in V_N, X \in (V_T \cup V_N),$$
$$\alpha, \beta \in (V_T \cup V_N)^*\}.$$

Шаг 2. Внести пустую строку во множество $FOLLOW(1, S)$, т.е.

$$FOLLOW(1, S) = FOLLOW(1, S) \cup \{\varepsilon\}.$$

Шаг 3. Для всех $A \in V_N$ вычислить:

$$FOLLOW'_i(1,A)=FOLLOW_i(1,A) \cup FIRST(1,B), \forall B \in (FOLLOW_i(1,A) \cap V_N).$$

Шаг 4. Для всех $A \in V_N$ положить:

$FOLLOW''_i(1, A) = FOLLOW'_i(1, A) \cup FOLLOW'_i(1, B)$,

$\forall B \in (FOLLOW'_i(1, A) \cap V_N)$, если $\exists$ правило $B \to \varepsilon$.

Шаг 5. Для всех $A \in V_N$ определить:

$FOLLOW_{i+1}(1, A) = FOLLOW''_i(1, A) \cup FOLLOW''_i(1, B)$,

для всех нетерминальных символов $B \in V_N$, имеющих правило вида

$B \to \alpha A$, $\alpha \in (V_T \cup V_N)^*$.

Шаг 6. Если существует $A \in V_N$ такой, что $FOLLOW_{i+1}(1,A) \neq FOLLOW_i(1,A)$, то положить $i := i+1$ и вернуться к шагу 3, иначе перейти к шагу 7.

Шаг 7. Исключить из построенных множеств все нетерминальные символы, т.е. $\forall A \in V_N \; FOLLOW(1, A) = FOLLOW_i(1, A)_{\setminus N}$.

Алгоритм 3. Функционирование распознавателя цепочек для *LL*(1)-грамматик

Шаг 1. Помещаем в стек начальный символ грамматики S, а во входной буфер исходную цепочку символов.

Шаг 2. До тех пор пока в стеке и во входном буфере останется только пустая строка ε либо будет обнаружена ошибка в алгоритме разбора, выполняем одно из следующих действий:

– если на верхушке стека находится нетерминальный символ A и очередной символ входной строки символ a, то выполняем операцию «свертка» по правилу $A \to x$ при условии, что $a \in FIRST(1, x)$, т.е. извлекаем из стека символ A и заносим в стек строку x, не меняя содержимого входного буфера;

– если на верхушке стека находится нетерминальный символ A и очередной символ входной строки символ a, то выполняем операцию «свертка» по правилу $A \to \varepsilon$ при условии, что $a \in FOLLOW(1, A)$, т.е. извлекаем из стека символ A и заносим в стек строку ε, не меняя содержимого входного буфера;

– если на верхушке стека находится терминальный символ a, совпадающий с очередным символом входной строки, то выполняем операцию «выброс», т.е. удаляем из стека и входного буфера данный терминальный символ;

– если содержимое стека и входного буфера пусто, то исходная строка

прочитана полностью, и разбор завершен удачно;

– если ни одно из данных условий не выполнено, то цепочка не принадлежит заданному языку, и алгоритм завершает свою работу с ошибкой.

Пример 1. Дана грамматика G ({S, T, R}, {+, -, (,), a, b}, P, S), с правилами P: 1) $S \to TR$; 2) $R \to \varepsilon \mid +TR \mid -TR$; 3) $T \to (S) \mid a \mid b$. Построить распознаватель для строки (a+(b-a)) языка грамматики G.

Этап 1. Преобразуем грамматику G в грамматику G', не содержащую ε-правил:

$N_0 = \{R\}$;

$N_1 = \{R\}$, т.к. $N_0 = N_1$, то во множество P' войдут правила:

1) $S \to TR \mid T$; 2) $R \to +TR \mid +T \mid -TR \mid -T$; 3) $T \to (S) \mid a \mid b$.

Этап 2. Построение множеств $FIRST(1, A)$ для каждого нетерминала A представлено в таблице 1.

Таблица 1 – Построение множеств $FIRST(1, A)$

$FIRST_i(1, A)$	0	1	2	$FIRST(1, A)$
S	T	T, (, a, b	T, (, a, b	(, a, b
R	+, -	+, -	+, -	+, -
T	(, a, b	(, a, b	(, a, b	(, a, b

Этап 3. Построение множеств $FOLLOW(1, A)$ для каждого нетерминала A представлено в таблице 2.

Таблица 2 – Построение множеств *FOLLOW*(1, *A*)

Шаг	Нетерминалы	$FOLLOW_i(1, A)$	$FOLLOW_i'(1, A)$	$FOLLOW_i''(1, A)$
0	***S***	**)**	**), ε**	**), ε**
	R	**∅**	**∅**	**∅**
	T	***R***	***R*, +, -**	***R*, +, -**
1	***S***	**), ε**	**), ε**	**), ε**
	R	**), ε**	**), ε**	**), ε**
	T	***R*, +, -**	***R*, +, -**	***R*, +, -,), ε**
2	***S***	**), ε**	**), ε**	**), ε**
	R	**), ε**	**), ε**	**), ε**
	T	***R*, +, -,), ε**	***R*, +, -,), ε**	***R*, +, -,), ε**
***FOLLOW*(1, *S*)**		**), ε**		
***FOLLOW*(1, *R*)**		**), ε**		
***FOLLOW*(1, *T*)**		**+, -,), ε**		

Этап 4. Множества *FIRST*(1, *A*) и *FOLLOW*(1, *A*) для каждого нетерминала *A* сведены в таблицу 3.

Таблица 3 – Множества *FIRST*(1, *A*) и *FOLLOW*(1, *A*)

A	***FIRST*(1, *A*)**	***FOLLOW*(1, *A*)**
S	**(, *a*, *b***	**), ε**
R	**+, -**	**), ε**
T	**(, *a*, *b***	**+, -,), ε**

Грамматика *G* является *LL*(1)-грамматикой, т.к. для каждого нетерминала *A*, имеющего альтернативные выводы, множества *FIRST*(1, *A*) попарно не пересекаются, а для нетерминала *R* они также не пересекаются со

множеством $FOLLOW(1, R)$.

Шаг 5. Разбор строки $(a+(b-a))$ для грамматики G показан в таблице 4.

Таблица 4 - Разбор строки $(a+(b-a))$ для грамматики G

Стек	Входной буфер	Действие
S	$(a+(b-a))$	свертка $S \to TR$, т.к. $(\in FIRST(1, TR)$
TR	$(a+(b-a))$	свертка $T \to (S)$, т.к. $(\in FIRST(1, (S))$
$(S)R$	$(a+(b-a))$	выброс
$S)R$	$a+(b-a))$	свертка $S \to TR$, т.к. $a \in FIRST(1, TR)$
$TR)R$	$a+(b-a))$	свертка $T \to a$, т.к. $a \in FIRST(1, a)$
$aR)R$	$a+(b-a))$	выброс
$R)R$	$+(b-a))$	свертка $R \to +TR$, т.к. $+ \in FIRST(1, TR)$
$+TR)R$	$+(b-a))$	выброс
$TR)R$	$(b-a))$	свертка $T \to (S)$, т.к. $(\in FIRST(1, (S))$
$(S)R)R$	$(b-a))$	выброс
$S)R)R$	$b-a))$	свертка $S \to TR$, т.к. $b \in FIRST(1, TR)$
$TR)R)R$	$b-a))$	свертка $T \to b$, т.к. $b \in FIRST(1, b)$
$bR)R)R$	$b-a))$	выброс
$R)R)R$	$-a))$	свертка $R \to -TR$, т.к. $- \in FIRST(1, -TR)$
$-TR)R)R$	$-a))$	выброс
$TR)R)R$	$a))$	свертка $T \to a$, т.к. $a \in FIRST(1, a)$
$aR)R)R$	$a))$	выброс
$R)R)R$	$))$	свертка $R \to \varepsilon$, т.к. $) \in FOLLOW(1, R)$
$)R)R$	$))$	выброс
$R)R$	$)$	свертка $R \to \varepsilon$, т.к. $) \in FOLLOW(1, R)$
$)R$	$)$	выброс
R	ε	свертка $R \to \varepsilon$, т.к. $\varepsilon \in FOLLOW(1, R)$

ε	**ε**	**строка принята полностью**

Шаг 6. Получили следующую цепочку вывода

$S \Rightarrow TR \Rightarrow (S)R \Rightarrow (TR)R \Rightarrow (aR)R \Rightarrow (a+TR)R \Rightarrow (a+(S)R)R \Rightarrow (a+(TR)R)R \Rightarrow$

$\Rightarrow (a+(bR)R)R \Rightarrow (a+(b-TR)R)R \Rightarrow (a+(b-aR)R)R \Rightarrow (a+(b-a)R)R \Rightarrow (a+(b-a))R$

$\Rightarrow (a+(b-a))$.

Нисходящее дерево разбора цепочки представлено на рисунке 1.

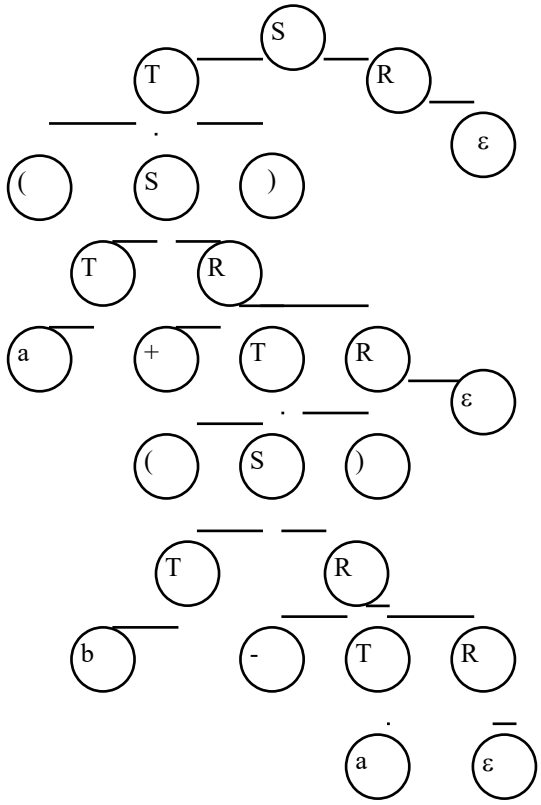


Рис. 1 Дерево вывода для цепочки $(a+(b-a))$ в грамматике G

3. ОПИСАНИЕ ГРАММАТИКИ ЯЗЫКА

Входными данными программы являются КС-грамматики, алфавит которых для нетерминальных символов состоит из произвольного текста, заключенного в уголковые скобки (обычно используется только английская буква в диапазоне от ‘A’ до ‘Z’ включительно, что не очень удобно и в некоторых случаях недостаточно). В качестве терминальных символов могут быть использованы специальные лексемы ‘сим’, ‘стр’, ‘ид’, ‘цел’ и ‘вещ’, обозначающие собой соответственно символьную константу и строку в стиле языка C++, идентификатор, целое и вещественное число. Все остальные терминальные символы могут быть представлены в виде зарезервированных слов языка программирования, взятых в апострофы, или в виде обычных односимвольных разделителей, за исключением символов пробела и табуляции. Вертикальная черта (‘|’) используется как признак альтернативы правил; символ ‘#’ служит в качестве обозначения пустой цепочки, традиционно обозначаемой как **ε**.

Для пояснений в тексте грамматики могут использоваться строки комментарии, начинающиеся символом ‘;’. Пустые строки и пробелы можно использовать в любом месте описания грамматики.

Дополнительно в комментариях грамматики можно указывать варианты использования комментариев, игнорируемых на этапе лексического анализа. Если не указаны следующие варианты (их может быть задано одновременно несколько, например, как однстрочный комментарий // и многострочный /* … */ в стиле языка программирования C++), то транслируемая программа не может содержать комментарии:

; #КОММЕНТАРИЙ // – однострочный комментарий C++ //

; #КОММЕНТАРИЙ /* – многострочный комментарий C++ /* … */

; #КОММЕНТАРИЙ (* – многострочный комментарий (* … *) Рапира

; #КОММЕНТАРИЙ { – многострочный комментарий ПАСКАЛЬ {…}

Кроме того, можно указать набор букв, используемых в идентификаторе, в том же комментарии следующим образом (для языка Рапира надо задавать оба алфавита, по умолчанию задан английский алфавит):

; #ИДЕНТИФИКАТОР РУССКИЙ – только русские буквы

; #ИДЕНТИФИКАТОР АНГЛИЙСКИЙ – только английские буквы

; #ИДЕНТИФИКАТОР РУССКИЙ_АНГЛИЙСКИЙ – оба алфавита

Описанные выше настройки лексического анализа должны задаваться перед синтаксическими правилами грамматики языка. Обработка семантических ошибок может задаваться в комментариях грамматики, расположенных после синтаксических правил грамматики следующим образом (после символа ‘$’ должна задаваться цифра, выполняющая семантический анализ в виде некоторой функции при обработке заданного нетерминала):

; $0 <НЕТЕРМИНАЛ>

Ниже представлена грамматика подмножества языка Рапира с некоторыми расширениями по сравнению с исходным заданием:

```
; Грамматика подмножества языка программирования Рапира
; #ИДЕНТИФИКАТОР РУССКИЙ_АНГЛИЙСКИЙ
; #КОММЕНТАРИЙ (*

<S> -> <ОписаниеПеременных> <Прог> 'КНЦ';
<ОписаниеПеременных> -> 'ЦЕЛЫЕ' <Ид1> <Список>
<Список> -> , <Ид1> <Список> | ; <НовоеОписание>
<НовоеОписание> -> <ОписаниеПеременных> | #
<Прог>         -> <Set> | <For> | <Rep> | <While> | <If> | #
<Set>          -> <Присваивание> = <Выр>; <Прог>
<For> -> 'ДЛЯ' <Ид2> 'ОТ' <Выр> 'ДО' <Выр> <Step> <Продолжение1>
<Rep>                -> 'ПОВТОР' <Выр> <РазРаза> <Продолжение>
<While>              -> 'ПОКА' <Условие> <Продолжение>
<If>   -> 'ЕСЛИ' <Условие> 'ТО' <Прог> <Else> 'ВСЕ'; <Прог>
<Else>               -> 'ИНАЧЕ' <Прог> | #
<Step>               -> 'ШАГ' <Выр> | #
<Продолжение>  -> '::' <Прог> 'ВСЕ'; <Прог>
<Продолжение1> -> '::' <Прог> <КонецДЛЯ>; <Прог>
<КонецДЛЯ>           -> 'ВСЕ'
<Условие>            -> <Выр> <Сравнение> <Выр>
<Сравнение>          -> '>=' | '<=' | '/=' | = | > | '<'
<РазРаза>            -> 'РАЗ' | 'РАЗА'
<Выражение>          -> <Term> <Expr>
<Expr>               -> -<Выр> | +<Выр> | #
<FG>                 -> <Factor> <G>
<Term>               -> <FG>
<G>                  -> *<FG> | /<FG> | '//'<FG> | #
<Factor> -> -<Factor>|(<Выр>) | <Число> |<Ид3>
<Число>              -> 'цел'
<Присваивание>       -> 'ид'
<Идентификатор1>     -> 'ид'
<Идентификатор2>     -> 'ид'
```

```
<Идентификатор3>     -> 'ид'


; $0 <Идентификатор1> Контроль дубликатов идентификаторов
; $1 <Идентификатор2> Контроль определения идентификатора
                      переменной цикла ДЛЯ
; $2 <Идентификатор2> Контроль пересечений переменных циклов ДЛЯ
; $1 <Идентификатор3> Контроль определения переменной в
                      выражении
; $3 <Число>          Контроль значения константы в диапазоне
                      двух байтов
; $4 <КонецДЛЯ>       Контроль завершения цикла ДЛЯ
; $1 <Присваивание>   Контроль определения переменной
                      присваивания
; $5 <Присваивание>   Контроль изменения переменной цикла ДЛЯ
```

Таким образом, данная программа может быть применена для трансляции самых различных языков программирования без изменения исходного кода, поскольку разработанный транслятор автоматически выполняет лексический и синтаксический анализ лишь при помощи описания *LL*(1)-грамматики. Только в случае обработки каких-либо семантических условий необходимо написать собственные функции для их реализации – в заданной работе были реализованы следующие функции:

$0 – контроль дубликатных переменных при их определении;

$1 – контроль определения переменной;

$2 – контроль пересечения переменных цикла ДЛЯ;

$3 – контроль диапазона целочисленной константы;

$4 – удаление переменной цикла ДЛЯ по его завершению;

$5 – контроль попытки изменения переменной цикла ДЛЯ.

Пользователь данной программы без проблем может добавить в грамматику описание, например, оператора печати языка Рапира, и данный транслятор без какой-либо переделки будет его транслировать.

4. РАЗРАБОТКА АРХИТЕКТУРЫ СИСТЕМЫ И АЛГОРИТМОВ

Программа состоит из следующих основных частей:

- контроль описания грамматики с точки зрения корректности записи синтаксических правил без учета принадлежности грамматики к классу *LL*(1)-грамматк;

- построение таблиц синтаксического разбора с одновременным контролем принадлежности грамматики к классу *LL*(1)-грамматик;

- выполнение лексического разбора программы на основании исходной грамматики языка;

- выполнение синтаксического разбора по автоматически построенной таблице разбора (пользователь НЕ ДОЛЖЕН сам создавать эту таблицу ручным образом!) с учетом семантических ограничений.

Рассмотрим методику работы (описанную в [2]) с грамматиками, который проверяет, является ли данная грамматика *LL*(1)-грамматикой, и определяет множества выбора, обеспечивающие построение нисходящего МП-распознавателя. Такая методика более простая для программной реализации по сравнению с алгоритмом, описанном анализе технического задания, потому что использует в основном матричные преобразования.

Полагаем, что все лишние нетерминалы в исходной грамматике предварительно удалены. Алгоритм содержит следующие шаги:

1. Нахождение аннулирущих нетерминалов и правил

Данная процедура включает в себя:

а) вычеркивание всех правил, правые части которых содержат хотя бы один терминальный символ (см. таблицу);

Исходная грамматика	Преобразование	Результат
1.<A>→a<B> 2.<B>→<D><C> 3.<B>→b 4.<C>→c<D> 5.<D>→a 6.<D>→**ε**	2.<B>→<D><C> 6.<D>→**ε**	6.<D>→**ε**

б) в оставшейся части определение непродуктивных (мертвых) нетерминалов

(выделены красным цветом);
в) удаление правил, содержащих мертвые нетерминалы;
Оставшиеся нетерминалы и правила будут аннулирущими.

2. Построение отношений НАЧИНАЕТСЯ-ПРЯМО-С (НПС)

Эти отношения определяются на множестве символов грамматики и означают: если применить к нетерминалу <A> точно одно правило и возможно, заменив в нем некоторые аннулирующие нетерминалы на **ε**, то можно получить цепочку, начинающуюся, например, с <B>. В этом случае пишут <A>НПС<B>. Тогда множество отношений НПС можно представить таблично. Запишем **<A> НПС a**, применив правило 1 и т.д.

3. Вычисление отношений НАЧИНАЕТСЯ-С (НС)

В общем случае матрица отношения НС определяется рефлексивно-транзитивным замыканием матрицы НПС. Пишут <A>НС<B> тогда и только тогда, когда существует цепочка, начинающаяся с <B>, которую можно вывести из <A>. Признак рефлексивности определяется тем, что любой символ всегда начинается с самого себя (звездочки на главной диагонали). Транзитивные отношения характеризуются всеми возможными подстановками при осуществлении вывода.

НПС

	A	B	C	D	a	b	c
A					1		
B			1	1		1	
C							1
D					1		
a							
b							
c							

НС

	A	B	C	D	a	b	c
A	*				1		
B		*	1	1	*	1	*
C			*				1
D				*	1		
a					*		

b						*	
c							*

ПП

	A	B	C	D	a	b	c
A							
B							
C							
D			1				
a		1					
b							
c				1			

Для рассмотренного примера и нетерминала <B> получим: **<B> НС <B>; <B> НС <C>** (после применения правила 6); **<B> НС <D>** (непосредственно); **<B> НС a** (после подстановки правила 5 в правило 2); **<B> НС b** (непосредственно из 3); **<B> НС c** (после подстановки правил 6 и 4 в правило 2). Полученный результат удобно представить в виде таблицы НС (матрицы).

Формальный алгоритм транзитивного замыкания включает в себя следующие шаги:

1) для каждого единичного элемента a_{ij} исходной матрицы в строку i дописать единичные элементы из строки j;

2) п.1 повторять сверху вниз и слева направо для всех элементов матрицы.

В качестве примера приведем работу алгоритма для единичного элемента **aвс**. Для этого элемента необходимо в строку В дописать единичные элементы из строки С (один элемент – звездочка).

4. Вычисление множества ПЕРВ для каждого нетерминала

Множество ПЕРВ(<A>) – множество всех терминалов, с которых могут начинаться цепочки, выводные из <A>. Для нашего примера ПЕРВ(<A>) = {a}, ПЕРВ(<B>) = {b}, ПЕРВ(<C>) = {c}, ПЕРВ(<D>) = {a} .

5. Вычисление множества ПЕРВ для каждого правила

В нашем случае ПЕРВ (1) ={a}, ПЕРВ (2) = ПЕРВ (<D>) U ПЕРВ (<C>) = {a}+{c} = {a,c}, ПЕРВ (3) ={b}, ПЕРВ (4) ={c}, ПЕРВ (5) ={a}, ПЕРВ (6) ={ }.

6. Построение отношений ПРЯМО - ПЕРЕД (ПП)

Эти отношения определяются следующим образом: мы пишем A ПП B тогда и только тогда, когда существуют правила вида: <D> → <A> <B> , <D> → <A> β <B> , где <A>, <B> - НТ символы; β – аннулирующая цепочка. Для всех правил и всех символов может быть составлена таблица ПП, отражающая это состояние. Для нашего примера: <D> ПП <C>; a ПП <B>; c ПП <D>.

7. Вычисление отношений ПРЯМО - НА - КОНЦЕ (ПНК)

Отношение определяется следующим образом: если <A> и <B> - символы грамматики, то <A> ПНК <B> тогда и только тогда, когда есть правила вида <B> →α <A>, <B> →α <A> β, где β – аннулирующая цепочка, α – произвольная цепочка. Это отношение является инверсией отношения НАЧИНАЕТСЯ–ПРЯМО С (НПС), если правую часть правила рассматривать "справа - налево".

ПНК

	A	B	C	D	a	b	c
A							
B	1						
C		1					
D			1				
a				1			
b		1					
c			1				

НК

	A	B	C	D	a	b	c
A	*						
B	1	*					
C	*	1	*				
D	*	*	1	*			
a	*	*	*	1	*		
b	*	1				*	

c	*	*	1				*

НК*ПП

	A	B	C	D	a	b	c
A							
B							
C							
D			*				
a		*	*				
b							
c				*			

Для нашего примера: <B> ПНК <A>; <C> ПНК <B>; <D> ПНК <C>; b ПНК <B>; c ПНК < C> (после применения правила 6 к <D> правила 4).

8. Вычисление отношений НА-КОНЦЕ (НК)

Мы пишем <A> НК <B> тогда и только тогда, когда из <B> можно вывести цепочку, заканчивающуюся символом <A>. В нашем примере: <B> НК <A> (непосредственно из правила 1); <C> НК <B> (непосредственно из правила 2); <C> НК <A> (после применения правила 2 к <B> правила 1) и т.д.

В общем случае матрица отношения НК определяется рефлексивно-транзитивным замыканием матрицы ПНК.

9. Вычисление отношений ПЕРЕД (П)

Мы пишем <A> П <B> тогда и только тогда, когда из начального символа можно вывести цепочку, в которой за вхождением <A> сразу же следует вхождение <B>.

П

	A	B	C	D	a	b	c
A							
B							
C							
D			*				*
a		*	*	*	*	*	*
b							

c				*	*		

$П^{+}$

	A	B	C	D	a	b	c	┤
A								1
B								1
C								1
D			1				1	1
a		1	1	1	1	1	1	1
b								1
c				1	1			1

Можно записать без вывода, что отношение П является произведением отношений НК, ПП и НС:

$$П = НК * ПП * НС .$$

В таблице НК*ПП приведен промежуточный результат произведения НК*ПП, а в таблице П – окончательный результат.

В нашем примере: <D> П <C> (непосредственно из правила 2); <D> П c (после применения правила 4 к <C> правила 2) и т.д. Формальный алгоритм произведения матриц отношений включает в себя следующие шаги:

1) для каждого единичного элемента a_{ij} матрицы-множимого в строку i результирующей матрицы дописать единичные элементы из строки j матрицы - множителя;

2) п.1 повторять сверху вниз и слева направо для всех элементов матрицы-множимого.

10. Расширение отношений ПЕРЕД, включив концевой маркер

Это отношение соответствует тем символам, которые находятся НА КОНЦЕ начального нетерминала для всех выводимых из него цепочек. В нашем случае: содержимое первого столбца матрицы **НК** (все символы - перед ┤).

11. Вычисление множеств СЛЕД для каждого аннулирующего нетерминала

Это легко выполнить, если определена матрица отношений ПЕРЕД. Тогда эти множества будут содержать все терминальные символы, для

которых аннулирующие нетерминалы расположены ПЕРЕД. В нашем случае: СЛЕД (< D >) = { c, ┤ } .

12. Вычисление множества ВЫБОР

Эти множества можно получить из уже вычисленных значений множеств ПЕРВ и СЛЕД.

ВЫБОР (1) = ПЕРВ (1) = {a}.
ВЫБОР (2) = ПЕРВ (2) = {a, c}.
ВЫБОР (3) = ПЕРВ (3) = {b}.
ВЫБОР (4) = ПЕРВ (4) = {c}.
ВЫБОР (5) = ПЕРВ (5) = {a}.
ВЫБОР (6) = СЛЕД (< D >) = {c, ┤}.

Множества выбора правил, имеющих в левой части один и тот же нетерминал, отсутствуют, следовательно, данная грамматика является *LL*(1)-грамматикой. Такой же вывод можно было сделать, если бы правила с одинаковыми левыми частями присутствовали, но в этом случае необходимо, чтобы множества выбора для этих правил не пересекались. На базе множеств ВЫБОР правил строится управляющая таблица МПА:

Управляющая таблица

	A	b	c	┤
A	#1	ОТВ	ОТВ	ОТВ
B	#2	#3	#2	ОТВ
C	ОТВ	ОТВ	#4	ОТВ
D	#3	ОТВ	#5	#5
∇	ОТВ	ОТВ	ОТВ	ДОП

Работа синтаксического блока:

1. Начальное состояние магазина – нетерминал <S>.
2. Автомат считывает очередную лексему и верхний символ из магазина.
3. Автомат обращается к управляющей таблице по двум значениям и получает управляющее правило – выход таблицы.
4. Автомат действует согласно управляющему правилу (в таблице правила пронумерованы), допускает или отвергает цепочку.

Семантика управляющих правил такова:

1. ЗАМЕНИТЬ(<B>), СДВИГ.
2. ЗАМЕНИТЬ(<C><D>), ДЕРЖАТЬ.
3. ВЫТОЛКНУТЬ, СДВИГ.

4. ЗАМЕНИТЬ(<D>), СДВИГ.
5. ВЫТОЛКНУТЬ, ДЕРЖАТЬ.

Действие ЗАМЕНИТЬ заставляет автомат заменить нетерминал вершины стека заменить на указанную в скобках цепочку символов. Действие ВЫТОЛКНУТЬ уничножает символ вершины стека. Действие СДВИГ выполняет чтение очередной лексемы, действие ДЕРЖАТЬ не изменяет текущую лексему. Действие ОТВ говорит об ошибочной цепочке, ДОП – о корректной цепочке. В таблице не указаны действия для случая, когда на вершине стека находится терминал – в этом случае отвергаются все цепочки, если текущая лексема не есть тот же самый терминал. Если совпадение терминалов есть, выполняется действие ВЫТОЛКНУТЬ,СДВИГ.

5. ТЕСТИРОВАНИЕ

Для тестирования транслятора можно использовать как встроенные примеры, так и создавать свои грамматики. Все примеры содержат безошибочные программы, которые при желании можно изменить с целью внесения в них ошибок. Продемонстрируем этот процесс на примере 6, который содержит грамматику и тестовую программу для языка Рапира.

Окно с грамматикой языка Рапира выглядит следующим образом:

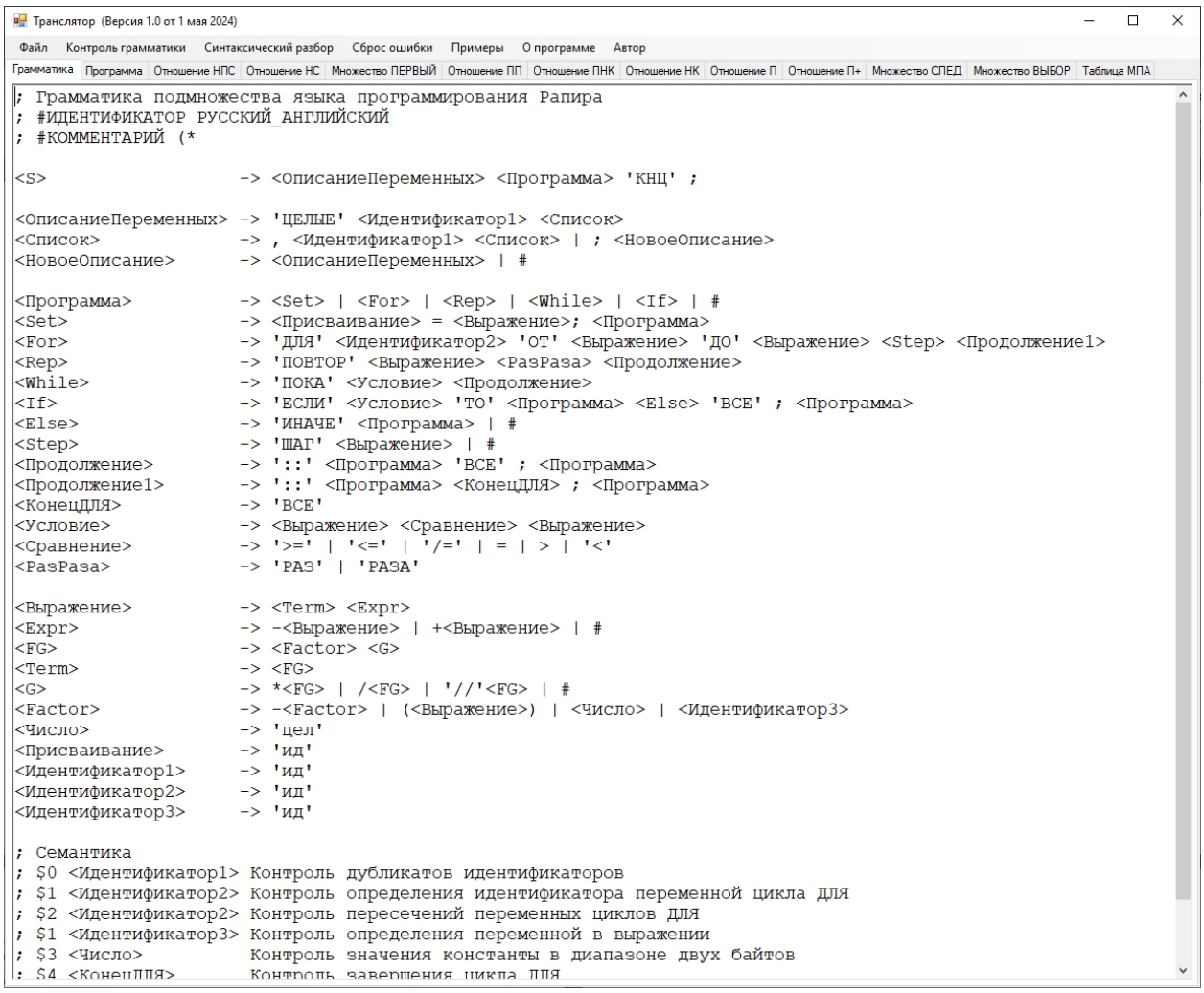


Рис. 2 Грамматика подмножества языка Рапира

Окно с программой на языке Рапира представлено на следующем рисунке:

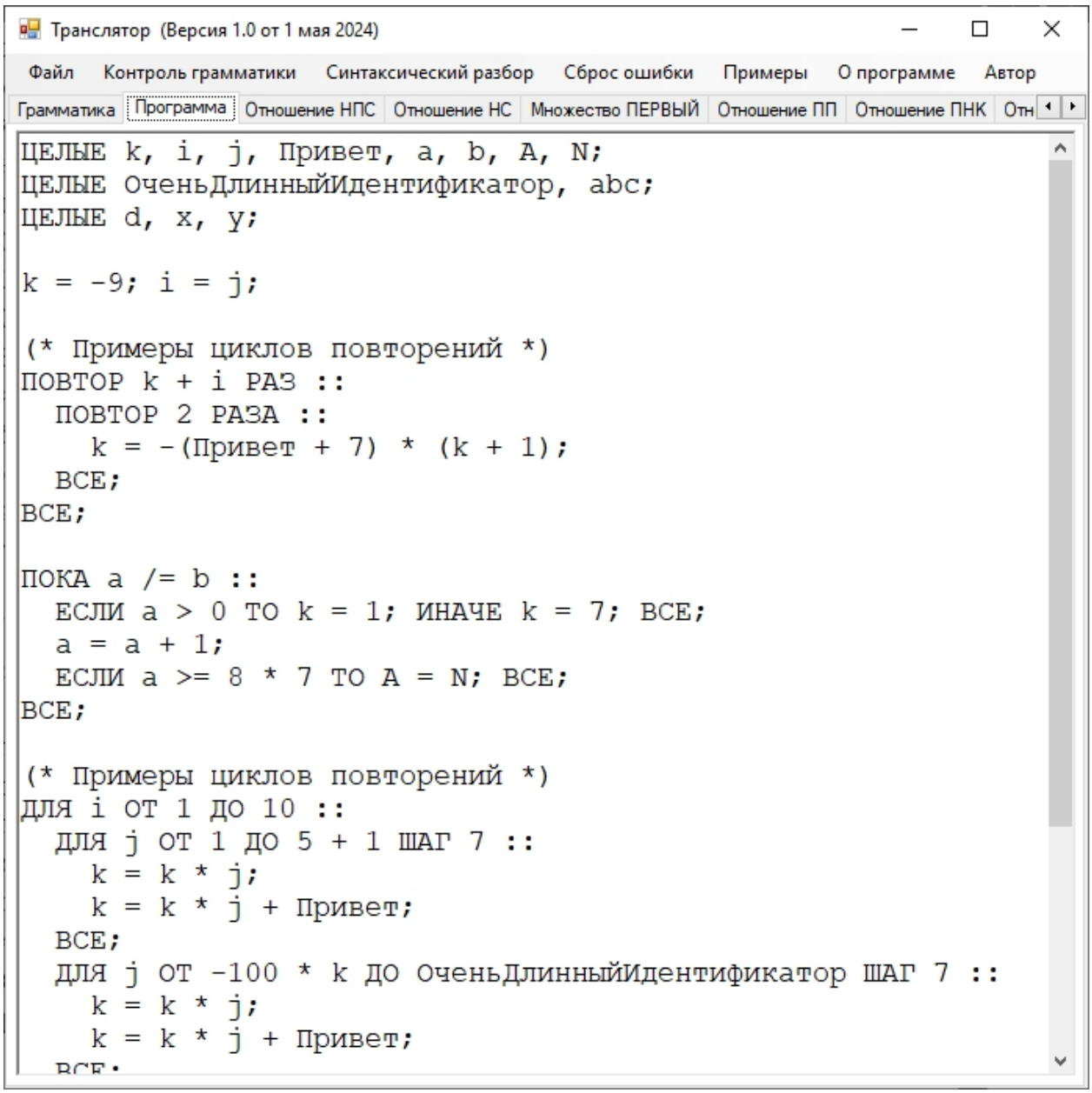


Рис. 3 Программа для подмножества языка Рапира

Тестируем ошибку дубликатного идентифмикатора:

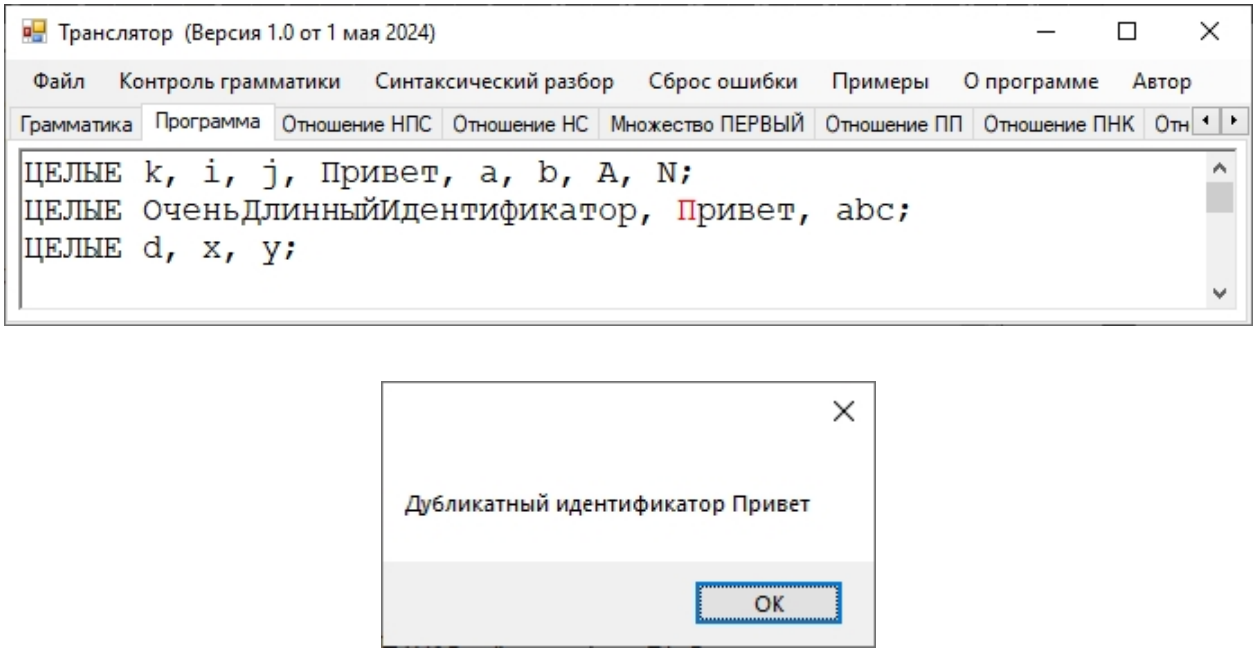


Рис. 4 Тестирование программы

Тестируем ошибку неопределенного идентификатора:

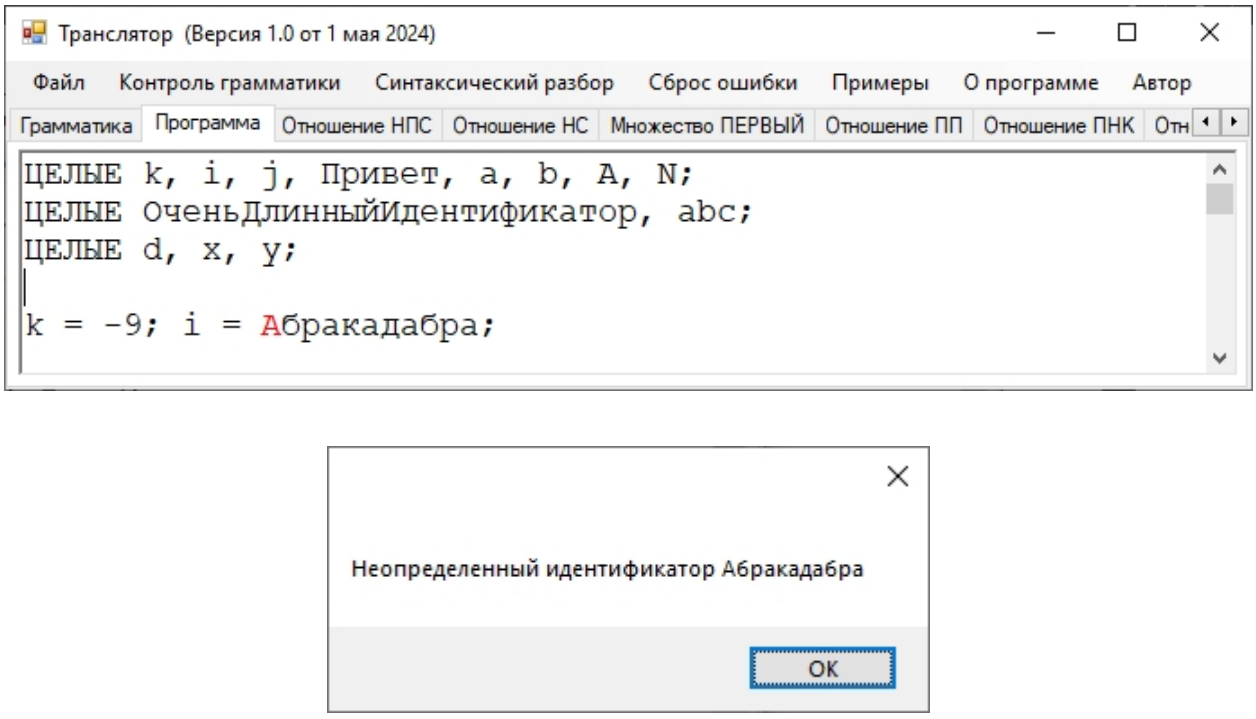


Рис. 5 Тестирование программы

Тестируем ошибку слишком большого числа:

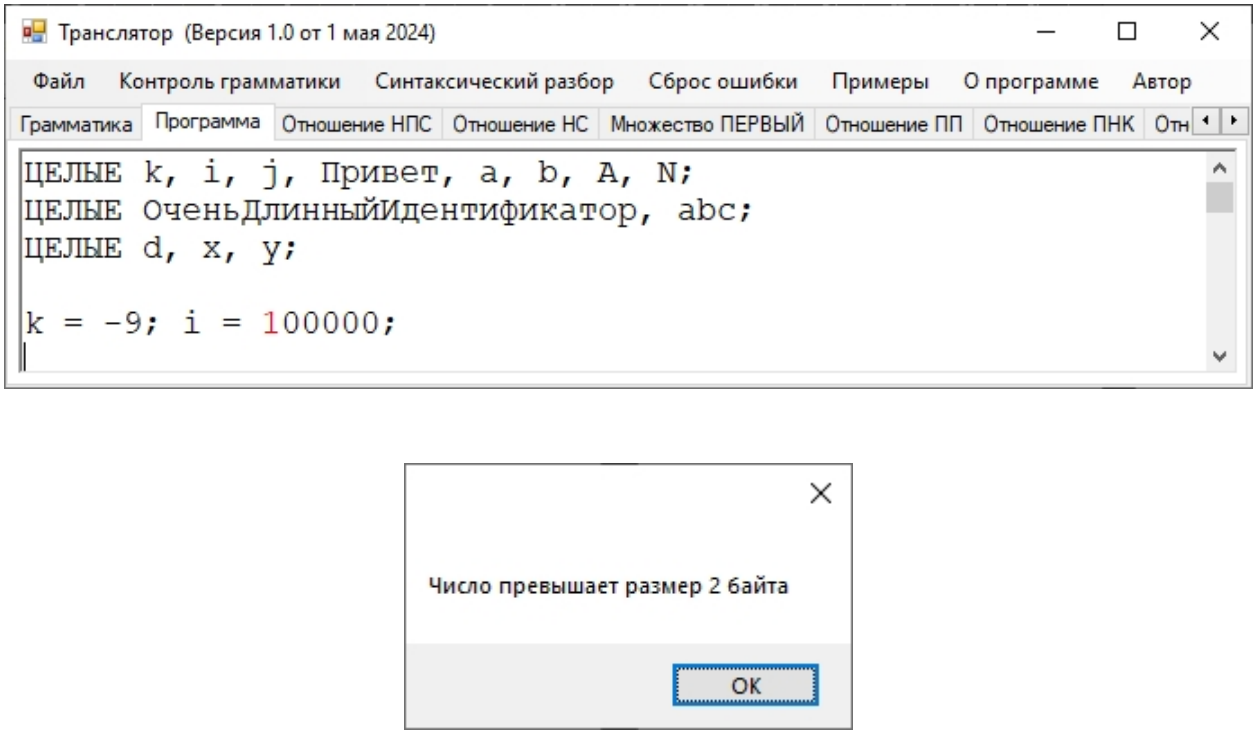


Рис. 6 Тестирование программы

Тестируем ошибку изменения переменной цикла:

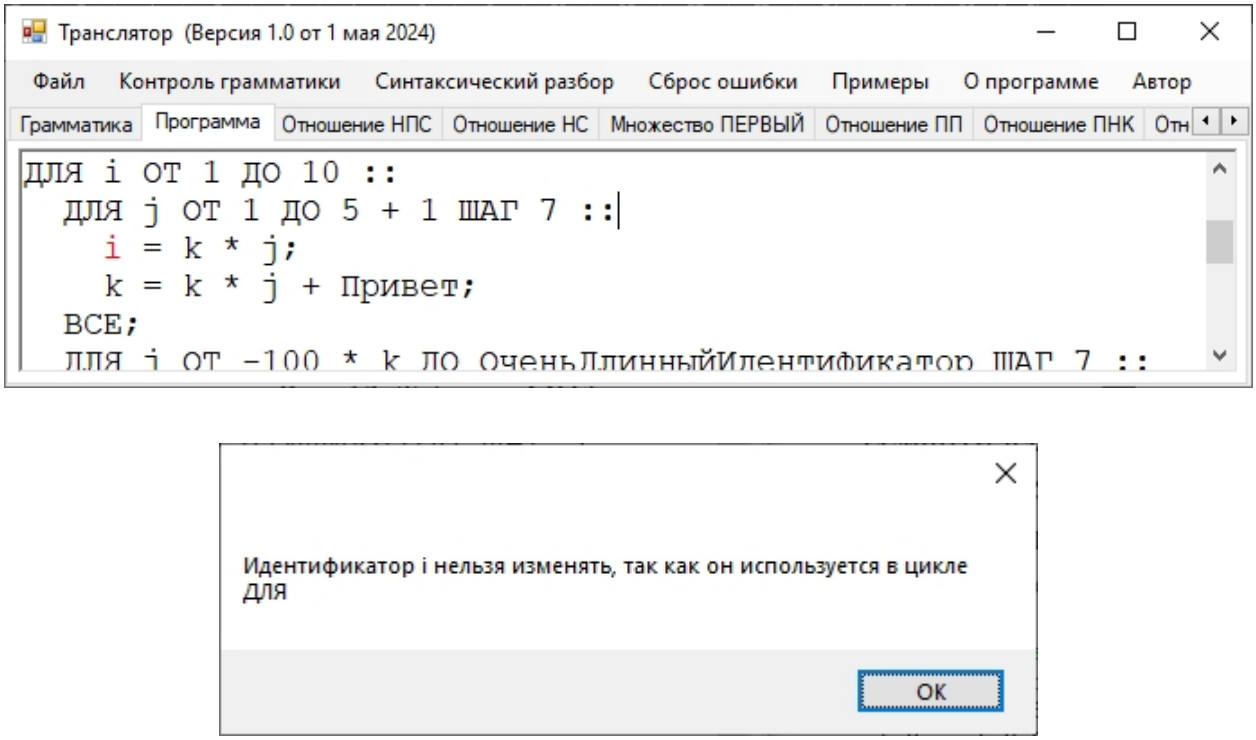


Рис. 7 Тестирование программы

Тестируем ошибку пересечения переменных цикла ДЛЯ:

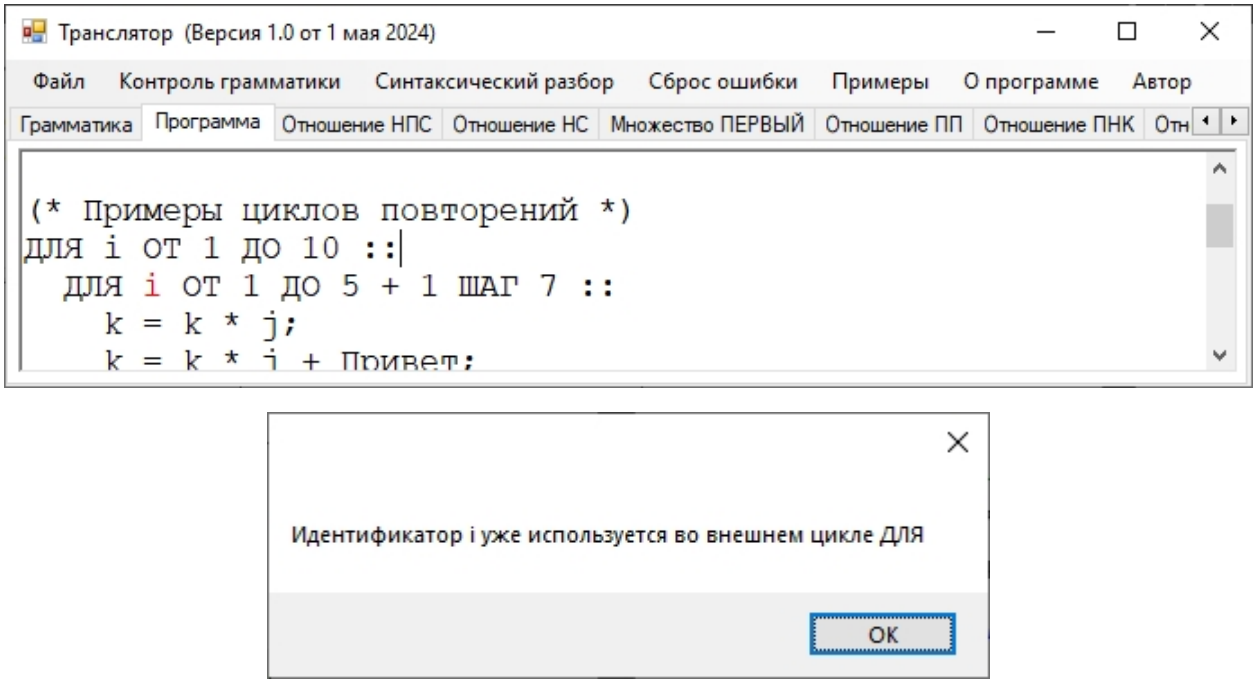


Рис. 8 Тестирование программы

Тестируем синтаксическую ошибку:

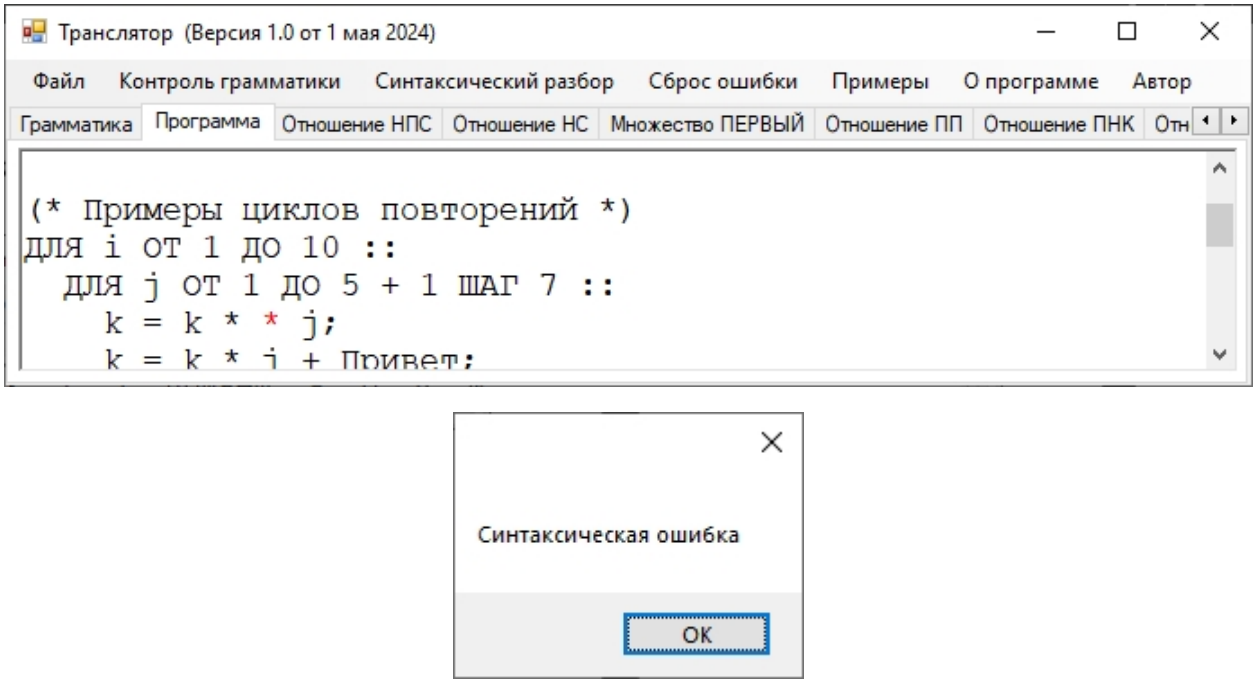


Рис. 9 Тестирование программы

Тестируем лексическую ошибку:

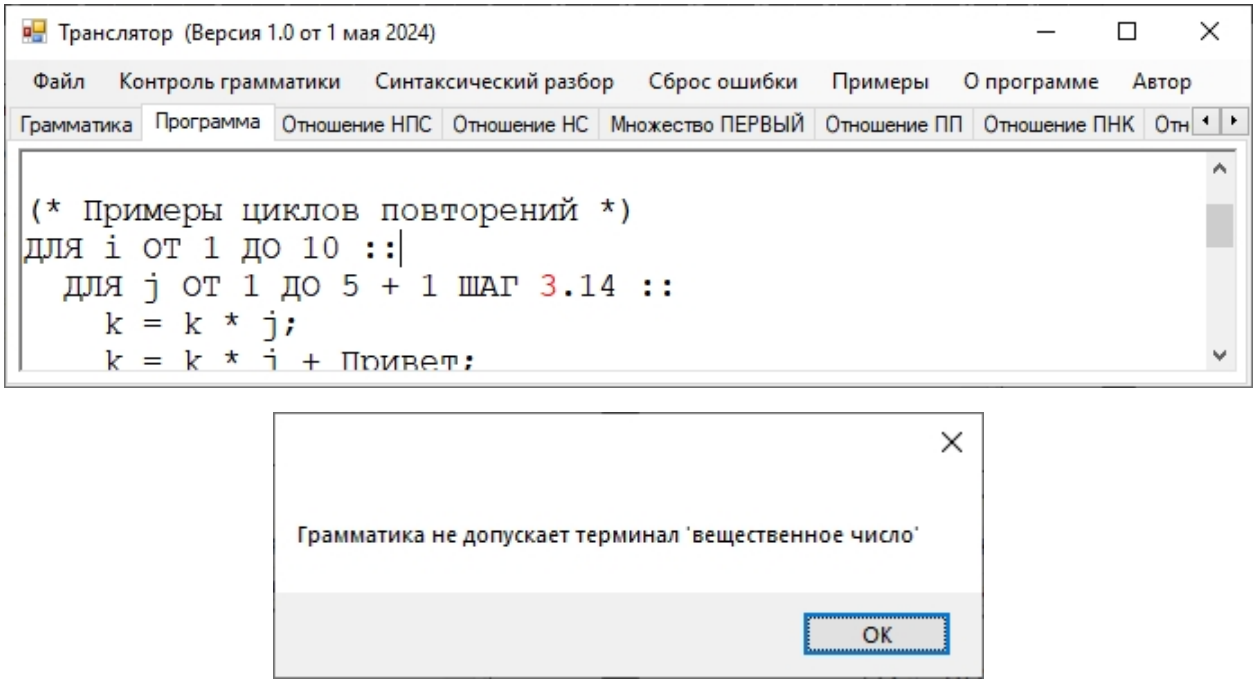


Рис. 10 Тестирование программы

Тестируем лексическую ошибку:

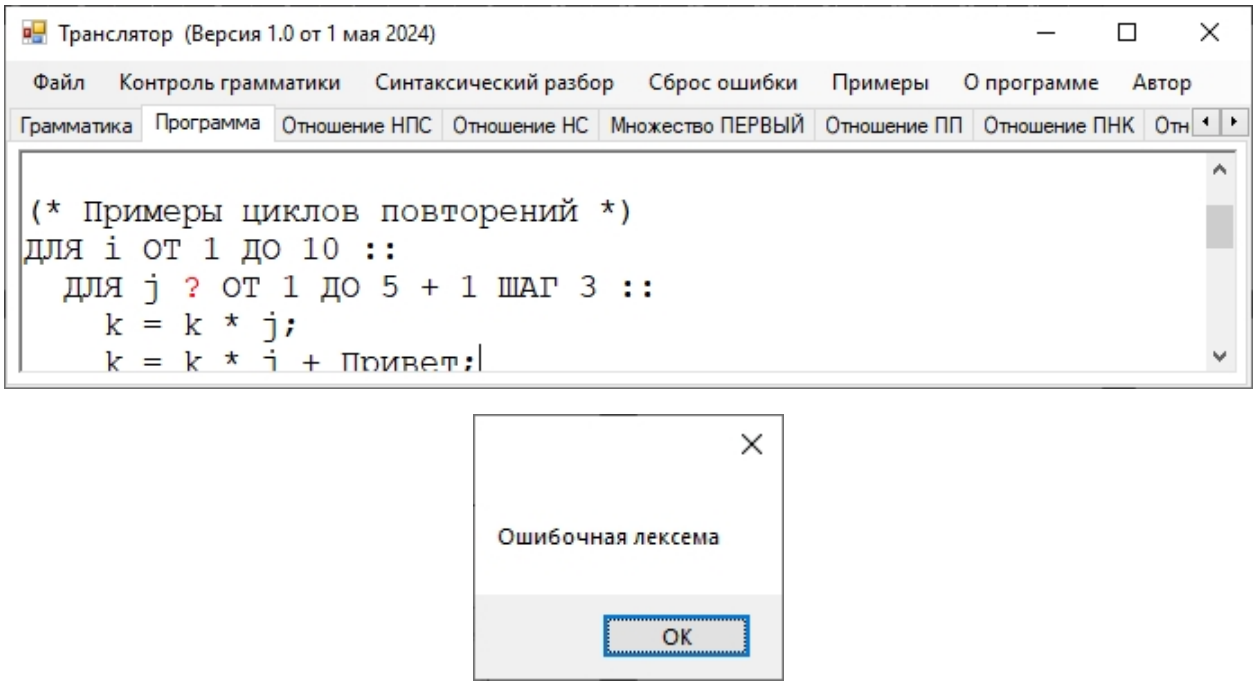


Рис. 11 Тестирование программы

Тестируем лексическую ошибку:

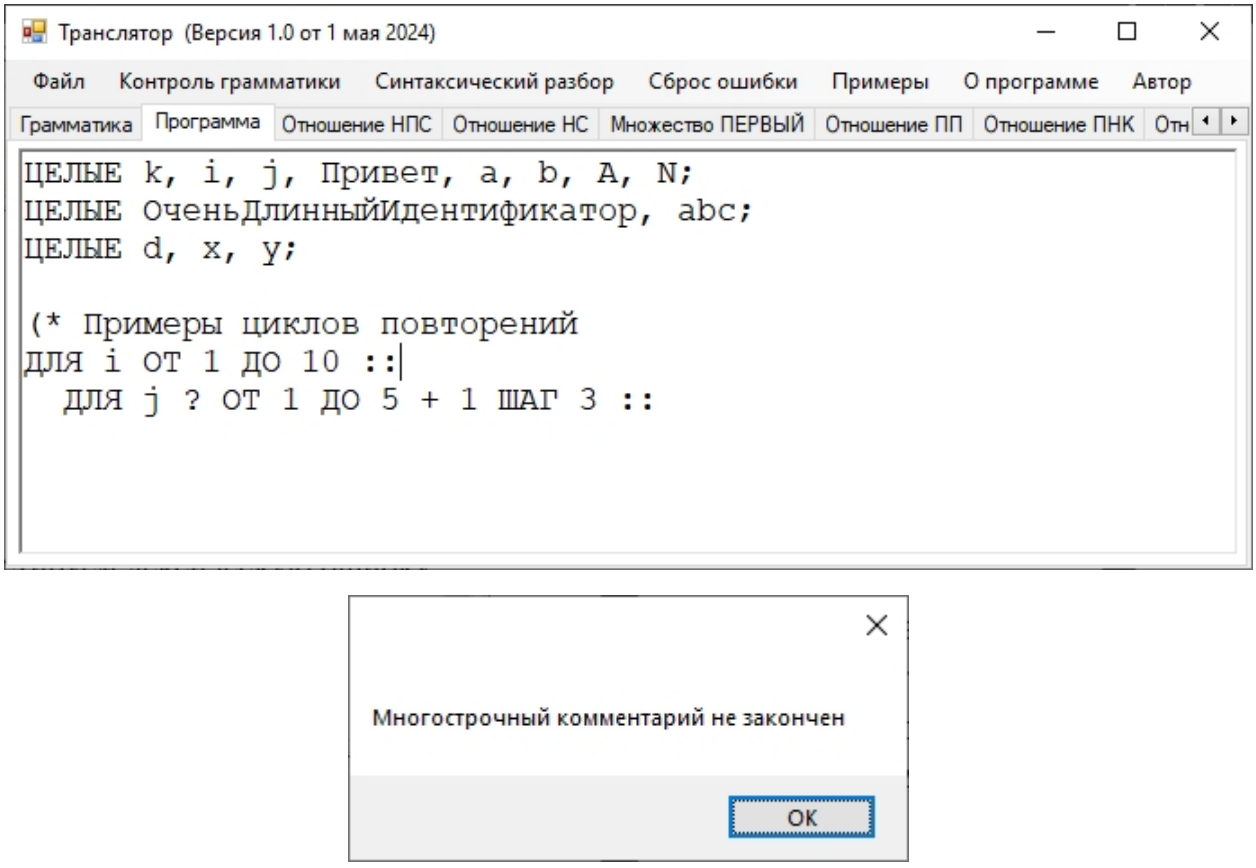


Рис.12 Тестирование программы

Тестируем распознавание *LL*(1)-грамматик:

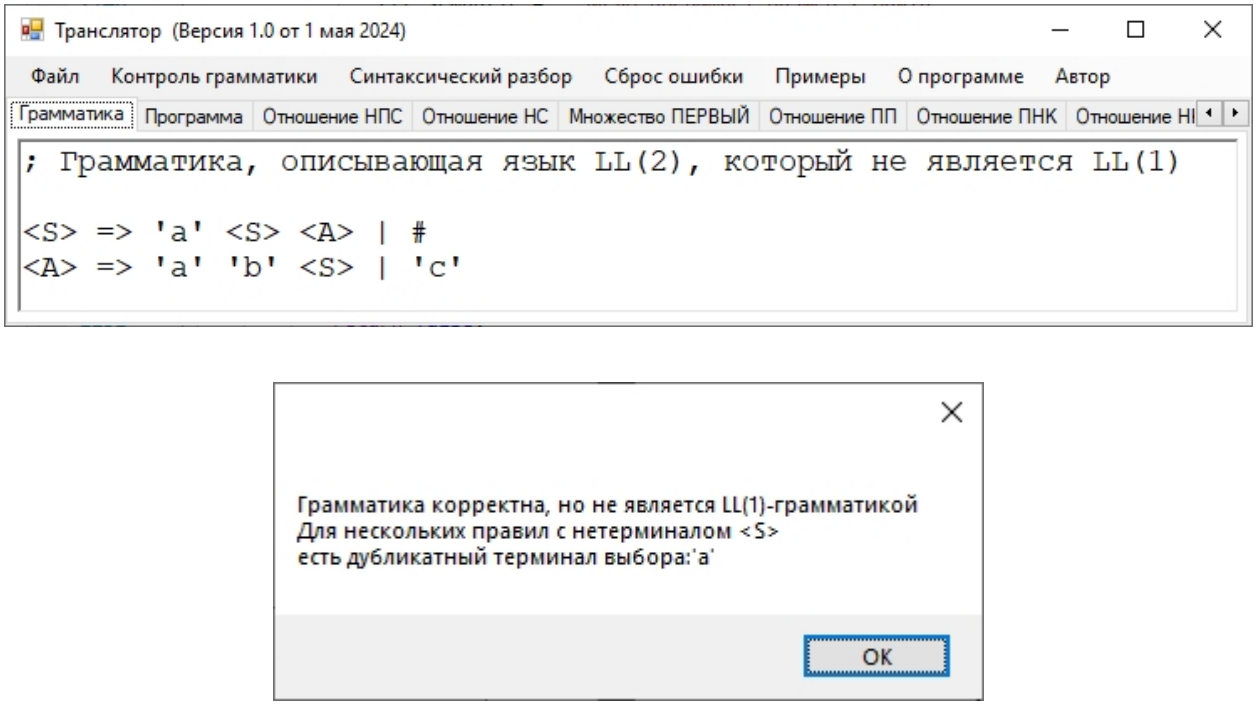


Рис. 13 Тестирование программы

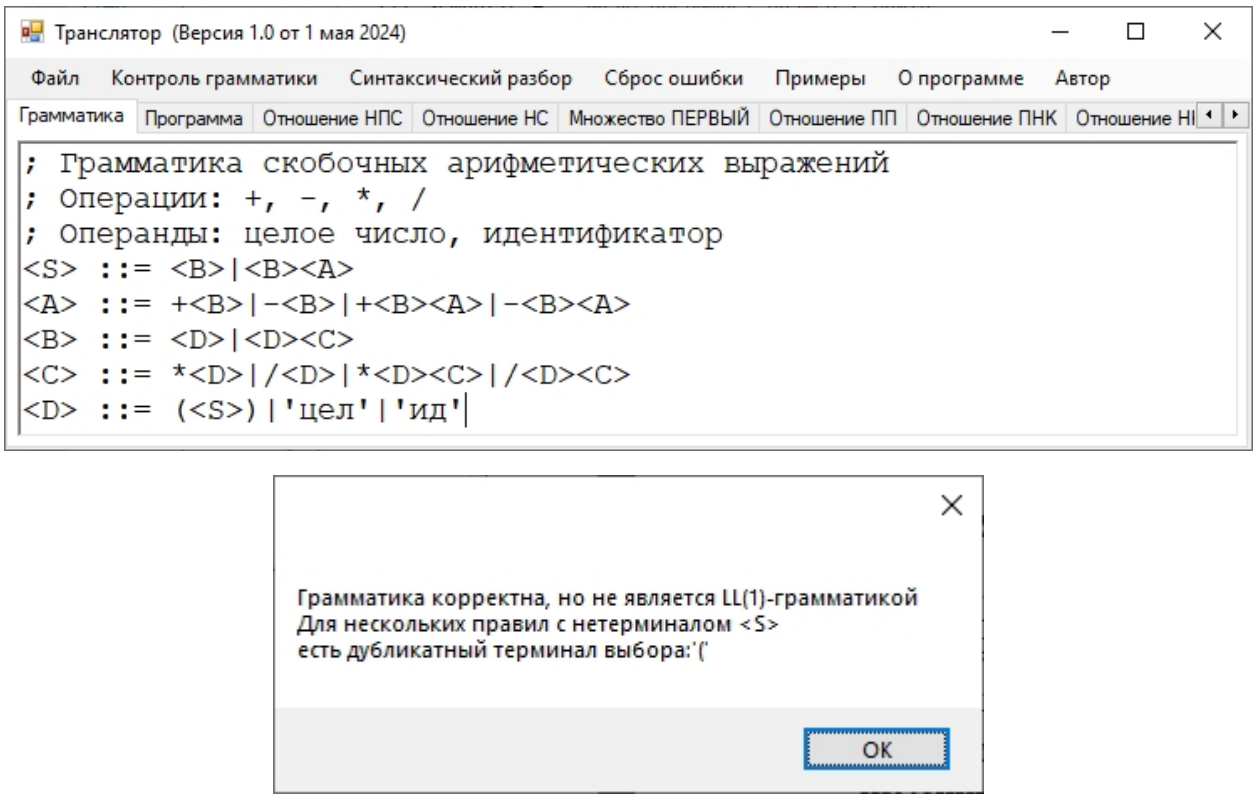


Рис. 14 Тестирование программы

6. РУКОВОДСТВО ПРОГРАММИСТА

Программирование транслятора выполнялось при помощи компилятора C# 2019. Программа довольно подробно задокументирования, поэтому при необходимости добавления в нее собственного функционала обработки семантических условий не должно быть каких-то сложностей. Достаточно изменить следующие фрагменты программы (в распоряжении программиста текущие лексемы будут расположены в переменных *ident, integer_num, double_num*):

```
// Поиск номера строки таблицы действий для нетерминала,
// находящегося на вершине стека; кроме того выполняется возможная семантика
str0 = str1.Substring(1);
str0 = str0.Substring(0, str0.Length - 1);

// Есть для этого нетерминала семантика?
for (j = 0; j < cnt_semantic && semantic[j] != str0; ++j)
    ;

// Да, есть - выполняем семантику
if (j != cnt_semantic)
{
    for (k = 0; k < actions[j, 0]; ++k)
    {
        switch (actions[j, k + 1])
        {
            case 0:
                if (Action0())
                {
                    ОшибкаТрансляции(err_semantic);
                    return;
                }
                continue;

            case 1:
                if (Action1())
                {
                    ОшибкаТрансляции(err_semantic);
                    return;
                }
                continue;

        . . . . . .

            case 5:
                if (Action5())
                {
                    ОшибкаТрансляции(err_semantic);
                    return;
                }
                continue;
            }
        }
    }
}
```

```
        . . . . . .

// Контроль дубликатов идентификаторов
// В случае ошибки возвращается true
private bool Action0()
{
    string str = ident.Length > 8 ? ident.Substring(0, 8) : ident;

    // Такой идентификатор уже определен?
    if (table.Contains(str))
    {
                err_semantic = "*Дубликатный идентификатор " + ident;
                return true;
            }

            // Добавляем идентификатор в таблицу идентификаторов
            table.Add(str);
            return false;
        }

        // Контроль определения идентификатора переменной
        // В случае ошибки возвращается true
        private bool Action1()
        {
            string str = ident.Length > 8 ? ident.Substring(0, 8) : ident;

            // Такой идентификатор не определен?
            if (!table.Contains(str))
            {
                err_semantic = "*Неопределенный идентификатор " + ident;
                return true;
            }

            return false;
        }

        // Контроль пересечений переменных циклов ДЛЯ
        // В случае ошибки возвращается true
        private bool Action2()
        {
            string str = ident.Length > 8 ? ident.Substring(0, 8) : ident;

            // Такой идентификатор уже используется внешним циклом ДЛЯ?
            if (table_for.Contains(str))
            {
                err_semantic = "*Идентификатор " + ident +
                               " уже используется во внешнем цикле ДЛЯ";
                return true;
            }

            // Добавляем идентификатор в таблицу переменных циклов
            table_for.Add(str);
            return false;
        }
```

```
// Контроль значения константы в диапазоне двух байтов
// В случае ошибки возвращается true
private bool Action3()
{
    if (integer_num > 32767)
    {
        err_semantic = "*Число превышает размер 2 байта";
        return true;
    }
    return false;
}

// Контроль завершения циклов ДЛЯ для удаления переменной цикла
// Всегда возвращается false
private bool Action4()
{
    table_for.RemoveAt(table_for.Count - 1);
    return false;
}

// Контроль изменения переменной цикла ДЛЯ
// В случае ошибки возвращается true
private bool Action5()
{
    string str = ident.Length > 8 ? ident.Substring(0, 8) : ident;

    // Такой идентификатор уже используется внешним циклом ДЛЯ?
    if (table_for.Contains(str))
    {
        err_semantic = "*Идентификатор " + ident +
                       " нельзя изменять, так как он используется в цикле ДЛЯ";
        return true;
    }
    return false;
}
```

7. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа содержит дружественный интерфейс, который позволит пользователю на встроенных примерах выполнить тестирование программы или даже создать свою собственную грамматику и без проблем транслировать учебные программы. Многочисленные вкладки демонстрируют построение таблиц синтаксического разбора.

По какой-либо ошибке позиция символа окрашивается в красный цвет, который перед редактированием текста необходимо сбросить по соответствующему пункту меню.

Как грамматику, так и транслируемую программу можно загружать, сохранять или редактировать при помощи традиционного оконного интерфейса.

Программа трансляции *LL*(1)-грамматик представляет собой окно следующего вида:

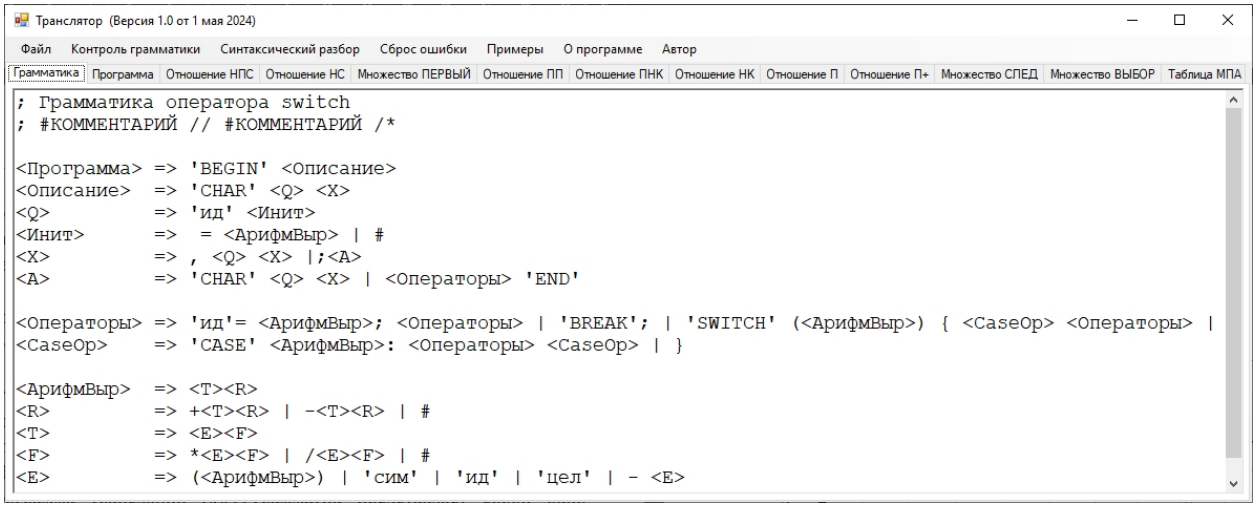


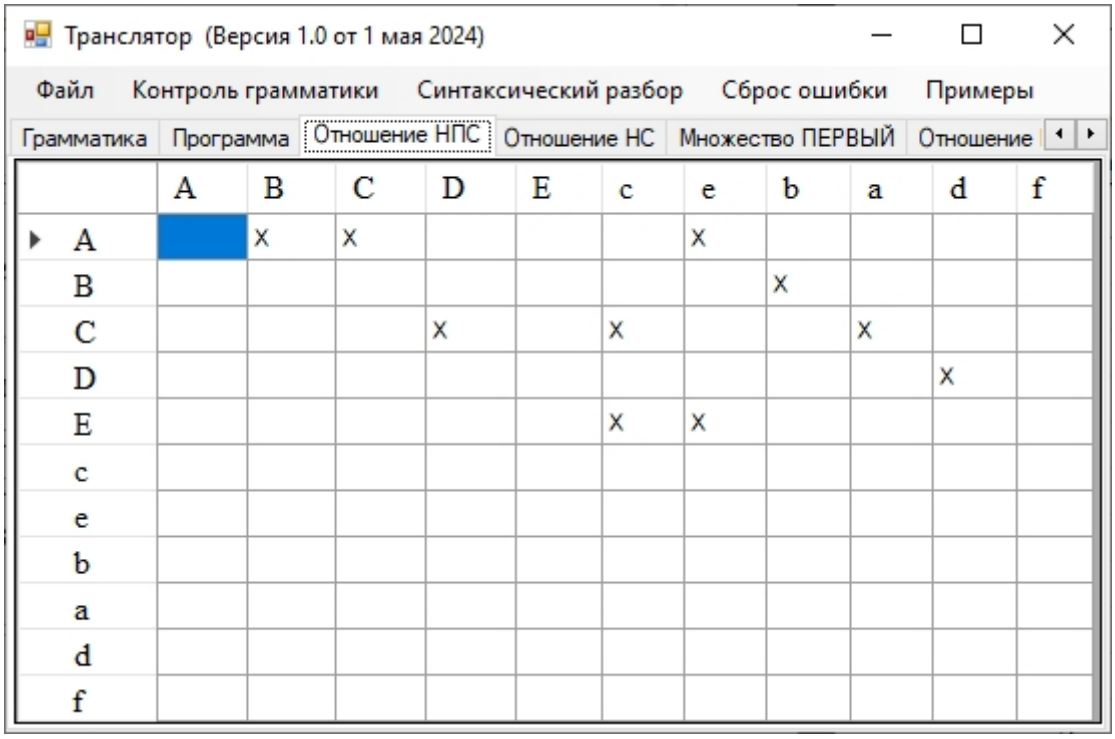
Рис. 15 Интерфейс программы

Программа представляет все свои данные в виде закладок, на которых можно ввести текст грамматики и исходную программу, а также увидеть весь процесс разбора грамматики в виде описанных ранее таблиц. В программе имеются встроенные примеры грамматик и программ.

Например, для следующей грамматики из встроенных примеров

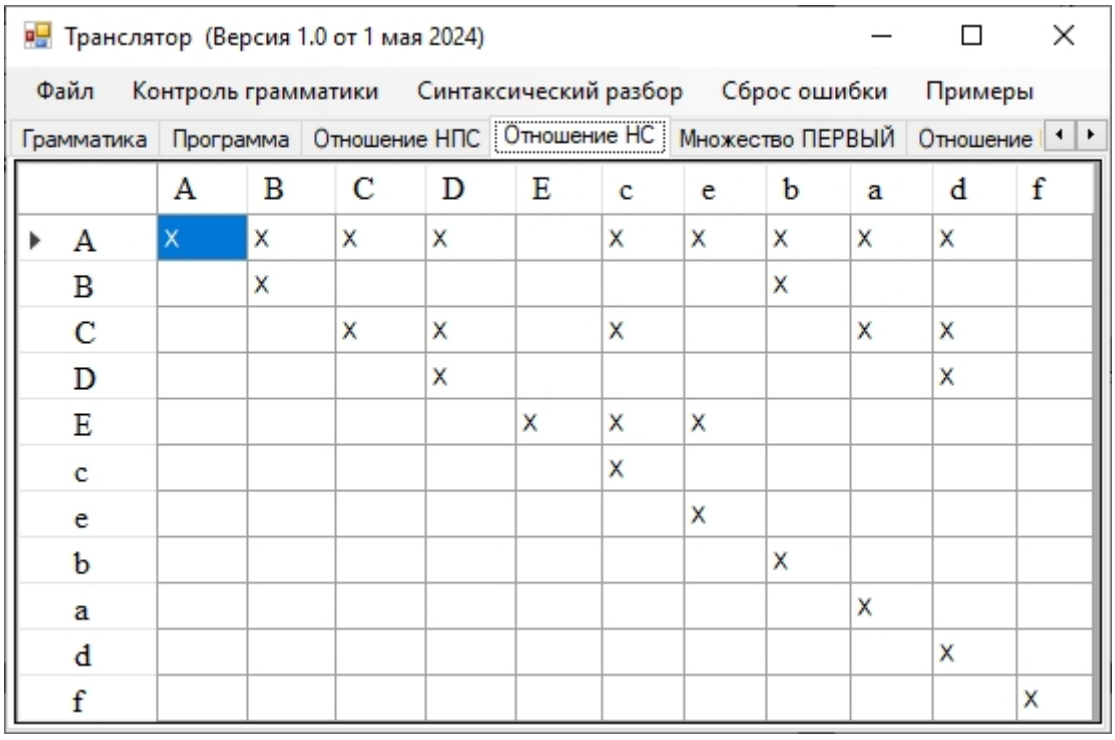
```
; Грамматика из книги Льюис Ф.
;"Теоретические основы проектирования компиляторв", стр. 284
<A> => <B> <C> 'c' | 'e' <D> <B>
<B> => # | 'b' <C> <D> <E>
<C> => <D> 'a' <B> | 'c' 'a'
<D> => # | 'd' <D>
<E> => 'e' <A> 'f' | 'c'
```

программа представит следующие таблицы:



	A	B	C	D	E	c	e	b	a	d	f
A		X	X				X				
B								X			
C				X		X			X		
D										X	
E						X	X				
c											
e											
b											
a											
d											
f											

Рис. 16 Отношение НПС



	A	B	C	D	E	c	e	b	a	d	f
A	X	X	X	X		X	X	X	X	X	
B		X						X			
C			X	X		X			X	X	
D				X						X	
E					X	X	X				
c						X					
e							X				
b								X			
a									X		
d										X	
f											X

Рис. 17 Отношение НС

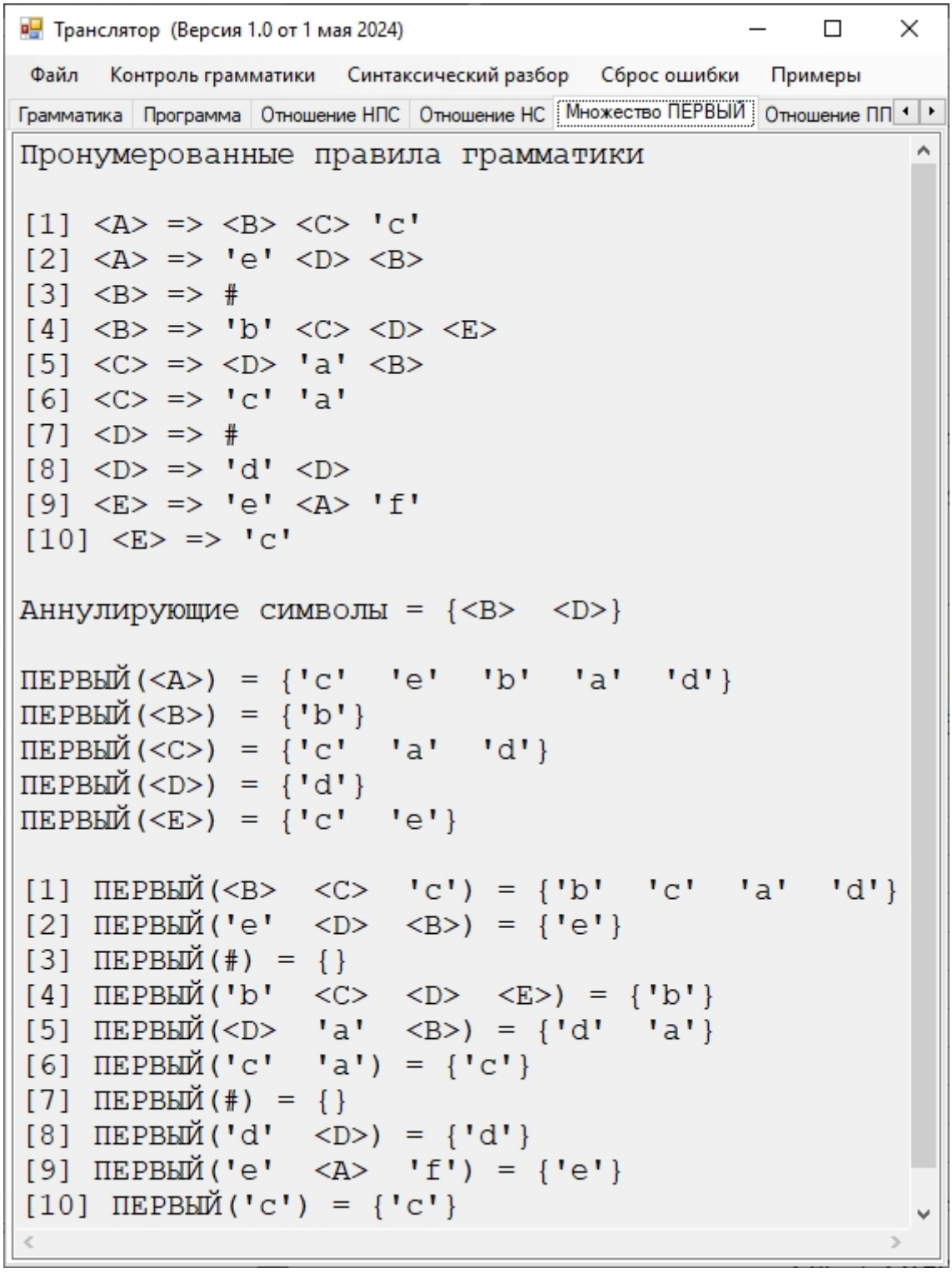
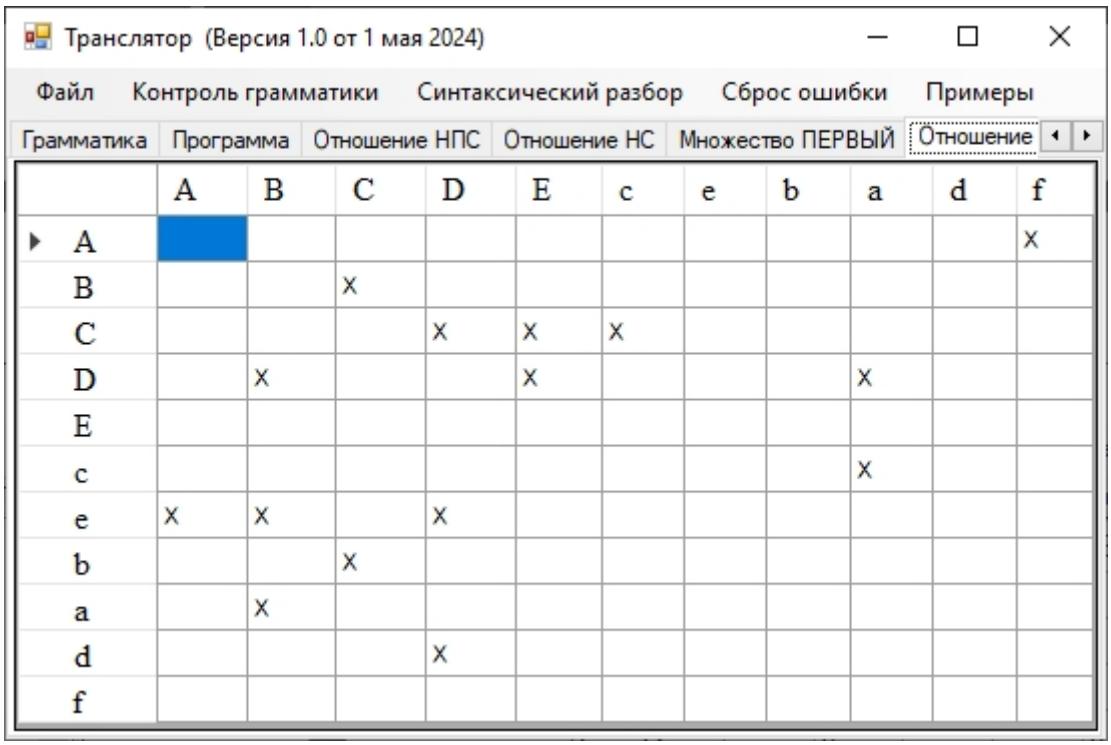
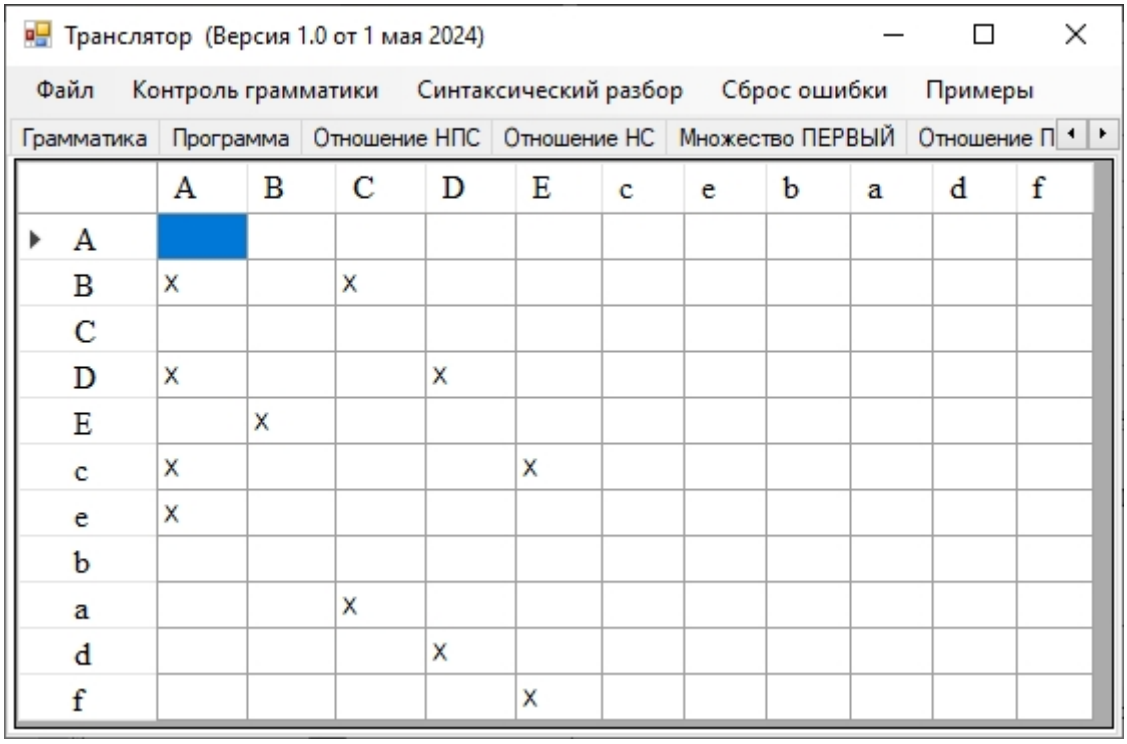


Рис. 18 Множество ПЕРВЫЙ



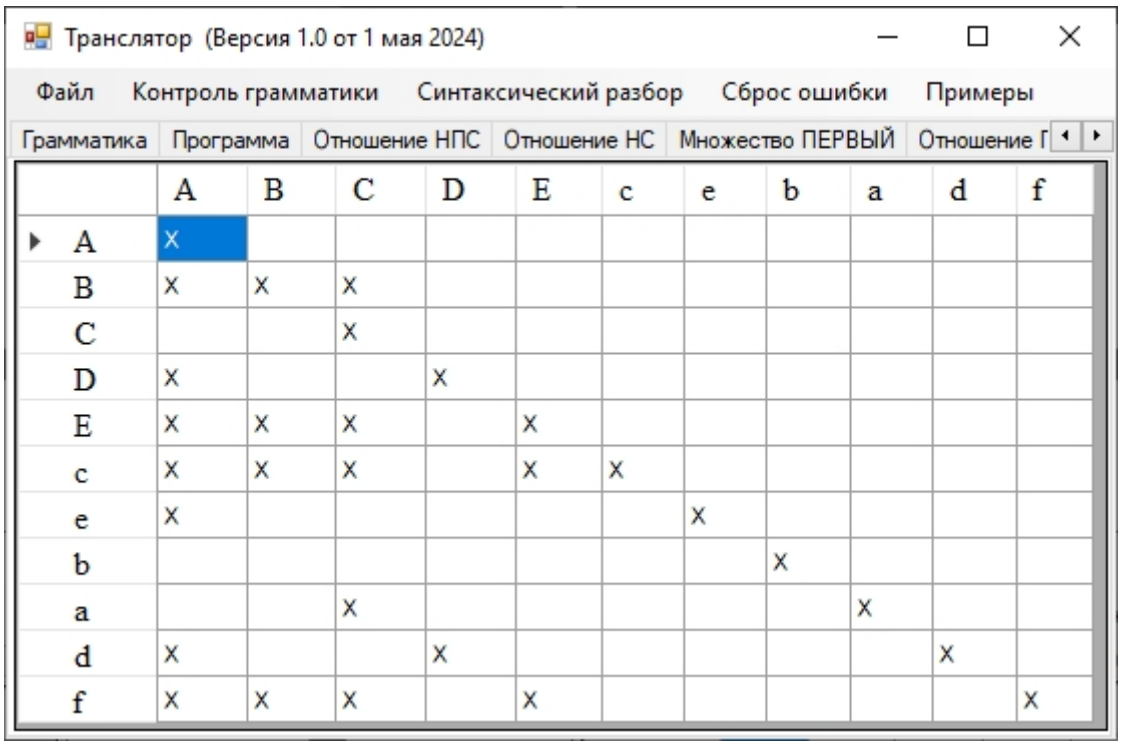
	A	B	C	D	E	c	e	b	a	d	f
A											X
B			X								
C				X	X	X					
D		X			X				X		
E											
c									X		
e	X	X		X							
b			X								
a		X									
d				X							
f											

Рис. 19 Отношение ПП



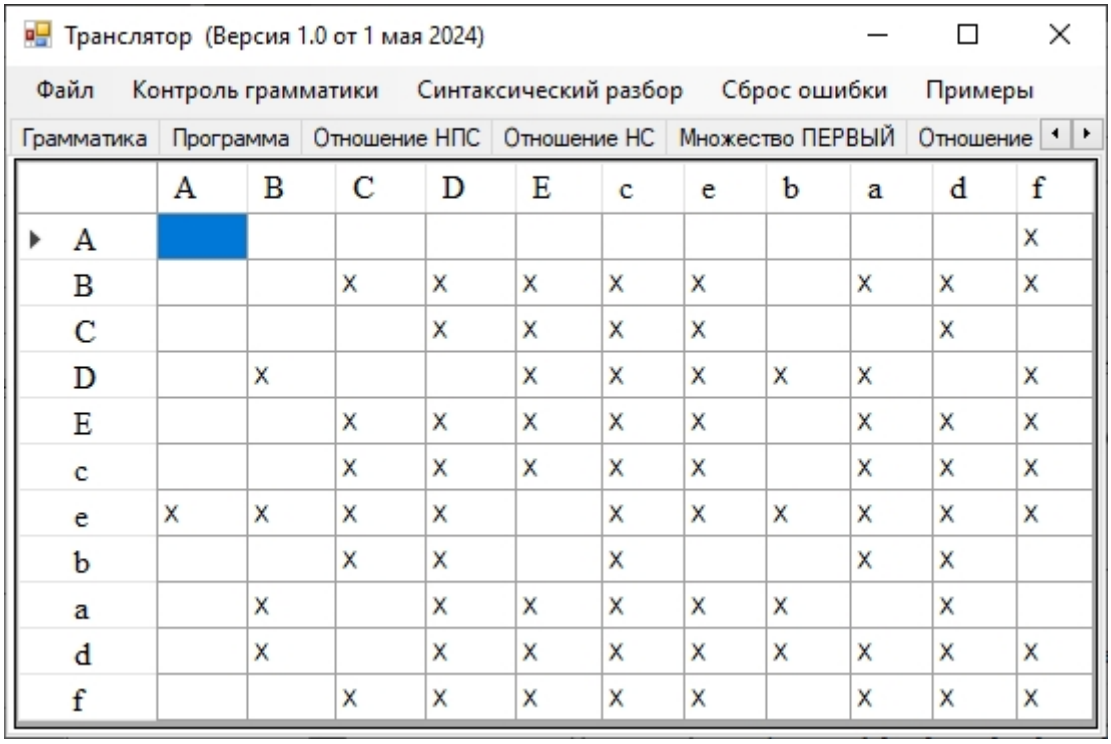
	A	B	C	D	E	c	e	b	a	d	f
A											
B	X		X								
C											
D	X			X							
E		X									
c	X				X						
e	X										
b											
a			X								
d				X							
f					X						

Рис. 20 Отношение ПНК



	A	B	C	D	E	c	e	b	a	d	f
A	X										
B	X	X	X								
C			X								
D	X			X							
E	X	X	X		X						
c	X	X	X		X	X					
e	X						X				
b								X			
a			X						X		
d	X			X						X	
f	X	X	X		X						X

Рис. 21 Отношение НК



	A	B	C	D	E	c	e	b	a	d	f
A											X
B			X	X	X	X	X		X	X	X
C				X	X	X	X			X	
D		X			X	X	X	X	X		X
E			X	X	X	X	X		X	X	X
c			X	X	X	X	X		X	X	X
e	X	X	X	X		X	X	X	X	X	X
b			X	X		X			X	X	
a		X		X	X	X	X	X		X	
d		X		X	X	X	X	X	X	X	X
f			X	X	X	X	X		X	X	X

Рис.229 Отношение П

	A	B	C	D	E	c	e	b	a	d	f	КОНЕЦ
A											x	x
B			x	x	x	x	x		x	x	x	x
C				x	x	x	x			x		
D		x			x	x	x	x	x		x	x
E			x	x	x	x	x		x	x	x	x
c			x	x	x	x	x		x	x	x	x
e	x	x	x	x		x	x	x	x	x	x	x
b			x	x		x			x	x		
a		x		x	x	x	x	x		x		
d		x		x	x	x	x	x	x	x	x	x
f			x	x	x	x	x		x	x	x	x

Рис. 23 Отношение П+

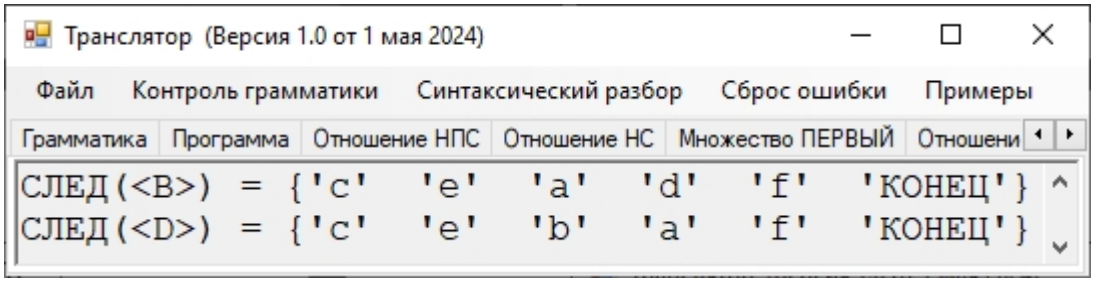


Рис. 24 Множество СЛЕД

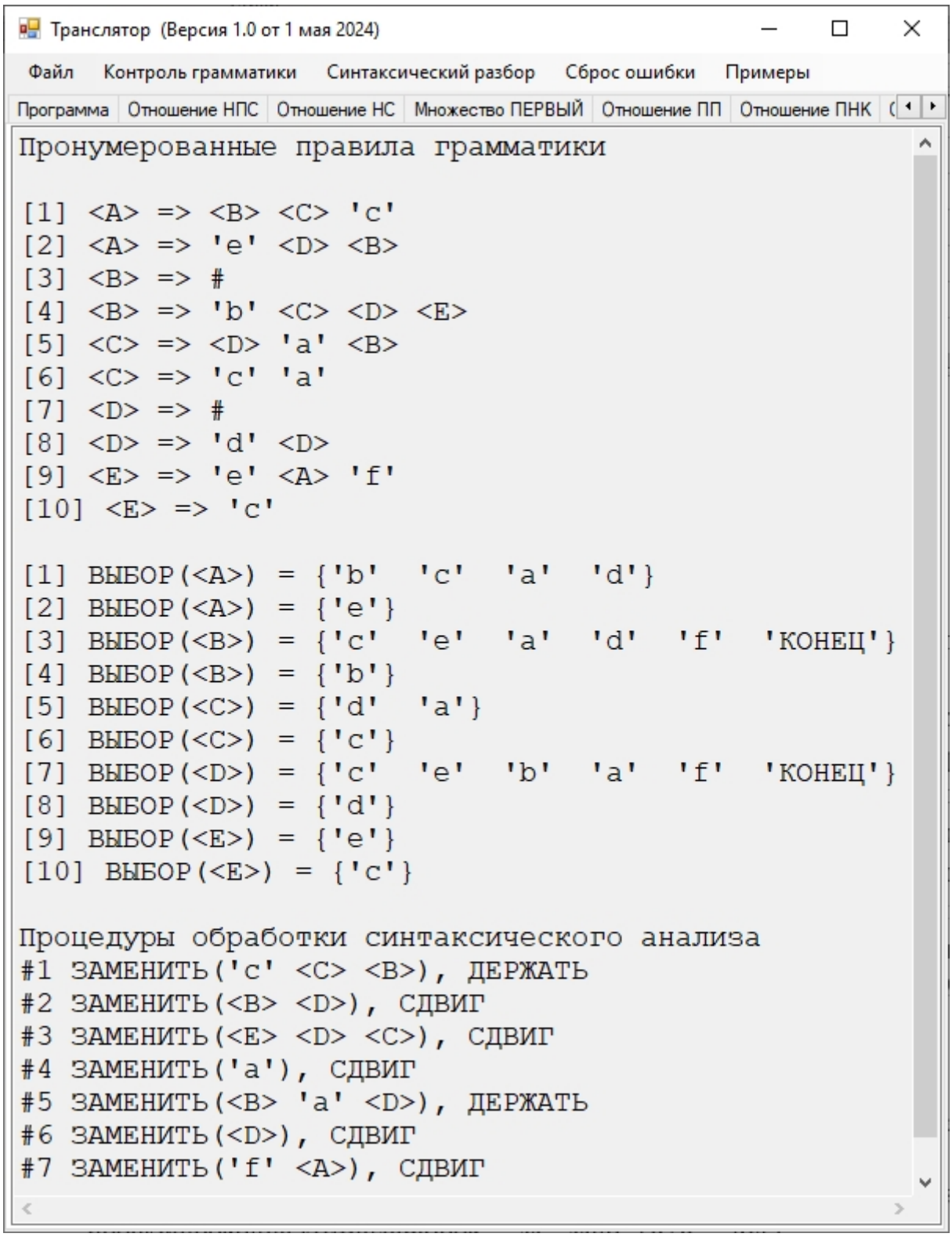


Рис. 25 Множество ВЫБОР

	c	e	b	a	d	f	КОНЕЦ
A	#1	#2	#1	#1	#1	ОТВ	ОТВ
B	ВЫТ,ДЕР	ВЫТ,ДЕР	#3	ВЫТ,ДЕР	ВЫТ,ДЕР	ВЫТ,ДЕР	ВЫТ,ДЕР
C	#4	ОТВ	ОТВ	#5	#5	ОТВ	ОТВ
D	ВЫТ,ДЕР	ВЫТ,ДЕР	ВЫТ,ДЕР	ВЫТ,ДЕР	#6	ВЫТ,ДЕР	ВЫТ,ДЕР
E	ВЫТ,СДВ	#7	ОТВ	ОТВ	ОТВ	ОТВ	ОТВ
НАЧАЛО	ОТВ	ОТВ	ОТВ	ОТВ	ОТВ	ОТВ	ДОП

Рис. 26 Таблица МПА

Именно таблица МПА является основой дальнейшего синтаксического разбора LL(1)-грамматик и она получается автоматически, достаточно лишь корректно составить такую грамматику, что, конечно не всегда просто сделать.

ЗАКЛЮЧЕНИЕ

Созданный программный проект полностью выполняет поставленное задание. Программа снабжена подробными комментариями, которые позволят без особого труда разобраться в ее работе. Интерфейс программы довольно дружественный и не требует дополнительного описания. Программа содержит пункт меню со справочной информацией и вполне может использоваться как рабочий инструмент при изучении дисциплин формальных грамматик и конструирования компиляторов. Дальнейшим развитием программы может быть добавление в нее функционала преобразования формальных грамматик, а также реализация альтернативного способа разбора *LL*(1)-грамматик при помощи множеств *FIRST* и *FOLLOW*.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Грис Д. Конструирование компиляторов для цифровых вычислительных машин. – М.: Мир, 1975. – 544 с.*

2. *Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. – М.: Мир, 1979. - 564 с.*

3. *Власенко А.В., Ключко В.И. Теория языков программирования и методы трансляции: учебное пособие для студентов специальности 220400. – Краснодар: Изд.КубГТУ, 2004. –120 с.*

ТЕКСТ ПРОГРАММЫ

```
using System;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Collections.Generic;

namespace Транслятор
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // Общий список определенных идентификаторов
        private List<string> table = new List<string>();

        // Список переменных циклов ДЛЯ
        private List<string> table_for = new List<string>();

        // Ошибка семантики
        private string err_semantic;

        // Прочитанный идентификатор
        private string ident;

        // Прочитанное целое число
        private int integer_num;

        // Прочитанное вещественное число
        private double double_num;

        // Максимальное число нетерминалов, для которых задана семантика
        private const int NOTERMINAL_SEMANTIC = 20;

        // Максимальное количество семантик для одного нетерминала
        private const int MAX_SEMANTIC = 10;

        // Список нетерминалов с семвнтикой
        private string[] semantic = new string[NOTERMINAL_SEMANTIC];

        // Набор семантик для каждого нетерминала (нулевой элемент
        // содержит количество семантик)
        private int[,] actions = new int[NOTERMINAL_SEMANTIC, MAX_SEMANTIC + 1];

        // Количество нетерминалов с семантикой
        private int cnt_semantic;

        private string     ТекущаяСтрокаГрамматики = "";
        private string[]   ЭпсилонНетерминалы;
        private string[][] МножествоСлед;
        private string[][] ПравилаГрамматики;
        private string[]   НетерминалыТерминалы;
        private int        КоличествоЭпсилонНетерминалов = 0;
        private int        НомерСтрокиГрамматики = 0;
        private int        НомерСимволаЭлементаГрамматики = 0;
        private int        НомерСимволаГрамматики = 0;
        private int        КоличествоПравилГрамматики = 0;
        private int        МаксДлинаПравила = 0;
```

```
private int[,]     matr, matrНК, matrПП, matrНС;
private int[,]     ПЕРВЫЙ;
private int        Высота;

// Тип комментария:
// [0] - //
// [1] - /* .... */
// [2] - (* ... *)
// [3] - {...}
private bool[]     type_comment = new bool[4];

// Тип идентификатора:
// РУССКИЙ            0
// АНГЛИЙСКИЙ         1
// РУССКИЙ_АНГЛИЙСКИЙ 2
private int type_ident;

// Терминалы в названии столбцов матрицы МПА
private string[]   ТерминалыМПА;

// Матрица действий для синтаксического анализа
string[,]          Действия;

// Набор терминалов множества ВЫБОР для всех правил грамматики
// Нулевой индекс sel[,0] содержит количество терминалов
// Далее идут индексы терминалов (КОНЕЦ есть индекс -1)
int[,]             sel;

// Признаки терминалов, которые будут обозначать колонки таблицы МПА
bool[]             mark;

// Все лексемы грамматики (за исключением целых и
// вещественных констант и идентификаторов)
readonly private string[] Лексемы = new string[200];

// Количество лексем грамматики
private int КоличествоЛексем;

// Очередной сканируемый символ
private int ind_line;

// Индекс начала очередной лексемы
private int lex_ind;

// Признак сканирования многострочного комментария
private bool yes_comment;

// Символы завершения многострочного комментария
private string end_comment;

// Очередная строка программы
private string src_line;

// Номер строки программы
private int record;

// Наличие в грамматике символьной константы
private bool ЕстьСимвол;

// Наличие в грамматике символьной строки
private bool ЕстьСтрока;
```

```
// Наличие в грамматике целого числа
private bool ЕстьЦелое;

// Наличие в грамматике вещественного числа
private bool ЕстьВещественное;

// Наличие в грамматике идентификатора
private bool ЕстьИдентификатор;

// Наличие корректной грамматики
private bool ХорошаяГрамматика = false;

// Поместить лексему в список лексем грамматики
private void СохранитьЛексему (string str)
{
    int i;

    str = str.Substring(2);
    for (i = 0; i < КоличествоЛексем; ++i)
    {
        if (Лексемы[i] == str)
            return;
    }
    Лексемы[КоличествоЛексем++] = str;
}

// Пузырьковая сортировка массива лексем по убыванию длин
private void СортировкаЛексем()
{
    int    i, j;
    string str;

    for (i = 0; i < КоличествоЛексем; ++i)
    {
        for (j = КоличествоЛексем - 1; j > i; --j)
        {
            if (Лексемы[j - 1].Length < Лексемы[j].Length)
            {
                str = Лексемы[j - 1];
                Лексемы[j - 1] = Лексемы[j];
                Лексемы[j] = str;
            }
        }
    }
}

// Проверка текста сканирование на принадлежность лексемам грамматики
private string ЭтоЛексема()
{
    int    i, len, len2;
    string str, str2;

    // Длина сканируемого текста до конца строки
    len = src_line.Length - lex_ind;

    // Взять текст с начала очередной лексемы
    str = src_line.Substring(lex_ind);

    // Пытаемся найти совпадение с лексемами грамматики
    // При этом гарантировано, что распознавание символов
    // <= произойдет раньше распознавания символа <,
    // так как лексемы грамматики отсортированы по убыванию длин лексем
```

```
    for (i = 0; i < КоличествоЛексем; ++i)
    {
        str2 = Лексемы[i];
        if ((len2 = str2.Length) > len)
            continue;
        if (str.Substring(0, len2) == str2)
            return str2;
    }

    // Лексему не нашли
    return "";
}

// Вернуть true, если идентификатор в грамматике определен как служебное слово
private bool ЭтоСлужебноеСлово(string str)
{
    int i;

    // Пытаемся найти совпадение с лексемами грамматики
    for (i = 0; i < КоличествоЛексем; ++i)
    {
        if (str == Лексемы[i])
            return true;
    }

    return false;
}

// Получить очередной элемент грамматики
// return: ""     - строка закончена
//         "*zzz" - строка ошибочна (zzz - сообщение об ошибке)
//         "Txxx" - терминал xxx
//         "Nyyy" - нетерминал yyy
//         "#"    - символ эпсилон
//         "|"    - разделитель правил правой части
private string ЭлементГрамматики()
{
    string str;
    char   c;

    // Игнорируем пустые символы
    while (НомерСимволаГрамматики != ТекущаяСтрокаГрамматики.Length)
    {
        if (!Char.IsWhiteSpace(ТекущаяСтрокаГрамматики[НомерСимволаГрамматики]))
            break;
        ++НомерСимволаГрамматики;
    }

    // Сохраним для возможной выдачи ошибки место в строке грамматики
    НомерСимволаЭлементаГрамматики = НомерСимволаГрамматики;

    // Конец строки?
    if (НомерСимволаГрамматики == ТекущаяСтрокаГрамматики.Length)
        return "";

    // Начало нетерминала?
    if ((c = ТекущаяСтрокаГрамматики[НомерСимволаГрамматики++]) == '<')
    {
        // Этот символ не должен быть последним в строке
        if (НомерСимволаГрамматики == ТекущаяСтрокаГрамматики.Length)
            return "*Неожиданный конец правила грамматики";
```

```
        // Два символа '<<' определяют терминал '<',
        // а не признак начала нетерминала
        if ((c = ТекущаяСтрокаГрамматики[НомерСимволаГрамматики++]) == '<')
        {
            СохранитьЛексему("T'<");
            return "T'<";
        }

        if (c == '>')
            return "*Пустой нетерминал";

        // Первый символ нетерминала
        str = "N" + c.ToString();

        // Собираем нетерминал до символа '>'
        //(он не должен содержать пустых символов)
        while (НомерСимволаГрамматики != ТекущаяСтрокаГрамматики.Length)
        {
            // Конец определения нетерминала?
            if ((c = ТекущаяСтрокаГрамматики[НомерСимволаГрамматики++]) == '>')
                return str;
            if (Char.IsWhiteSpace(c))
                break;
            str += c.ToString();
        }

        // Ошибочный нетерминал
        return "*Ошибочное определение нетерминала";
    }

    // Разделитель правил или пустой символ?
    if (c == '|' || c == '#')
        return c.ToString();

    // Начало специальных терминалов, заданных в апострофах:
    // 'цел'                    - целочисленная константа
    // 'вещ'                    - вещественная константа
    // 'сим'                    - символ в апострофах
    // 'ид'                     - переменная
    // 'if', 'while' и так далее - служебные слова языка
    if (c == '\'')
    {
        // Этот символ не должен быть последним в строке
        if (НомерСимволаГрамматики == ТекущаяСтрокаГрамматики.Length)
            return "*Неожиданный конец правила грамматики";

        // Два символа '' определяют терминал '
        if ((c = ТекущаяСтрокаГрамматики[НомерСимволаГрамматики++]) == '\'')
        {
            СохранитьЛексему("T''");
            return "T''";
        }

        // Собираем терминал
        str = "T'" + c.ToString();
        while (НомерСимволаГрамматики != ТекущаяСтрокаГрамматики.Length)
        {
            // Конец определения терминала?
            if ((c = ТекущаяСтрокаГрамматики[НомерСимволаГрамматики++]) == '\'')
            {
                // Определение наличия лексем символьных константант и строк,
                // целых и вещественных чисел, а также идентификаторов
```

```
                if (str == "T'сим")
                {
                    ЕстьСимвол = true;
                    return str;
                }
                if (str == "T'стр")
                {
                    ЕстьСтрока = true;
                    return str;
                }
                if (str == "T'цел")
                {
                    ЕстьЦелое = true;
                    return str;
                }
                if (str == "T'вещ")
                {
                    ЕстьВещественное = true;
                    return str;
                }
                if (str == "T'ид")
                {
                    ЕстьИдентификатор = true;
                    return str;
                }
                СохранитьЛексему(str);
                return str;
            }
            if (Char.IsWhiteSpace(c))
                break;
            str += c.ToString();
        }

        // Ошибочный терминал
        return "*Ошибочное определение терминала";
    }

    // Прочие терминалы не должны быть цифрами, латинскими буквами
    // или символом подчеркивания
    if (c == '_' || Char.IsLetterOrDigit(c))
        return "*Недопустимый символ грамматики";
    str = "T'" + c.ToString();
    СохранитьЛексему(str);
    return str;
}

// Контроль строки грамматики
//         ""     - строка безошибочна
//         "*zzz" - строка ошибочна (zzz - сообщение об ошибке)
private string КонтрольСтрокиГрамматики()
{
    string str, tmp;
    int    i, pos = 0;
    bool   эпсилон;

    // Первый элемент должен быть нетерминалом
    str = ЭлементГрамматики();
    if (str[0] == '*')
        return str;
    if (str[0] != 'N')
        return "*Ожидался нетерминал";
```

```
    // Далее ожидаются разделители левой и правой части правила
    // в виде: -> => ::=
    // Игнорируем пустые символы
    while (НомерСимволаГрамматики != ТекущаяСтрокаГрамматики.Length)
    {
        if (!Char.IsWhiteSpace(ТекущаяСтрокаГрамматики[НомерСимволаГрамматики]))
            break;
        ++НомерСимволаГрамматики;
    }

    // Сохраним для возможной выдачи ошибки место в строке грамматики
    НомерСимволаЭлементаГрамматики = НомерСимволаГрамматики;

    // Конец строки?
    if (НомерСимволаГрамматики == ТекущаяСтрокаГрамматики.Length)
        return "*Отсутствует правая часть правила";

    // Добавим в конец строки два пробела, чтобы корректно проверить
    // наличие разделителей правила в виде: -> => ::=
    ТекущаяСтрокаГрамматики += "  ";

    // Проверяем наличие разделителей правила
    if ((tmp = ТекущаяСтрокаГрамматики.Substring(НомерСимволаГрамматики, 2)) ==
          "->" || tmp == "=>")
        НомерСимволаГрамматики += 2;
    else if (ТекущаяСтрокаГрамматики.Substring(НомерСимволаГрамматики, 3) == "::=")
        НомерСимволаГрамматики += 3;
    else
        return "*Ошибочен разделитель левой и правой части правила грамматики";

    // Убираем добавленные пробелы
    ТекущаяСтрокаГрамматики = ТекущаяСтрокаГрамматики.Substring
                    (0, ТекущаяСтрокаГрамматики.Length - 2);

    // Контролируем все альтернативы правила
    for (;;)
    {
        i = 2;
        str = ЭлементГрамматики();
        if (str == "")
            return "*Отсутствует правая часть правила";
        if (str[0] == '*')
            return str;
        if (str == "|")
            return "*Ошибочное использование альтернативы";

        ++КоличествоПравилГрамматики;
        for (;;)
        {
            // Символ эпсилон должен быть единственным в правой части правила
            if (эпсилон = str == "#")
            {
                if (i != 2)
                    return "*Символ эпсилон должен быть единственным в правиле";
                pos = НомерСимволаЭлементаГрамматики;
            }

            str = ЭлементГрамматики();
            if (str == "" || str[0] == '*')
            {
                if (i > МаксДлинаПравила)
                    МаксДлинаПравила = i;
```

```
                return str;
            }

            if (эпсилон && str != "|")
            {
                НомерСимволаЭлементаГрамматики = pos;
                return "*Символ эпсилон должен быть единственным в правиле";
            }

            // Новая альтернатива?
            if (str == "|")
            {
                if (i > МаксДлинаПравила)
                    МаксДлинаПравила = i;
                break;
            }
            ++i;
        }
    }
}

// Сохранение правил грамматики
private void СохранениеПравилГрамматики()
{
    string   str, Нетерминал, tmp;
    string[] Альтернатива = new string[МаксДлинаПравила];
    int      i, j;

    // Порождающий нетерминал
    Нетерминал = ЭлементГрамматики();

    // Далее ожидаются разделители левой и правой части правила
    // в виде: -> => ::=
    // Игнорируем пустые символы
    while (НомерСимволаГрамматики != ТекущаяСтрокаГрамматики.Length)
    {
        if (!Char.IsWhiteSpace(ТекущаяСтрокаГрамматики[НомерСимволаГрамматики]))
            break;
        ++НомерСимволаГрамматики;
    }

    // Добавим в конец строки два пробела, чтобы корректно проверить
    // наличие разделителей правила в виде: -> => ::=
    ТекущаяСтрокаГрамматики += "  ";

    // Проверяем наличие разделителей правила
    if ((tmp = ТекущаяСтрокаГрамматики.Substring(НомерСимволаГрамматики, 2)) ==
              "->" || tmp == "=>")
        НомерСимволаГрамматики += 2;
    else
        НомерСимволаГрамматики += 3;

    // Убираем добавленные пробелы
    ТекущаяСтрокаГрамматики = ТекущаяСтрокаГрамматики.Substring
                (0, ТекущаяСтрокаГрамматики.Length - 2);

    // Собираем все альтернативы правила
    for (;;)
    {
        i = 0;
        Альтернатива[i++] = Нетерминал;
        Альтернатива[i++] = ЭлементГрамматики();
```

```
        for (;;)
        {
            str = ЭлементГрамматики();
            if (str == "" || str == "|")
            {
                ПравилаГрамматики[КоличествоПравилГрамматики] = new string[i];
                for (j = 0; j < i; ++j)
                 ПравилаГрамматики[КоличествоПравилГрамматики][j] = Альтернатива[j];

                ++КоличествоПравилГрамматики;

                // Альтернатив больше нет?
                if (str == "")
                    return;
            }

            // Новая альтернатива?
            if (str == "|")
                break;
            Альтернатива[i++] = str;
        }
    }
}

// Поиск epsilon-порождающих нетерминалов
private void Эпсилон()
{
    int    i, j, k, n;
    string str1, str2;

    ЭпсилонНетерминалы = new string[КоличествоПравилГрамматики];

    // Просмотреть всю грамматику и выявить все нетерминалы типа A -> eps
 for (i = КоличествоЭпсилонНетерминалов = 0; i < КоличествоПравилГрамматики; ++i)
    {
        // Есть нетерминал с таким символом?
        if (ПравилаГрамматики[i][1] == "#")
            ЭпсилонНетерминалы[КоличествоЭпсилонНетерминалов++] =
                        ПравилаГрамматики[i][0];
    }

    // Пустых правил нет?
    if (КоличествоЭпсилонНетерминалов == 0)
        return;

    // Теперь перебираем все правила с подстановкой пустых
    // нетерминалов до тех пор, пока это возможно
    for (;;)
    {
        n = КоличествоЭпсилонНетерминалов;
        for (i = 0; i < КоличествоПравилГрамматики; ++i)
        {
            // Этот нетерминал уже есть в множестве пустых правил?
            for (j = 0; j < КоличествоЭпсилонНетерминалов &&
                   ПравилаГрамматики[i][0] != ЭпсилонНетерминалы[j]; ++j)
                ;
            if (j != КоличествоЭпсилонНетерминалов)
                continue;

            // Пытаемся путем подстановок уже пустых правил выяснить, не
            // будет ли пустым и этот нетерминал
            str1 = "";
```

```
            for (j = 1; j < ПравилаГрамматики[i].Length; ++j)
            {
                str2 = ПравилаГрамматики[i][j];

                // Это терминал?
                if (str2[0] == 'T')
                {
                    str1 = "NoEmpty";
                    break;
                }
                if (str2 == "#")
                    break;

                // Этот нетерминал уже есть в множестве пустых правил?
                for (k = 0; k < КоличествоЭпсилонНетерминалов &&
                                str2 != ЭпсилонНетерминалы[k]; ++k)
                    ;
                if (k == КоличествоЭпсилонНетерминалов)
                {
                    str1 = "NoEmpty";
                    break;
                }
            }

            // Этот нетерминал стал пустым?
            if (str1 == "")
                // Добавляем его в множество
                ЭпсилонНетерминалы[КоличествоЭпсилонНетерминалов++] =
                            ПравилаГрамматики[i][0];
        }

        // Множество пустых правил изменилось?
        if (n == КоличествоЭпсилонНетерминалов)
            return;
    }
}

// Получить список всех всех нетерминалов и терминалов
private void ВсеНетерминалыТерминалы()
{
    int      i, j, k, n;
    string[] str = new string[1000];
    string   tmp;

    // Нетерминалы
    for (i = n = 0; i < КоличествоПравилГрамматики; ++i)
    {
        tmp = ПравилаГрамматики[i][0];
        for (j = 0; j < n && str[j] != tmp; ++j)
            ;
        if (j == n)
            str[n++] = tmp;
    }

    // Терминалы
    for (i = 0; i < КоличествоПравилГрамматики; ++i)
    {
        for (j = 1; j < ПравилаГрамматики[i].Length; ++j)
        {
            tmp = ПравилаГрамматики[i][j];
            if (tmp[0] != 'T')
                continue;
```

```
                    for (k = 0; k < n && str[k] != tmp; ++k)
                        ;
                    if (k == n)
                        str[n++] = tmp;
                }
            }

            // Формируем результат
            НетерминалыТерминалы = new string[n];
            for (i = 0; i < n; ++i)
                НетерминалыТерминалы[i] = str[i];
        }

        // Вернуть true, если символ str аннулирующий
        private bool Аннулирующий (string str)
        {
            int i;

            if (str[0] != 'N')
                return false;

            for (i = 0; i < КоличествоЭпсилонНетерминалов &&
                           ЭпсилонНетерминалы[i] != str; ++i)
                ;

            return i != КоличествоЭпсилонНетерминалов;
        }

        // Построить заготовку таблицы
        private void ЗаготовкаТаблицы (ref DataGridView p)
        {
            int    i;
            string str;

            DataGridViewTextBoxColumn[] column =
new DataGridViewTextBoxColumn[НетерминалыТерминалы.Length];
            for (i = 0; i < НетерминалыТерминалы.Length; ++i)
            {
                column[i] = new DataGridViewTextBoxColumn();
                str = НетерминалыТерминалы[i].Substring(1);
                if (str != "\'" && str[0] == '\'')
                    str = str.Substring(1);
                column[i].HeaderText = str;
                column[i].Name = i.ToString();
            }
            p.Columns.AddRange(column);

            for (i = 0; i < НетерминалыТерминалы.Length; ++i)
            {
                p.Columns[i].HeaderCell.Style.Alignment =
 DataGridViewContentAlignment.MiddleCenter;
                p.Rows.Add();
                str = НетерминалыТерминалы[i].Substring(1);
                if (str != "\'" && str[0] == '\'')
                    str = str.Substring(1);
                p.Rows[i].HeaderCell.Value = str;
            }
            p.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
            p.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.AllCells;
            p.RowHeadersWidthSizeMode =
 DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders;
        }
```

```
        // Построить заготовку таблицы П+
        private void ЗаготовкаТаблицы()
        {
            int    i;
            string str;

            DataGridViewTextBoxColumn[] column =
new DataGridViewTextBoxColumn[НетерминалыТерминалы.Length + 1];
            for (i = 0; i < НетерминалыТерминалы.Length; ++i)
            {
                column[i] = new DataGridViewTextBoxColumn();
                str = НетерминалыТерминалы[i].Substring(1);
                if (str != "\'" && str[0] == '\'')
                    str = str.Substring(1);
                column[i].HeaderText = str;
                column[i].Name = i.ToString();
            }
            column[i] = new DataGridViewTextBoxColumn();
            column[i].HeaderText = "КОНЕЦ";
            column[i].Name = i.ToString();

            dataGridView7.Columns.AddRange(column);

            for (i = 0; i < НетерминалыТерминалы.Length + 1; ++i)
            {
                dataGridView7.Columns[i].HeaderCell.Style.Alignment =
 DataGridViewContentAlignment.MiddleCenter;
            }

            for (i = 0; i < НетерминалыТерминалы.Length; ++i)
            {
                dataGridView7.Rows.Add();
                str = НетерминалыТерминалы[i].Substring(1);
                if (str != "\'" && str[0] == '\'')
                    str = str.Substring(1);
                dataGridView7.Rows[i].HeaderCell.Value = str;
            }
            dataGridView7.AutoSizeColumnsMode     =
 DataGridViewAutoSizeColumnsMode.AllCells;
            dataGridView7.AutoSizeRowsMode        =
 DataGridViewAutoSizeRowsMode.AllCells;
            dataGridView7.RowHeadersWidthSizeMode =
 DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders;
        }

        // Построить заготовку таблицы МПА
        private void ЗаготовкаТаблицыМПА()
        {
            int    i, j, n1, n2, n3;
            string str;

            // Подсчет числа нетерминалов и терминалов
            for (n1 = 0; НетерминалыТерминалы[n1][0] == 'N'; ++n1)
                ;
            i = n3 = n1;

            // Подсчет числа столбцов с учетом КОНЕЦ
          for (n2 = mark[mark.Length - 1] ? 1 : 0; n1 < НетерминалыТерминалы.Length; ++n1)
            {
                if (mark[n1])
                    ++n2;
            }
```

```
            ТерминалыМПА = new string[n2];
            DataGridViewTextBoxColumn[] column = new DataGridViewTextBoxColumn[n2];
            for (j = 0; i < НетерминалыТерминалы.Length; ++i)
            {
                if (mark[i])
                {
                    str = НетерминалыТерминалы[i];
                    ТерминалыМПА[j] = str;
                    column[j] = new DataGridViewTextBoxColumn();
                    column[j].HeaderText = str.Substring(2);
                    column[j].Name = j.ToString();
                    ++j;
                }
            }

            if (mark[mark.Length - 1])
            {
                ТерминалыМПА[j]      = " КОНЕЦ";
                column[j]            = new DataGridViewTextBoxColumn();
                column[j].HeaderText = "КОНЕЦ";
                column[j].Name       = j.ToString();
                ++j;
            }

            Действия = new string[Высота = n3 + 1, j];

            dataGridView8.Columns.AddRange(column);

            for (i = 0; i < n2; ++i)
            {
                dataGridView8.Columns[i].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleCenter;
            }

            for (i = 0; i < n3 + 1; ++i)
            {
                dataGridView8.Rows.Add();
                if (i == n3)
                    str = "НАЧАЛО";
                else
                    str = НетерминалыТерминалы[i].Substring(1);
                dataGridView8.Rows[i].HeaderCell.Value = str;
            }

            dataGridView8.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
            dataGridView8.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.AllCells;
            dataGridView8.RowHeadersWidthSizeMode =
DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders;
        }
        // Вернутить индекс символа в списке НетерминалыТерминалы
        private int ind(string str)
        {
            int i;

            for (i = 0; i < НетерминалыТерминалы.Length &&
                            str != НетерминалыТерминалы[i]; ++i)
                ;

            if (i == НетерминалыТерминалы.Length)
                --i;
            return i;
        }
```

```
// Отношение НПС
private void НПС()
{
    int    i, j, k, m;
    string str, str1;

    matr = new int[НетерминалыТерминалы.Length, НетерминалыТерминалы.Length];
    for (i = 0; i < НетерминалыТерминалы.Length; ++i)
    {
        for (j = 0; j < НетерминалыТерминалы.Length; ++j)
        {
            matr[i, j] = 0;
        }
    }

    ЗаготовкаТаблицы(ref dataGridView1);

    // Построение отношения
    for (i = 0; i < НетерминалыТерминалы.Length; ++i)
    {
        str = НетерминалыТерминалы[i];
        if (str[0] == 'T')
            break;

        // Просматриваем все правила с данным нетерминалом
        for (j = 0; j < КоличествоПравилГрамматики; ++j)
        {
            if (str != ПравилаГрамматики[j][0])
                continue;

            // Правило с одним эпсилон?
            if (ПравилаГрамматики[j][1] == "#")
                continue;

            for (m = 1; ;)
            {
                str1 = ПравилаГрамматики[j][m];

                // Правило завершилось эпсилоном?
                if (str1 == "#")
                    break;

                // Устанавливаем отношение с очередным символом правила
                k = ind(str1);
                dataGridView1.Rows[i].Cells[k].Value = "X";
                matr[i, k] = 1;

                // Символ не аннулирующий?
                if (!Аннулирующий(str1))
                    break;

                // Правило закончилось?
                if (++m == ПравилаГрамматики[j].Length)
                    break;
            }
        }
    }
}
```

```
// Вычисление рефлексивно-транзитивного замыкания
// методом  Воршалла
private void РефлексТранзит()
{
    int i, j, k;

    for (i = 0; i < НетерминалыТерминалы.Length; ++i)
    {
        for (j = 0; j < НетерминалыТерминалы.Length; ++j)
        {
            if (matr[j, i] != 0)
            {
                for (k = 0; k < НетерминалыТерминалы.Length; ++k)
                {
                    if (matr[i, k] != 0)
                        matr[j, k] = 1;
                }
            }
        }
        matr[i, i] = 1;
    }
}

// Отношение НС
private void НС()
{
    int i, j;

    ЗаготовкаТаблицы(ref dataGridView2);

    // Получаем рефлексивно-транзитивное замыкание
    РефлексТранзит();

    // Обнародуем отношение
    for (i = 0; i < НетерминалыТерминалы.Length; ++i)
    {
        for (j = 0; j < НетерминалыТерминалы.Length; ++j)
        {
            if (matr[i, j] == 1)
                dataGridView2.Rows[i].Cells[j].Value = "X";
        }
    }

    // Сохраним матрицу
    matrНС = new int[НетерминалыТерминалы.Length, НетерминалыТерминалы.Length];
    for (i = 0; i < НетерминалыТерминалы.Length; ++i)
    {
        for (j = 0; j < НетерминалыТерминалы.Length; ++j)
        {
            matrНС[i, j] = matr[i, j];
        }
    }
}
```

```
// Отношение ПП
private void ПП()
{
    int    i, j, k, ind1, ind2;
    string str;

    ЗаготовкаТаблицы(ref dataGridView3);

    // Матрица отношений
    matrПП = new int[НетерминалыТерминалы.Length, НетерминалыТерминалы.Length];
    for (i = 0; i < НетерминалыТерминалы.Length; ++i)
    {
        for (j = 0; j < НетерминалыТерминалы.Length; ++j)
        {
            matrПП[i, j] = 0;
        }
    }

    // Просматриваем все правила
    for (i = 0; i < КоличествоПравилГрамматики; ++i)
    {
        // Просматриваем каждый символ правила и все символы правее до тех пор,
        // пока они разделены аннулирующими нетерминалами
        for (j = 1; j < ПравилаГрамматики[i].Length - 1; ++j)
        {
            ind1 = ind(ПравилаГрамматики[i][j]);
            for (k = j + 1; k < ПравилаГрамматики[i].Length; ++k)
            {
                str = ПравилаГрамматики[i][k];
                if (str == "#")
                    break;

                // Делаем отметку
                ind2 = ind(str);
                dataGridView3.Rows[ind1].Cells[ind2].Value = "X";
                matrПП[ind1, ind2] = 1;

                // Символ не аннулирующий?
                if (!Аннулирующий(str))
                    break;
            }
        }
    }
}

// Отношение ПНК
private void ПНК()
{
    int    i, j, k, m, n;
    string str, str1;

    matr = new int[НетерминалыТерминалы.Length, НетерминалыТерминалы.Length];
    for (i = 0; i < НетерминалыТерминалы.Length; ++i)
    {
        for (j = 0; j < НетерминалыТерминалы.Length; ++j)
        {
            matr[i, j] = 0;
        }
    }

    ЗаготовкаТаблицы(ref dataGridView4);
```

```
    // Построение отношения
    for (i = 0; i < НетерминалыТерминалы.Length; ++i)
    {
        str = НетерминалыТерминалы[i];

        // Просматриваем все правила с данным символом
        for (j = 0; j < КоличествоПравилГрамматики; ++j)
        {
            if (str != ПравилаГрамматики[j][0])
                continue;

            // Правило с одним эпсилон?
            if (ПравилаГрамматики[j][1] == "#")
                continue;

            for (m = 1; ;)
            {
                str1 = ПравилаГрамматики[j][m];

                // Правило завершилось эпсилоном?
                if (str1 == "#")
                    break;

                // Правее символа str1 все символы аннулирующие
                // или он последний в правиле?
                for (n = m + 1; n < ПравилаГрамматики[j].Length &&
                          Аннулирующий(ПравилаГрамматики[j][n]); ++n)
                    ;

                // Да, этот символ не прямо на конце
                if (n >= ПравилаГрамматики[j].Length)
                {
                    // Устанавливаем отношение с очередным символом правила
                    k = ind(str1);
                    dataGridView4.Rows[k].Cells[i].Value = "X";
                    matr[k, i] = 1;
                }

                // Правило закончилось?
                if (++m == ПравилаГрамматики[j].Length)
                    break;
            }
        }
    }
}

// Отношение НК
private void НК()
{
    int i, j;

    ЗаготовкаТаблицы(ref dataGridView5);

    // Получаем рефлексивно-транзитивное замыкание
    РефлексТранзит();

    // Обнародуем отношение
    for (i = 0; i < НетерминалыТерминалы.Length; ++i)
    {
        for (j = 0; j < НетерминалыТерминалы.Length; ++j)
        {
            if (matr[i, j] == 1)
```

```
                dataGridView5.Rows[i].Cells[j].Value = "X";
            }
        }

        // Сохраним матрицу
        matrНК = new int[НетерминалыТерминалы.Length, НетерминалыТерминалы.Length];
        for (i = 0; i < НетерминалыТерминалы.Length; ++i)
        {
            for (j = 0; j < НетерминалыТерминалы.Length; ++j)
            {
                matrНК[i, j] = matr[i, j];
            }
        }
    }

    // Умножение матриц, результат в matr
    private void Mult(int[,] a, int[,] b)
    {
        int i, j, k, n = НетерминалыТерминалы.Length;

        for (i = 0; i < n; ++i)
        {
            for (j = 0; j < n; ++j)
            {
                matr[i, j] = 0;
                for (k = 0; k < n; ++k)
                {
                    if (a[i, k] != 0 && b[k, j] != 0)
                    {
                        matr[i, j] = 1;
                        break;
                    }
                }
            }
        }
    }

    // Отношение П = НК * ПП * НС
    private void П()
    {
        int    i, j, n = НетерминалыТерминалы.Length;
        int[,] t = new int[n, n];

        ЗаготовкаТаблицы(ref dataGridView6);

        Mult(matrНК, matrПП);
        for (i = 0; i < n; ++i)
        {
            for (j = 0; j < n; ++j)
            {
                t[i, j] = matr[i, j];
            }
        }

        Mult(t, matrНС);

        // Обнародуем отношение
        for (i = 0; i < n; ++i)
        {
            for (j = 0; j < n; ++j)
            {
                if (matr[i, j] == 1)
```

```
                dataGridView6.Rows[i].Cells[j].Value = "X";
            }
        }
    }

    // Отношение П+
    private void ПL()
    {
        int i, j, n = НетерминалыТерминалы.Length;

        ЗаготовкаТаблицы();

        // Обнародуем отношение П
        for (i = 0; i < n; ++i)
        {
            for (j = 0; j < n; ++j)
            {
                if (matr[i, j] == 1)
                    dataGridView7.Rows[i].Cells[j].Value = "X";
            }
        }

        // Добавляем в последний столбец первый столбец отношения НК
        for (i = 0; i < n; ++i)
        {
            if (matrНК[i, 0] == 1)
                dataGridView7.Rows[i].Cells[n].Value = "X";
        }
    }

    // Вычисление множества СЛЕД
    private void СЛЕД()
    {
        int    i, j, n, ind1;
        string str, str1;

        if (КоличествоЭпсилонНетерминалов == 0)
            return;
        МножествоСлед = new string[КоличествоЭпсилонНетерминалов][];

        for (i = 0; i < КоличествоЭпсилонНетерминалов; ++i)
        {
            ind1 = ind(ЭпсилонНетерминалы[i]);

            // Подсчет числа элементов
            n = matrНК[ind1, 0];
            for (j = 0; j < НетерминалыТерминалы.Length; ++j)
            {
                if (НетерминалыТерминалы[j][0] == 'T')
                    n += matr[ind1, j];
            }

            if (n != 0)
            {
                МножествоСлед[i] = new string[n];
                for (j = n = 0; j < НетерминалыТерминалы.Length; ++j)
                {
                    str = НетерминалыТерминалы[j];
                    if (str[0] == 'T' && matr[ind1, j] != 0)
                        МножествоСлед[i][n++] = str;
                }
```

```
            if (matrНК[ind1, 0] != 0)
                МножествоСлед[i][n] = " КОНЕЦ";
        }
    }
    // Обнародуем множество
    for (i = 0; i < КоличествоЭпсилонНетерминалов; ++i)
    {
        str = "СЛЕД(<" + ЭпсилонНетерминалы[i].Substring(1) + ">) = {";
        for (j = 0; j < МножествоСлед[i].Length; ++j)
        {
            str1 = МножествоСлед[i][j].Substring(1);
            if (str1 == "КОНЕЦ")
                str1 = "'КОНЕЦ";
            str += str1 + "'  ";
        }
        str = str.Substring(0, str.Length - 2) + "}\r\n";
        richTextBox3.AppendText(str);
    }
}
// Добавить в множество ПЕРВЫЙ индекс терминала,
// если его там нет
private void Добавить (int i, int ind)
{
    int j;

    for (j = 1; j <= ПЕРВЫЙ[i, 0]; ++j)
    {
        if (ПЕРВЫЙ[i, j] == ind)
            return;
    }
    ПЕРВЫЙ[i, j] = ind;
    ++ПЕРВЫЙ[i, 0];
}

// Печать пронумерованных правил грамматики
private void ПечатьПравилГрамматики(ref TextBox p)
{
    int    i, j;
    string str = "Пронумерованные правила грамматики\r\n\r\n", str1;

    for (i = 0; i < КоличествоПравилГрамматики; ++i)
    {
        str1 = ПравилаГрамматики[i][0];
        str += "[" + (i + 1).ToString() + "] <" + str1.Substring(1) + "> =>";
        for (j = 1; j < ПравилаГрамматики[i].Length; ++j)
        {
            str1 = ПравилаГрамматики[i][j];
            if (str1 == "#")
            {
                str += " #";
                break;
            }
            // Собрать правую часть правила
            if (str1[0] == 'T')
                str += " " + str1.Substring(1) + "'";
            else
                str += " <" + str1.Substring(1) + ">";
        }
        str += "\r\n";
    }
    p.AppendText(str + "\r\n");
}
```

```
        private void Form1_Load(object sender, EventArgs e)
        {
            dataGridView1.Visible = false;
            dataGridView2.Visible = false;
            dataGridView3.Visible = false;
            dataGridView4.Visible = false;
            dataGridView5.Visible = false;
            dataGridView6.Visible = false;
            dataGridView7.Visible = false;
            dataGridView8.Visible = false;
        }

        // Примеры грамматик
        readonly private string example1 =
        "; Грамматика оператора присваивания с бинарными операциями + и *\r\n" +
        "; #КОММЕНТАРИЙ //\r\n" +
        "\r\n" +
        "<S>         => <Tip> <iD> = <E>\r\n" +
        "<E>         => <T> <E-список>\r\n" +
        "<E-список> => + <T> <E-список> | #\r\n" +
        "<T>         => <P> <T-список>\r\n" +
        "<T-список> => * <P> <T-список> | #\r\n" +
        "<P>         => (<E>) | 'ид'\r\n" +
        "<Tip>       => 'int'\r\n" +
        "<iD>        => 'ид'";

        readonly private string prog1 =
        "// Грамматика оператора присваивания с бинарными операциями + и *\r\n" +
        "int a = (ijk + b) * (c + d) + z";

        readonly private string example2 =
        "; Грамматика из книги Льюис Ф. \"Теоретические основы проектирования
компиляторв\", стр. 284\r\n" +
        "; #КОММЕНТАРИЙ /*\r\n" +
        "\r\n" +
        "<A> => <B> <C> 'c' | 'e' <D> <B>\r\n" +
        "<B> => # | 'b' <C> <D> <E>\r\n" +
        "<C> => <D> 'a' <B> | 'c' 'a'\r\n" +
        "<D> => # | 'd' <D>\r\n" +
        "<E> => 'e' <A> 'f' | 'c'";

        readonly private string prog2 =
        "/* Грамматика из книги Льюис Ф.\r\n" +
        "\"Теоретические основы проектирования компиляторв\", стр. 284 */\r\n" +
        "e d d d";

        readonly private string example3 =
        "; Грамматика арифметического выражения со скобками, массивами и вызовом
функций\r\n" +
        "<Z>  ::= <E> | #\r\n" +
        "<TM> ::= <T> <M>\r\n" +
        "<E>  ::= <TM>\r\n" +
        "<M>  ::= -<TM> | +<TM> | #\r\n" +
        "<FG> ::= <F> <G>\r\n" +
        "<T>  ::= <FG>\r\n" +
        "<G>  ::= *<FG> | /<FG> | #\r\n" +
        "<F>  ::= -<F> | (<E>) | 'цел' | 'вещ' |'ид' <A>\r\n" +
        "<A>  ::= <Y> | #\r\n" +
        "<Y>  ::= [<E>] |(<L> | #\r\n" +
        "<EN> ::= <E> <N>\r\n" +
        "<L>  ::= ) | <EN>)\r\n" +
        "<N>  ::= ,<EN> | #\r\n";
```

```
        readonly private string prog3 =
        "-Func0 () / a * Func1 (a * (-17 + z / -(7 + xy)), b / 3.14, array[cos (tan
(100.8))])\r\n";

        readonly private string example4 =
        "; Грамматика оператора switch\r\n" +
        "; #КОММЕНТАРИЙ // #КОММЕНТАРИЙ /*\r\n" +
        "\r\n" +
        "<Программа> => 'BEGIN' <Описание>\r\n" +
        "<Описание>  => 'CHAR' <Q> <X>\r\n" +
        "<Q>         => 'ид' <Инит>\r\n" +
        "<Инит>      =>  = <АрифмВыр> | #\r\n" +
        "<X>         => , <Q> <X> |;<A>\r\n" +
        "<A>         => 'CHAR' <Q> <X> | <Операторы> 'END'\r\n" +
        "\r\n" +
        "<Операторы> => 'ид'= <АрифмВыр>; <Операторы> | 'BREAK'; | 'SWITCH' (<АрифмВыр>)
{ <CaseOp> <Операторы> | #\r\n" +
        "<CaseOp>    => 'CASE' <АрифмВыр>: <Операторы> <CaseOp> | }\r\n" +
        "\r\n" +
        "<АрифмВыр>  => <T><R>\r\n" +
        "<R>         => +<T><R> | -<T><R> | #\r\n" +
        "<T>         => <E><F>\r\n" +
        "<F>         => *<E><F> | /<E><F> | #\r\n" +
        "<E>         => (<АрифмВыр>) | 'сим' | 'ид' | 'цел' | - <E>\r\n";

        readonly private string prog4 =
        "// Это корректная программа\r\n" +
        "BEGIN\r\n" +
        "    // Описание переменных\r\n" +
        "    CHAR a = '\\t', b = 'a';\r\n" +
        "    CHAR x, y, z = '\\n';\r\n" +
        "\r\n" +
        "    // Операторы присваивания\r\n" +
        "    a = -100 / 7 + x * y * z;\r\n" +
        "    x = -a; y = x + 12; z = -x - y;\r\n" +
        "\r\n" +
        "    // Вложенные switch\r\n" +
        "    SWITCH (z + y)\r\n" +
        "    {\r\n" +
        "    CASE '1':\r\n" +
        "        b = -1 * b;\r\n" +
        "        SWITCH (12 / y * -Pi / xxx)\r\n" +
        "        {\r\n" +
        "        CASE '3': CASE 3:\r\n" +
        "            y = y - 4;\r\n" +
        "            BREAK;\r\n" +
        "\r\n" +
        "        CASE '4':\r\n" +
        "            z = x + y-1 * b;\r\n" +
        "        }\r\n" +
        "\r\n" +
        "    CASE 2:\r\n" +
        "        b = b + -1;\r\n" +
        "    }\r\n" +
        "\r\n" +
        "    // switch с пустым телом\r\n" +
        "    SWITCH (a)\r\n" +
        "    {\r\n" +
        "\r\n" +
        "    }\r\n" +
        "END\r\n" +
```

```
        "\r\n" +
        "/*" +
        "// А эта ошибочна\r\n" +
        "BEGIN\r\n" +
        "    // Описание переменных\r\n" +
        "    CHAR a = 100, b = a * 1e-3;\r\n" +
        "\r\n" +
        "    a = -100;\r\n" +
        "END\r\n" +
        "*/";

        readonly private string example5 =
        "; Грамматика, описывающая язык LL(2), который не является LL(1)\r\n" +
        "\r\n" +
        "<S> => 'a' <S> <A> | #\r\n" +
        "<A> => 'a' 'b' <S> | 'c'";

        readonly private string prog5 =
        "// Программа отсутствует...";

        readonly private string example6 =
                "; Грамматика подмножества языка программирования Рапира\r\n" +
        "; #ИДЕНТИФИКАТОР РУССКИЙ_АНГЛИЙСКИЙ\r\n" +
        "; #КОММЕНТАРИЙ (*\r\n" +
        "\r\n" +
        "<S>                    -> <ОписаниеПеременных> <Программа> 'КНЦ' ;\r\n" +
        "\r\n" +
        "<ОписаниеПеременных> -> 'ЦЕЛЫЕ' <Идентификатор1> <Список>\r\n" +
        "<Список>               -> , <Идентификатор1> <Список> | ; <НовоеОписание>\r\n" +
        "<НовоеОписание>       -> <ОписаниеПеременных> | #\r\n" +
        "\r\n" +
        "<Программа>           -> <Set> | <For> | <Rep> | <While> | <If> | #\r\n" +
        "<Set>                  -> <Присваивание> = <Выражение>; <Программа>\r\n" +
        "<For>                  -> 'ДЛЯ' <Идентификатор2> 'ОТ' <Выражение> 'ДО' <Выражение>
<Step> <Продолжение1>\r\n" +
        "<Rep>                  -> 'ПОВТОР' <Выражение> <РазРаза> <Продолжение>\r\n" +
        "<While>                -> 'ПОКА' <Условие> <Продолжение>\r\n" +
        "<If>                   -> 'ЕСЛИ' <Условие> 'ТО' <Программа> <Else> 'ВСЕ' ;
<Программа>\r\n" +
        "<Else>                 -> 'ИНАЧЕ' <Программа> | #\r\n" +
        "<Step>                 -> 'ШАГ' <Выражение> | #\r\n" +
        "<Продолжение>          -> '::' <Программа> 'ВСЕ' ; <Программа>\r\n" +
        "<Продолжение1>         -> '::' <Программа> <КонецДЛЯ> ; <Программа>\r\n" +
        "<КонецДЛЯ>             -> 'ВСЕ'\r\n" +
        "<Условие>              -> <Выражение> <Сравнение> <Выражение>\r\n" +
        "<Сравнение>            -> '>=' | '<=' | '/=' | = | > | '<'\r\n" +
        "<РазРаза>              -> 'РАЗ' | 'РАЗА'\r\n" +
        "\r\n" +
        "<Выражение>            -> <Term> <Expr>\r\n" +
        "<Expr>                 -> -<Выражение> | +<Выражение> | #\r\n" +
        "<FG>                   -> <Factor> <G>\r\n" +
        "<Term>                 -> <FG>\r\n" +
        "<G>                    -> *<FG> | /<FG> | '//'<FG> | #\r\n" +
        "<Factor>               -> -<Factor> | (<Выражение>) | <Число> |
<Идентификатор3>\r\n" +
        "<Число>                -> 'цел'\r\n" +
        "<Присваивание>         -> 'ид'\r\n" +
        "<Идентификатор1>       -> 'ид'\r\n" +
        "<Идентификатор2>       -> 'ид'\r\n" +
        "<Идентификатор3>       -> 'ид'\r\n" +
        "\r\n" +
        "; Семантика\r\n" +
```

```
        "; $0 <Идентификатор1> Контроль дубликатов идентификаторов\r\n" +
        "; $1 <Идентификатор2> Контроль определения идентификатора переменной цикла
ДЛЯ\r\n" +
        "; $2 <Идентификатор2> Контроль пересечений переменных циклов ДЛЯ\r\n" +
        "; $1 <Идентификатор3> Контроль определения переменной в выражении\r\n" +
        "; $3 <Число>          Контроль значения константы в диапазоне двух байтов\r\n" +
        "; $4 <КонецДЛЯ>       Контроль завершения цикла ДЛЯ\r\n" +
        "; $1 <Присваивание>   Контроль определения переменной присваивания\r\n" +
        "; $5 <Присваивание>   Контроль изменения переменной цикла ДЛЯ\r\n";


        /*
        https://ppt-online.org/839368

        // Это другой вид синтаксиса арифметического выражения
        <S> -> (<S>) <V> <U> | <A> <V> <U> | <Y> <G> <V> <U>
        <U> -> + <T> <U> | - <T> <U> | #
        <T> -> (<S>) <V> | <A> <V> | <Y> <G> <V>
        <V> -> * <F> <V> | / <F> <V> | #
        <F> -> (<S>) | <A> | <Y> <G>
        <G> -> (<S>) | <A>
        <A> -> 'ид' | 'цел'
        <Y> -> + | -
        */
        readonly private string prog6 =
        "ЦЕЛЫЕ k, i, j, Привет, a, b, A, N;\r\n" +
        "ЦЕЛЫЕ ОченьДлинныйИдентификатор, abc;\r\n" +
        "ЦЕЛЫЕ d, x, y;\r\n" +
        "\r\n" +
        "k = -9; i = j;\r\n" +
        "\r\n" +
        "(* Примеры циклов повторений *)\r\n" +
        "ПОВТОР k + i РАЗ ::\r\n" +
        "  ПОВТОР 2 РАЗА ::\r\n" +
        "    k = -(Привет + 7) * (k + 1);\r\n" +
        "  ВСЕ;\r\n" +
        "ВСЕ;\r\n" +
        "\r\n" +
        "ПОКА a /= b ::\r\n" +
        "  ЕСЛИ a > 0 ТО k = 1; ИНАЧЕ k = 7; ВСЕ;\r\n" +
        "  a = a + 1;\r\n" +
        "  ЕСЛИ a >= 8 * 7 ТО A = N; ВСЕ;\r\n" +
        "ВСЕ;\r\n" +
        "\r\n" +
        "(* Примеры циклов повторений *)\r\n" +
        "ДЛЯ i ОТ 1 ДО 10 ::\r\n" +
        "  ДЛЯ j ОТ 1 ДО 5 + 1 ШАГ 7 ::\r\n" +
        "    k = k * j;\r\n" +
        "    k = k * j + Привет;\r\n" +
        "  ВСЕ;\r\n" +
        "  ДЛЯ j ОТ -100 * k ДО ОченьДлинныйИдентификатор ШАГ 7 ::\r\n" +
        "    k = k * j;\r\n" +
        "    k = k * j + Привет;\r\n" +
        "  ВСЕ;\r\n" +
        "  k = k * j + ОченьДли / 8;\r\n" +
        "  d = k * j + 1;\r\n" +
        "  d = k * j / 1;\r\n" +
        "ВСЕ;\r\n" +
        "\r\n" +
        "abc = 9; x = y;\r\n" +
        "\r\n" +
        "КНЦ;\r\n";
```

```
        // Установить новый пример грамматики
        private void ПримерГрамматики(string txt1, string txt2)
        {
            richTextBox1.Clear();
            richTextBox1.AppendText(txt1);
            richTextBox2.Clear();
            richTextBox2.AppendText(txt2);

            richTextBox1.SelectionStart = 0;
            richTextBox2.SelectionStart = 0;

            tabControl1.SelectTab(tabPage1);
            ХорошаяГрамматика = false;
        }

        private void пример1ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            ПримерГрамматики(example1, prog1);
        }

        private void пример2ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            ПримерГрамматики(example2, prog2);
        }

        private void пример3ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            ПримерГрамматики(example3, prog3);
        }

        private void пример4ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            ПримерГрамматики(example4, prog4);
        }

        private void пример5ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            ПримерГрамматики(example5, prog5);
        }

        private void пример6ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            ПримерГрамматики(example6, prog6);
        }

        // return true, если символ подчеркивания или английская/русская буква
        private bool Буква(char c)
        {
            string rus =
"ёйцукенгшщзхъфывапролджэячсмитьбюЁЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮ";

            if (c == '_')
                return true;
            // Идентификатор только из английских букв?
            if (type_ident == 1)
                return c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z';
            if (rus.Contains(c.ToString()))
                return true;
            if (type_ident == 2 && c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z')
                return true;
            return false;
        }
```

```
// return true, если символ подчеркивания, английская буква или цифра
private bool БуквaЦифра(char c)
{
    return Буква(c) || Char.IsDigit(c);
}

// Вернуть true, если символ шестнадцатиричная цифра
private bool Шестнадцатиричная (char c)
{
    return Char.IsDigit(c) || c >= 'a' && c <= 'f' || c >= 'A' && c <= 'F';
}

// Получить очередную лексему
private string lex()
{
    char   c;
    int    j;
    string str;

    loop:

    for (;;)
    {
        // Надо читать очередную строку программы?
        if (ind_line < 0)
        {
            // Программа закончена?
            if (richTextBox2.Lines.Length == record)
            {
                // Выполнялся разбор комментария?
                if (yes_comment)
                {
                    lex_ind = 0;
                    return "*Многострочный комментарий не закончен";
                }

                // Конец программы
                lex_ind = 0;
                return "";
            }

            // Получим очередную строку программы
            src_line = richTextBox2.Lines[record++];

            // Начало строки разбора
            ind_line = 0;
        }

        if (yes_comment)
        {
            // При анализе многострочного комментария пытаемся
            // определить его конец
            if ((j = src_line.Substring(ind_line).IndexOf(end_comment)) < 0)
            {
                // Комментарий продолжается
                ind_line = -1;
                continue;
            }
```

```
            // Нашли конец многострочного комментария,
            // продолжаем сканирование после него
            yes_comment = false;
            ind_line += j + end_comment.Length;
        }

        // Игнорируем пустые символы
        while (ind_line < src_line.Length &&
                    Char.IsWhiteSpace(src_line[ind_line]))
            ++ind_line;

        // Строка не закончена?
        if (ind_line < src_line.Length)
            break;

        // Признак чтения следующей строки программы
        ind_line = -1;
    }

    // Начало очередной лексемы для выдачи ошибок
    lex_ind = ind_line;

    str = src_line + " ";

    // Это многострочный комментарий {...}?
    if ((c = src_line[ind_line]) == '{')
    {
        if (type_comment[3])
        {
            ++ind_line;
            yes_comment = true;

            // Устанавливаем значение символов,
            // завершающих многострочный комментарий
            end_comment = "}";
            goto loop;
        }

        // Контроль наличия в грамматике лексемы, начинающейся с символа '{'
        if ((str = ЭтоЛексема()) == "")
            return "*Ошибочная лексема";

        // Это терминал
        ind_line += str.Length;
        return "T'" + str;
    }

    // Это многострочный комментарий (*...*)?
    if (c == '(' && str[ind_line + 1] == '*')
    {
        if (type_comment[2])
        {
            ind_line += 2;
            yes_comment = true;

            // Устанавливаем значение символов,
            // завершающих многострочный комментарий
            end_comment = "*)";
            goto loop;
        }
```

```
        // Контроль наличия в грамматике лексемы, начинающейся с символа '('
        if ((str = ЭтоЛексема()) == "")
            return "*Ошибочная лексема";

        // Это терминал
        ind_line += str.Length;
        return "T'" + str;
    }

    // Это многострочный комментарий /*...*/?
    if (c == '/' && str[ind_line + 1] == '*')
    {
        if (type_comment[1])
        {
            ind_line += 2;
            yes_comment = true;

            // Устанавливаем значение символов,
            // завершающих многострочный комментарий
            end_comment = "*/";
            goto loop;
        }

        // Контроль наличия в грамматике лексемы, начинающейся с символа '/'
        if ((str = ЭтоЛексема()) == "")
            return "*Ошибочная лексема";

        // Это терминал
        ind_line += str.Length;
        return "T'" + str;
    }

    // Это однострочный комментарий //?
    if (c == '/' && str[ind_line + 1] == '/')
    {
        if (type_comment[0])
        {
            ind_line = -1;
            goto loop;
        }

        // Контроль наличия в грамматике лексемы, начинающейся с символа '/'
        if ((str = ЭтоЛексема()) == "")
            return "*Ошибочная лексема";

        // Это терминал
        ind_line += str.Length;
        return "T'" + str;
    }

    // Начало символьной константы?
    if (c == '\'')
    {
        // Грамматика допускает символьные константы?
        if (!ЕстьСимвол)
            return "*Грамматика не допускает терминал 'символьная константа'";

        str = src_line + "   ";
```

```
    // Стуация '\ ?
    if ((c = str[++ind_line]) == '\\')
    {
        switch (c = str[++ind_line])
        {
       case 'n': case 'r': case 't': case 'b': case '\\': case '0': case '"':
                if (str[++ind_line] != '\'')
                    return "*Ошибочная запись символьной константы";
                ++ind_line;
                return "T'сим";

            case 'x': case 'X':
                if (!Шестнадцатиричная(c) ||
                    !Шестнадцатиричная(str[++ind_line]) ||
                     str[++ind_line] != '\'')
                    return "*Ошибочная запись символьной константы";
                ++ind_line;
                return "T'сим";
        }
        return "*Ошибочная запись символьной константы";
    }

    // Два апострофа подряд - ошибка
    if (c == '\'')
        return "*Ошибочная запись символьной константы";
    if (str[++ind_line] != '\'')
        return "*Ошибочная запись символьной константы";
    ++ind_line;
    return "T'сим";
}

// Начало символьной строки?
if (c == '"')
{
    // Грамматика допускает символьные константы?
    if (!ЕстьСтрока)
        return "*Грамматика не допускает терминал 'символьная строка'";

    str = src_line + "\r";

    // До конца символьной строки или до конца оператора
    while ((c = str[++ind_line]) != '"' && c != '\r')
    {
        // Стуация \ ?
        if (c == '\\')
        {
            switch (c = str[++ind_line])
            {
       case 'n': case 'r': case 't': case 'b': case '\\': case '0': case '"':
                    continue;

                case 'x':
                case 'X':
                    if (!Шестнадцатиричная(c) ||
                        !Шестнадцатиричная(str[++ind_line]) ||
                         str[++ind_line] != '\'')
                        return "*Ошибочная запись символьной строки";
                    continue;
            }
            return "*Ошибочная запись символьной строки";
        }
    }
```

```
            if (c == '\r')
                return "*Незавершенная символьная строка";
            ++ind_line;
            return "Т'стр";
        }

        // Начало числа?
        if (Char.IsDigit(c))
        {
            int beg_pos = ind_line;

            str = src_line + " ";
            while (Char.IsDigit(str[ind_line]))
                ++ind_line;

            // Это целое число?
            if ((c = str[ind_line]) != '.' && c != 'e' && c != 'E')
            {
                // Грамматика допускает целые числа?
                if (!ЕстьЦелое)
                    return "*Грамматика не допускает терминал 'целое число'";
                if (Буква(c))
                    return "*Ошибочная запись целого числа";

                // Сохраняем число для семантики
    if (!Int32.TryParse(str.Substring(beg_pos, ind_line - beg_pos), out integer_num))
                    return "*Слишком большое число";
                return "Т'цел";
            }

            // Грамматика допускает вещественные числа?
            if (!ЕстьВещественное)
                return "*Грамматика не допускает терминал 'вещественное число'";

            // Есть дробная часть вещественного числа?
            if (c == '.')
            {
                if (!Char.IsDigit(str[++ind_line]))
                    return "*Ошибочная запись вещественного числа";
                while (Char.IsDigit(str[++ind_line]))
                    ;
                c = str[ind_line];
            }

            // Есть экспонента вещественного числа?
            if (c == 'e' || c == 'E')
            {
                c = str[++ind_line];
                if (c == '+' || c == '-')
                    c = str[++ind_line];
                if (!Char.IsDigit(c))
                    return "*Ошибочная запись вещественного числа";
                while (Char.IsDigit(str[++ind_line]))
                    ;
            }

            // Сохраняем число для семантики
    if (!Double.TryParse(str.Substring(beg_pos, ind_line - beg_pos), out double_num))
                return "*Ошибочная запись вещественного числа";
            return "Т'вещ";
        }
```

```
        // Начало идентификатора?
        if (Буква(c))
        {
            str = src_line + " ";
            string str2;

            // Определяем конец идентификатора
            while (БукваЦифра(str[ind_line]))
                ++ind_line;

            // Это служебное слово?
            str2 = src_line.Substring(lex_ind, ind_line - lex_ind);
            if (ЭтоСлужебноеСлово(str2))
                return "T'" + str2;

            // Грамматика допускает идентификатор?
            if (!ЕстьИдентификатор)
                return "*Грамматика не допускает терминал 'идентификатор'";

            // Сохраняем идентификатор для семантики
            ident = str2;

            // Терминал идентификатор
            return "T'ид";
        }

        // Контроль наличия в грамматике лексемы, начинающейся с текущего символа
        if ((str = ЭтоЛексема()) == "")
            return "*Ошибочная лексема";

        // Это терминал
        ind_line += str.Length;
        return "T'" + str;
    }

    // Вычисление множества ПЕРВ
    private void ПЕРВ()
    {
        int i, j, k, ind1, n = КоличествоПравилГрамматики;
        string str, str1, str2, str3;

        ПечатьПравилГрамматики(ref textBox1);

        // Аннулирующие символы
        if (КоличествоЭпсилонНетерминалов == 0)
        {
            textBox1.AppendText("Аннулирующих символов нет\r\n\r\n");
        }
        else
        {
            str = "Аннулирующие символы = {";
            for (i = 0; i < КоличествоЭпсилонНетерминалов; ++i)
                str += "<" + ЭпсилонНетерминалы[i].Substring(1) + ">  ";
            str = str.Substring(0, str.Length - 2) + "}\r\n\r\n";
            textBox1.AppendText(str);
        }
```

```
// Нетерминалы
for (i = 0; i < НетерминалыТерминалы.Length; ++i)
{
    str = НетерминалыТерминалы[i];
    if (str[0] != 'N')
        break;
    str = "ПЕРВЫЙ(<" + str.Substring(1) + ">) = {";
    for (j = 0; j < НетерминалыТерминалы.Length; ++j)
    {
        str1 = НетерминалыТерминалы[j];
        if (matrHC[i, j] != 0 && str1[0] == 'T')
        {
            str += str1.Substring(1) + "'  ";
        }
    }
    str = str.Substring(0, str.Length - 2) + "}\r\n";
    textBox1.AppendText(str);
}

// Правые части правил
ПЕРВЫЙ = new int[n, 100];
textBox1.AppendText("\r\n");
for (i = 0; i < n; ++i)
{
    // Просматриваем каждый символ правила и все символы правее до тех пор,
    // пока они разделены аннулирующими нетерминалами
    ПЕРВЫЙ[i, 0] = 0;
    for (j = 1; j < ПравилаГрамматики[i].Length; ++j)
    {
        str1 = ПравилаГрамматики[i][j];
        if (str1 == "#")
            break;
        ind1 = ind(str1);
        if (str1[0] == 'T')
        {
            Добавить(i, ind1);
            break;
        }

        // Добавляем множество ПЕРВЫЙ нетерминала
        for (k = 0; k < НетерминалыТерминалы.Length; ++k)
        {
            str2 = НетерминалыТерминалы[k];
            if (matrHC[ind1, k] != 0 && str2[0] == 'T')
                Добавить(i, k);
        }

        // Символ не аннулирующий?
        if (!Аннулирующий(str1))
            break;
    }

    // Собрать правую часть правила
    if (ПравилаГрамматики[i][1] == "#")
    {
        str3 = "#";
    }
    else
    {
        str3 = "";
```

```
            for (j = 1; j < ПравилаГрамматики[i].Length; ++j)
            {
                str1 = ПравилаГрамматики[i][j];
                if (str1[0] == 'T')
                    str3 += "  " + str1.Substring(1) + "'";
                else
                    str3 += "  <" + str1.Substring(1) + ">";
            }
            str3 = str3.Substring(2);
        }

        str = "[" + (i + 1).ToString() + "] ПЕРВЫЙ(" + str3 + ") = {";

        if (ПЕРВЫЙ[i, 0] == 0)
        {
            str += "}\r\n";
        }
        else
        {
            for (j = 0; j < ПЕРВЫЙ[i, 0]; ++j)
            {
                str1 = НетерминалыТерминалы[ПЕРВЫЙ[i, j + 1]];
                str += str1.Substring(1) + "'  ";
            }
            str = str.Substring(0, str.Length - 2) + "}\r\n";
        }
        textBox1.AppendText(str);
    }
}

// Вычисление множества ВЫБОР
// return сообщение об ошибке, если это не LL(1) грамматика
// Иначе возвращается пустая строка
private string ВЫБОР()
{
    int i, j, k, x, r, n, n1 = КоличествоПравилГрамматики;
    string str, str1;
    sel = new int[n1, n1 + 1];
    mark = new bool[НетерминалыТерминалы.Length + 1];

    // Сброс признаков терминалов, которые будут обозначать колонки таблицы МПА
    for (i = 0; i < mark.Length; ++i)
        mark[i] = false;

    ПечатьПравилГрамматики(ref textBox2);
    mark[НетерминалыТерминалы.Length] = true;

    for (i = 0; i < n1; ++i)
    {
        str1 = ПравилаГрамматики[i][0];
     str = "[" + (i + 1).ToString() + "] ВЫБОР(<" + str1.Substring(1) + ">) = {";
        sel[i, 0] = 0;

        // Это правило типа A => эпсилон?
        if (ПравилаГрамматики[i][1] == "#")
        {
            // Берем множество СЛЕД для порождающего символа
            for (j = 0; str1 != ЭпсилонНетерминалы[j]; ++j)
                ;
            for (n = 0; n < МножествоСлед[j].Length; ++n)
            {
                str1 = МножествоСлед[j][n];
```

```
                if (str1 == " КОНЕЦ")
                {
                    str1 = " 'КОНЕЦ";
                    sel[i, ++sel[i, 0]] = -1;
                    mark[НетерминалыТерминалы.Length] = true;
                }
                else
                {
                    sel[i, ++sel[i, 0]] = k = ind(str1);
                    mark[k] = true;
                }
                str1 = str1.Substring(1);
                str += str1 + "'  ";
            }
            str = str.Substring(0, str.Length - 2) + "}\r\n";
        }
        else
        {
            if (ПЕРВЫЙ[i, 0] == 0)
            {
                str += "}\r\n";
            }
            else
            {
                for (j = 0; j < ПЕРВЫЙ[i, 0]; ++j)
                {
                    str1 = НетерминалыТерминалы[ПЕРВЫЙ[i, j + 1]];
                    sel[i, ++sel[i, 0]] = k = ind(str1);
                    str1 = str1.Substring(1);
                    str += str1 + "'  ";
                    mark[k] = true;
                }
                str = str.Substring(0, str.Length - 2) + "}\r\n";
            }
        }
        textBox2.AppendText(str);
    }

    // Для того, чтобы это была LL(1) грамматика, значения ВЫБОР
    // для одинаковых порождающих нетерминалов должны быть разными
    for (i = 0; i < n1; ++i)
    {
        str = ПравилаГрамматики[i][0];
        for (j = 0; j < n1; ++j)
        {
            if (i == j)
                continue;
            if (str == ПравилаГрамматики[j][0])
            {
                for (k = 1; k <= sel[i, 0]; ++k)
                {
                    r = sel[i, k];
                    for (x = 1; x <= sel[j, 0]; ++x)
                    {
                        if (r == sel[j, x])
                        {
                            str = "Для нескольких правил с нетерминалом <" +
                                   str.Substring(1) +
                                  ">\nесть дубликатный терминал выбора:";
                            if (r == -1)
                                str += "'КОНЕЦ'";
                            else
```

```
                                str += НетерминалыТерминалы[r].Substring(1) + "'";
                            return str;
                        }
                    }
                }
            }
        }

        return "";
    }

    // Печать сообщения об ошибке программы
    private void ОшибкаТрансляции(string str)
    {
        int i, pos;

        // Установим позицию ошибки не более длины строки
        if (lex_ind >= src_line.Length)
            lex_ind = src_line.Length - 1;

        // Получим позицию начала ошибки
        for (i = pos = 0; i < richTextBox2.Lines.Length && i < record - 1; ++i)
            pos += richTextBox2.Lines[i].Length + 1;

        // Окрашиваем место ошибки
        richTextBox2.SelectionStart = pos + lex_ind;
        richTextBox2.SelectionLength = 1;
        richTextBox2.SelectionColor = Color.Red;

        tabControl1.SelectTab(tabPage2);

        // Печатаем текст ошибки
        MessageBox.Show(str.Substring(1));
    }

    // Контроль дубликатов идентификаторов
    // В случае ошибки возвращается true
    private bool Action0()
    {
        string str = ident.Length > 8 ? ident.Substring(0, 8) : ident;

        // Такой идентификатор уже определен?
        if (table.Contains(str))
        {
            err_semantic = "*Дубликатный идентификатор " + ident;
            return true;
        }

        // Добавляем идентификатор в таблицу идентификаторов
        table.Add(str);
        return false;
    }
```

```
// Контроль определения идентификатора переменной
// В случае ошибки возвращается true
private bool Action1()
{
    string str = ident.Length > 8 ? ident.Substring(0, 8) : ident;

    // Такой идентификатор не определен?
    if (!table.Contains(str))
    {
        err_semantic = "*Неопределенный идентификатор " + ident;
        return true;
    }

    return false;
}

// Контроль пересечений переменных циклов ДЛЯ
// В случае ошибки возвращается true
private bool Action2()
{
    string str = ident.Length > 8 ? ident.Substring(0, 8) : ident;

    // Такой идентификатор уже используется внешним циклом ДЛЯ?
    if (table_for.Contains(str))
    {
        err_semantic = "*Идентификатор " + ident +
                       " уже используется во внешнем цикле ДЛЯ";
        return true;
    }

    // Добавляем идентификатор в таблицу переменных циклов
    table_for.Add(str);
    return false;
}

// Контроль значения константы в диапазоне двух байтов
// В случае ошибки возвращается true
private bool Action3()
{
    if (integer_num > 32767)
    {
        err_semantic = "*Число превышает размер 2 байта";
        return true;
    }
    return false;
}

// Контроль завершения циклов ДЛЯ для удаления переменной цикла
// Всегда возвращается false
private bool Action4()
{
    table_for.RemoveAt(table_for.Count - 1);
    return false;
}
```

```
// Контроль изменения переменной цикла ДЛЯ
// В случае ошибки возвращается true
private bool Action5()
{
    string str = ident.Length > 8 ? ident.Substring(0, 8) : ident;

    // Такой идентификатор уже используется внешним циклом ДЛЯ?
    if (table_for.Contains(str))
    {
        err_semantic = "*Идентификатор " + ident +
                       " нельзя изменять, так как он используется в цикле ДЛЯ";
        return true;
    }
    return false;
}

// Синтаксический разбор
private void Syntax()
{
    string   str, str0, str1, str2;
    int      i, j, k, ind_stack, x, y;
    string[] stack;
    char     c;

    // Была обработка грамматики?
    if (!ХорошаяГрамматика)
    {
        // Открываем закладку с грамматикой
        tabControl1.SelectTab(tabPage1);

        MessageBox.Show("Выполните корректный разбор LL(1) грамматики");
        return;
    }

    // Открываем закладку с программой
    tabControl1.SelectTab(tabPage2);

    // Выполняем полный лексический контроль
    ind_line = -1;
    record = 0;
    yes_comment = false;

    for(;;)
    {
        str = lex();
        if (str == "")
            break;

        if (str[0] == '*')
        {
            ОшибкаТрансляции(str);
            return;
        }
    }

    // Выполняем синтаксический контроль
    ind_line    = -1;
    record      = 0;
    yes_comment = false;
    stack       = new string[100];
    table.Clear();
    table_for.Clear();
```

```
// Помещаем в стек НАЧАЛО и стартовый нетерминал
stack[0]  = "НАЧАЛО";
stack[1]  = "<" + НетерминалыТерминалы[0].Substring(1) + ">";
ind_stack = 1;

// Разбор программы
if ((str = lex()) == "")
{
    MessageBox.Show("Программа не задана");
    return;
}

for (;;)
{
    // Верхний элемент стека
    str1 = stack[ind_stack];

    // Программа не содержит ошибок, если прочитан конец
    // программы и стек содержит НАЧАЛО
    if (str == "")
    {
        if (str1 == "НАЧАЛО")
        {
            MessageBox.Show("Программа не содержит ошибок");
            return;
        }

        // Если вершина стека содержит терминал и
        // прочитан конец программы, это ошибка
        if (str1[0] == '\'')
            break;

        // Номер столбца таблицы действий
        x = ТерминалыМПА.Length - 1;
    }
    else
    {
        // Вершина стека содержит терминал?
        // Тогда он должен соападать с текущей прочитанной лексемой
        if (str1[0] == '\'')
        {
            str2 = str.Substring(1) + "'";
            if (str1 != str2)
                break;

            // Операция ВЫТОЛКНУТЬ,СДВИГ
            --ind_stack;
            str = lex();
            continue;
        }

        // Для текущей лексемы найдем номер столбца таблицы действий
      for (x = 0; x < ТерминалыМПА.Length - 1 && ТерминалыМПА[x] != str; ++x)
            ;

        // Если не нашли, ошибка
        if (x == ТерминалыМПА.Length - 1)
            break;
    }
```

```
// Поиск номера строки таблицы действий для нетерминала,
// находящегося на вершине стека; кроме того
// выполняется возможная семантика
str0 = str1.Substring(1);
str0 = str0.Substring(0, str0.Length - 1);

// Есть для этого нетерминала семантика?
for (j = 0; j < cnt_semantic && semantic[j] != str0; ++j)
    ;

// Да, есть - выполняем семантику
if (j != cnt_semantic)
{
    for (k = 0; k < actions[j, 0]; ++k)
    {
        switch (actions[j, k + 1])
        {
            case 0:
                if (Action0())
                {
                    ОшибкаТрансляции(err_semantic);
                    return;
                }
                continue;
            case 1:
                if (Action1())
                {
                    ОшибкаТрансляции(err_semantic);
                    return;
                }
                continue;
            case 2:
                if (Action2())
                {
                    ОшибкаТрансляции(err_semantic);
                    return;
                }
                continue;
            case 3:
                if (Action3())
                {
                    ОшибкаТрансляции(err_semantic);
                    return;
                }
                continue;
            case 4:
                if (Action4())
                {
                    ОшибкаТрансляции(err_semantic);
                    return;
                }
                continue;
            case 5:
                if (Action5())
                {
                    ОшибкаТрансляции(err_semantic);
                    return;
                }
                continue;
        }
    }
}
```

```
            str0 = "N" + str0;
            y = ind(str0);
            if (y >= Высота)
                y = Высота - 1;

            // Получаем действие
            str0 = Действия[y, x];

            // Ошибка?
            if (str0 == "ОТВ")
                break;

            // Операция ВЫТОЛКНУТЬ,СДВИГ?
            if (str0 == "ВЫТ,СДВ")
            {
                --ind_stack;
                str = lex();
                continue;
            }

            // Операция ВЫТОЛКНУТЬ,ДЕРЖАТЬ?
            if (str0 == "ВЫТ,ДЕР")
            {
                --ind_stack;
                continue;
            }

            // Выполняем замену верхнего нетерминала стека
            --ind_stack;
            for (i = 3; str0[i] != '\t'; ++i)
            {
                // Формируем очередной элемент стека
                for (str2 = ""; (c = str0[i]) != '\n'; ++i)
                    str2 += c.ToString();

                stack[++ind_stack] = str2;
            }

            // Выполнять СДВИГ?
            if (str0.Substring(0, 3) == "СДВ")
                str = lex();
        }

        ОшибкаТрансляции("*Синтаксическая ошибка");
    }

    private void синтаксическийРазборToolStripMenuItem_Click_1
                 (object sender, EventArgs e)
    {
        Syntax();
    }

    // Сброс ошибки
    private void СбросОшибки(ref RichTextBox p)
    {
        int i;
        string[] str = new string[p.Lines.Length];

        for (i = 0; i < str.Length; ++i)
            str[i] = p.Lines[i];
        p.Clear();
```

```
    for (i = 0; i < str.Length; ++i)
    {
        if (i + 1 != str.Length)
            p.AppendText(str[i] + "\r\n");
        else
            p.AppendText(str[i]);
    }
    p.SelectionStart = 0;
}

private void сбросОшибкиToolStripMenuItem_Click(object sender, EventArgs e)
{
    СбросОшибки(ref richTextBox1);
    СбросОшибки(ref richTextBox2);
}

private void загрузитьПрограммуГрамматику(ref RichTextBox p)
{
    if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
        return;
    try
    {
        using (StreamReader sr = new StreamReader(openFileDialog1.FileName,
                            Encoding.GetEncoding(1251)))
        {
            string line;

            p.Clear();

            while ((line = sr.ReadLine()) != null)
            {
                p.AppendText(line + "\r\n");
            }
        }
    }
    catch (Exception)
    {
        MessageBox.Show("Ошибка чтения файла");
    }
}

private void сохранитьПрограммуГрамматику(ref RichTextBox p)
{
    if (saveFileDialog1.ShowDialog() == DialogResult.Cancel)
        return;
    try
    {
        using (StreamWriter sw = new StreamWriter(saveFileDialog1.FileName,
                             false, Encoding.GetEncoding(1251)))
        {
            string line;

            int i;

            for (i = 0; i < p.Lines.Length; ++i)
            {
                line = p.Lines[i];
                sw.WriteLine(line);
            }
        }
    }
```

```
        catch (Exception)
        {
            MessageBox.Show("Ошибка записи файла");
        }
    }

    private void загрузитьГраммматикуToolStripMenuItem_Click(object sender, EventArgs e)
    {
        загрузитьПрограммуГрамматику(ref richTextBox1);
    }

    private void загрузитьПрограммуToolStripMenuItem_Click(object sender, EventArgs e)
    {
        загрузитьПрограммуГрамматику(ref richTextBox2);
    }

    private void сохранитьГраммматикуToolStripMenuItem_Click(object sender, EventArgs e)
    {
        сохранитьПрограммуГрамматику(ref richTextBox1);
    }

    private void сохранитьПрограммуToolStripMenuItem_Click(object sender, EventArgs e)
    {
        сохранитьПрограммуГрамматику(ref richTextBox2);
    }

    private void выходToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Close();
    }

    private void оПрограммеToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Form2 dlg = new Form2();
        dlg.ShowDialog();
    }

    private void авторToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Form3 dlg = new Form3();
        dlg.ShowDialog();
    }

    // Построить таблицу МПА
    private void МПА()
    {
        int i, j, k, n, n1, n2, i1, НомераПроцедур;
        string str, Нетерминал, Реверс, str1, str2;
        bool yes;
        string[] Процедуры = new string[100];

        // Построить заготовку таблицы МПА
        ЗаготовкаТаблицыМПА();

        НомераПроцедур = 0;
```

```
// Просмотр всех строк таблицы (по количеству нетерминалов)
for (i = 0; НетерминалыТерминалы[i][0] == 'N'; ++i)
{
    // Текущий нетерминал
    Нетерминал = НетерминалыТерминалы[i];

    // Просмотр в строке всех терминалов
    for (j = 0; j < ТерминалыМПА.Length; ++j)
    {
        str = ТерминалыМПА[j];

        // Проход по всем правилам, где встречается текущий нетерминал
        yes = false;
        for (i1 = 0; i1 < КоличествоПравилГрамматики; ++i1)
        {
            if (ПравилаГрамматики[i1][0] != Нетерминал)
                continue;

            // Количество элементов ВЫБОР нетерминала
            n = sel[i1, 0];

            if (str == " КОНЕЦ")
            {
                if (sel[i1, n] == -1)
                {
                    dataGridView8.Rows[i].Cells[j].Value = "ВЫТ,ДЕР";
                    Действия[i, j] = "ВЫТ,ДЕР";
                    yes = true;
                    break;
                }
                continue;
            }

            // Индекс терминала
            n1 = ind(str);

            // Содержится ли этот терминал во множестве ВЫБОР нетерминала?
            for (k = 1; k <= n && sel[i1, k] != n1; ++k)
                ;
            // Содержится
            if (k <= n)
            {
                yes = true;

                // Это аннулирующее правило?
                if (ПравилаГрамматики[i1][1] == "#")
                {
                    dataGridView8.Rows[i].Cells[j].Value = "ВЫТ,ДЕР";
                    Действия[i, j] = "ВЫТ,ДЕР";
                    yes = true;
                    break;
                }
                // Правило нетерминала начинается с терминала?
                if (ПравилаГрамматики[i1][1][0] == 'T')
                {
                    // Терминал только один?
                    if (ПравилаГрамматики[i1].Length == 2)
                    {
                        dataGridView8.Rows[i].Cells[j].Value = "ВЫТ,СДВ";
                        Действия[i, j] = "ВЫТ,СДВ";
                        break;
                    }
```

```
                        // Переписать правило в обратном
                        // порядке без первого терминала
                        Реверс = "";
                        str1 = "СДВ";
                        for (n2 = ПравилаГрамматики[i1].Length; --n2 != 1;)
                        {
                            str = ПравилаГрамматики[i1][n2];
                            if (str[0] == 'N')
                                str = "<" + str.Substring(1) + ">";
                            else
                                str = str.Substring(1) + "'";
                            Реверс += " " + str;
                            str1 += str + "\n";
                        }
                        str1 += "\t";
                        str2 = "), СДВИГ";
                    }
                    else
                    {
                        // Переписать правило в обратном порядке целиком
                        Реверс = "";
                        str1 = "ДЕР";
                        for (n2 = ПравилаГрамматики[i1].Length; --n2 != 0;)
                        {
                            str = ПравилаГрамматики[i1][n2];
                            if (str[0] == 'N')
                                str = "<" + str.Substring(1) + ">";
                            else
                                str = str.Substring(1) + "'";
                            Реверс += " " + str;
                            str1 += str + "\n";
                        }
                        str1 += "\t";
                        str2 = "), ДЕРЖАТЬ";
                    }

                    Реверс = Реверс.Substring(1);
                    Реверс = "ЗАМЕНИТЬ(" + Реверс + str2;

                    // Такая процедура уже была?
                for (n2 = 0; n2 < НомераПроцедур && Процедуры[n2] != Реверс; ++n2)
                        ;

                    // Не было, добавим ее в список
                    if (n2 == НомераПроцедур)
                        Процедуры[НомераПроцедур++] = Реверс;

                dataGridView8.Rows[i].Cells[j].Value = "#" + (n2 + 1).ToString();
                    Действия[i, j] = str1;
                    break;
                }
            }

            if (!yes)
            {
                dataGridView8.Rows[i].Cells[j].Value = "ОТВ";
                Действия[i, j] = "ОТВ";
            }
        }
    }
```

```
            // Последняя строка
            for (j = 0; j < ТерминалыМПА.Length - 1; ++j)
            {
                dataGridView8.Rows[i].Cells[j].Value = "ОТВ";
                Действия[i, j] = "ОТВ";
            }
            dataGridView8.Rows[i].Cells[j].Value = "ДОП";
            Действия[i, j] = "ДОП";

            str = "\r\nПроцедуры обработки синтаксического анализа\r\n";
            for (i = 0; i < НомераПроцедур; ++i)
                str += "#" + (i + 1).ToString() + " " + Процедуры[i] + "\r\n";
            textBox2.AppendText(str);
        }

        // Удаление всех строк и столбцов матрицы
        private void УдалитьМатрицу (ref DataGridView p)
        {
            p.Rows.Clear();
            int count = p.Columns.Count;
            for (int i = 0; i < count; ++i)
                p.Columns.RemoveAt(0);
        }

        // Контроль грамматики
        // Грамматика может содержать комментарийные строки,
        // начинающиеся символом ';', а также пустые строки
       private void контрольГрамматикиToolStripMenuItem_Click(object sender, EventArgs e)
        {
            int i, pos;
            string str;

            // Убираем предыдущее решение
            textBox1.Clear();
            textBox2.Clear();
            richTextBox3.Clear();

            dataGridView1.Visible = false;
            dataGridView2.Visible = false;
            dataGridView3.Visible = false;
            dataGridView4.Visible = false;
            dataGridView5.Visible = false;
            dataGridView6.Visible = false;
            dataGridView7.Visible = false;
            dataGridView8.Visible = false;

            УдалитьМатрицу(ref dataGridView1);
            УдалитьМатрицу(ref dataGridView2);
            УдалитьМатрицу(ref dataGridView3);
            УдалитьМатрицу(ref dataGridView4);
            УдалитьМатрицу(ref dataGridView5);
            УдалитьМатрицу(ref dataGridView6);
            УдалитьМатрицу(ref dataGridView7);
            УдалитьМатрицу(ref dataGridView8);

            // Открываем закладку с грамматикой
            tabControl1.SelectTab(tabPage1);

            ЕстьЦелое        = ЕстьВещественное = ЕстьИдентификатор = ЕстьСимвол =
                               ЕстьСтрока = ХорошаяГрамматика = false;
            КоличествоЛексем = cnt_semantic = 0;
```

```
            // По умолчанию в программе не допускаются комментарии
          type_comment[0]  = type_comment[1] = type_comment[2] = type_comment[3] = false;

            // По умолчанию в программе идентификаторы из английских букв
            type_ident = 1;

            for (МаксДлинаПравила = КоличествоПравилГрамматики = НомерСтрокиГрамматики =
0; НомерСтрокиГрамматики < richTextBox1.Lines.Length; ++НомерСтрокиГрамматики)
            {
                // Очередная строка грамматики
                ТекущаяСтрокаГрамматики = richTextBox1.Lines[НомерСтрокиГрамматики];

                // Игнорируем пустую строку
                if (String.IsNullOrWhiteSpace(ТекущаяСтрокаГрамматики))
                    continue;

                // Игнорируем комментарии грамматики, но в комментариях
                // может задаваться тип комментария транслируемой
                // программы или/и состав элементов идентификаторов:
                // #КОММЕНТАРИЙ // /* (* {
                // #ИДЕНТИФИКАТОР РУССКИЙ АНГЛИЙСКИЙ РУССКИЙ_АНГЛИЙСКИЙ
                for (i = 0; i < ТекущаяСтрокаГрамматики.Length &&
                        Char.IsWhiteSpace(ТекущаяСтрокаГрамматики[i]); ++i)
                    ++i;
                if (ТекущаяСтрокаГрамматики[i] == ';')
                {
                    string t = ТекущаяСтрокаГрамматики.Substring(i + 1);

                    // Контроль действий, выполняемых при
                    // обработке некоторого синтаксического правила
                    string[] Действие =
{ "$0 <", "$1 <", "$2 <", "$3 <", "$4 <", "$5 <", "$6 <", "$7 <", "$8 <", "$9 <" };
                    int ind, ind2 = 0;

                    for (ind = 0;
ind < Действие.Length && (ind2 = t.IndexOf(Действие[ind])) < 0; ++ind)
                        ;
                    if (ind != Действие.Length)
                    {
                        int k2, k3;

                        // Пытаемся найти порождающий нетерминал,
                        // для которого задано семантическое действие
                        t = t.Substring(ind2 + Действие[ind].Length);
                        if ((ind2 = t.IndexOf(">")) < 0)
                            continue;

                        // Нетерминал с семантикой
                        t = t.Substring(0, ind2);

                        // Для этого нетерминала уже была какая-то семантика?
                        for (k2 = 0; k2 < cnt_semantic && semantic[k2] != t; ++k2)
                            ;

                        // Да, такое уже было
                        if (k2 != cnt_semantic)
                        {
                            // Дубликат семантики и слишком много семантики игнорируется
                    for (k3 = 0; k3 < actions[k2, 0] && ind != actions[k2, k3 + 1]; ++k3)
                                ;
                            if (k3 != actions[k2, 0] || k3 == MAX_SEMANTIC)
                                continue;
```

```
                    // Добавляем очередную семантику
                    actions[k2, ++actions[k2, 0]] = ind;
                    continue;
                }

                // Есть место для очередного нетерминала с семантикой?
                if (cnt_semantic != NOTERMINAL_SEMANTIC)
                {
                    // Сохраняем нетерминал
                    semantic[cnt_semantic] = t;

                    // Устанавливаем количество семантик
                    actions[cnt_semantic, 0] = 1;

                    // Сохраняем семантику
                    actions[cnt_semantic, 1] = ind;
                    ++cnt_semantic;
                }
                continue;
            }

            if (t.Contains("#КОММЕНТАРИЙ //"))
                type_comment[0] = true;
            if (t.Contains("#КОММЕНТАРИЙ /*"))
                type_comment[1] = true;
            if (t.Contains("#КОММЕНТАРИЙ (*"))
                type_comment[2] = true;
            if (t.Contains("#КОММЕНТАРИЙ {"))
                type_comment[3] = true;
            if (t.Contains("#ИДЕНТИФИКАТОР РУССКИЙ_АНГЛИЙСКИЙ"))
                type_ident = 2;
            else if (t.Contains("#ИДЕНТИФИКАТОР РУССКИЙ"))
                type_ident = 0;
            else if (t.Contains("#ИДЕНТИФИКАТОР АНГЛИЙСКИЙ"))
                type_ident = 1;
            continue;
        }
        НомерСимволаГрамматики = i;
        // Контроль строки грамматики
        str = КонтрольСтрокиГрамматики();
        // Есть ошибка?
        if (str != "")
        {
            // Установим позицию ошибки не более длины строки
            if (НомерСимволаЭлементаГрамматики >= ТекущаяСтрокаГрамматики.Length)
            НомерСимволаЭлементаГрамматики = ТекущаяСтрокаГрамматики.Length - 1;

            // Получим позицию начала ошибки
            for (i = pos = 0; i < richTextBox1.Lines.Length &&
                        i < НомерСтрокиГрамматики; ++i)
                pos += richTextBox1.Lines[i].Length + 1;

            // Окрашиваем место ошибки
            richTextBox1.SelectionStart = pos + НомерСимволаЭлементаГрамматики;
            richTextBox1.SelectionLength = 1;
            richTextBox1.SelectionColor = Color.Red;

            // Печатаем текст ошибки
            MessageBox.Show(str.Substring(1));
            return;
        }
    }
```

```
if (КоличествоПравилГрамматики == 0)
{
    MessageBox.Show("Грамматика не определена");
    return;
}

СортировкаЛексем();

// Массив правил грамматики
ПравилаГрамматики = new string[КоличествоПравилГрамматики][];

// Формируем массив правил грамматики
for (КоличествоПравилГрамматики = НомерСтрокиГрамматики = 0;
 НомерСтрокиГрамматики < richTextBox1.Lines.Length; ++НомерСтрокиГрамматики)
{
    // Очередная строка грамматики
    ТекущаяСтрокаГрамматики = richTextBox1.Lines[НомерСтрокиГрамматики];

    // Игнорируем пустую строку
    if (String.IsNullOrWhiteSpace(ТекущаяСтрокаГрамматики))
        continue;

    // Игнорируем комментарии
    for (i = 0; i < ТекущаяСтрокаГрамматики.Length &&
          Char.IsWhiteSpace(ТекущаяСтрокаГрамматики[i]); ++i)
        ++i;
    if (ТекущаяСтрокаГрамматики[i] == ';')
        continue;
    НомерСимволаГрамматики = i;

    СохранениеПравилГрамматики();
}

// Получаем список всех всех нетерминалов и терминалов
ВсеНетерминалыТерминалы();

// Делаем видимыми таблицы
dataGridView1.Visible = true;
dataGridView2.Visible = true;
dataGridView3.Visible = true;
dataGridView4.Visible = true;
dataGridView5.Visible = true;
dataGridView6.Visible = true;
dataGridView7.Visible = true;
dataGridView8.Visible = true;

// Поиск epsilon-порождающих нетерминалов
Эпсилон();

// Отношение НПС
НПС();

// Отношение НС
НС();

// Отношение ПП
ПП();

// Отношение ПНК
ПНК();
```

```
            // Отношение НК
            НК();

            // Отношение П
            П();

            // Отношение П+
            ПL();

            // Вычисление множества ПЕРВ
            ПЕРВ();

            // Вычисление множества СЛЕД
            СЛЕД();

            // Вычисление множества ВЫБОР
            if ((str = ВЫБОР()) == "")
            {
                // Построить таблицу МПА
                МПА();

                ХорошаяГрамматика = true;
                MessageBox.Show("Грамматика корректна и является LL(1)-грамматикой");
            }
            else
            {
                dataGridView8.Visible = false;
                MessageBox.Show
("Грамматика корректна, но не является LL(1)-грамматикой\n" + str);
            }
        }
    }
}
```