НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ
КАФЕДРА МАТЕМАТИЧЕСКОГО И КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ

Курсовая работа
по дисциплине “МО ЭВМ”
“Построение транслятора”

Выполнил студент гр. А-14-20
Фоминых. Д .А
Принял преподаватель Князев. А. В

Москва 2023

1. **Индивидуальное задание:**

Оператор присваивания:
<ид.>:=<ар.выр>;

Условный оператор:
if(<лог.выр.>) <оператор> [else <оператор>]

Оператор цикла:
repeat <совок. операторов> until <лог.выр.>;

Арифметическое выражение:
<E> ::= <T> <E-список>
<E-список> ::= + <T> <E-список>
<E-список> ::= !
<T> ::= <F> <T-список>
<T-список> ::= *<F> <T-список>
<T-список> ::= !
<F>::=<Id>
<F>::=<Int>

Логическое выражение:
<лог.выр.>::=<F><лог.опер.><F>
<лог.опер.>::= =
<лог.опер.>::= !=

Пример программы:

```
a:=16*3+1;
b:=11+2*a;
c:=3*a+2;
if(b!=c) a:=4*b; else a:=2*b+3;
k:=0; s:=0;
repeat
  k:=k+1;
  s:=s+1;
until k=10;
```

2. Преобразовать заданную грамматику в LL(1)-грамматику:

1<Программа> -> < Оператор >< Совокупность операторов >
2 < Программа > -> !
3< Оператор > -> <Id> := <E>;
4<Оператор> -> if (<логическое выражение>) <оператор> <конец if>
5 <Оператор> -> repeat <Совокупность операторов> until <логическое выражение>;
6 < Совокупность операторов > ->< Оператор >< Совокупность операторов>
7 < Совокупность операторов > -> !
8 <конец if> -> else <оператор>
9 <конец if> -> !

10 <E > -> <T><E-список>
11<E-список> -> +<T><E-список>
12 <E-список> -> !
13 <T> -> <F><T-список>
14 <T-список> -> *<F><T-список>
15 <T-список> ->!
16 <F> -> <Id>
17 <F> -> <Int>

18<логическое выражение> -> <F><логический оператор>
19 <логический оператор> -> =<F>
20<логический оператор>-> !=<F>

3. **Формальное построение множеств выбора:**

Аннулирующие нетерминалы:

<Программа>,< Совокупность операторов >,<E-список>,<T-список>,<конец if>

Аннулирующие правила: 2,7, 9, 12, 15.

Начинается Прямо С

1	Id	int	if	until	else	+	*	(	)	;	:=	=	!=	repeat	оператор	совокупность операторов	Конец if	E	E-список	T	T-список	F	Логическая операция	Логическое выражение
Оператор	1		1											1										
Совокупность операторов															1									
Конец if					1																			
E																				1				
E-список						1																		
T																						1		
T-список							1																	
F	1	1																						
Логическое выражение																						1		
Логическая операция												1	1											
Программа															1									

Начинается С

1	Id	int	if	until	else	+	*	(	)	;	:=	=	!=	repeat	оператор	совокупность операторов	Конец if	E	E-список	T	T-список	F	Логическая операция	Логическое выражение
Оператор	1		1											1										
Совокупность операторов	1		1											1	1									
Конец if					1																			
E	1	1																		1		1		
E-список						1																		
T	1	1																				1		
T-список							1																	
F	1	1																						
Логическое выражение	1	1																				1		
Логическая операция												1	1											
Программа	1		1											1	1									

Построение множества Перв для нетерминалов

1. Перв{<Программа>} = {Id,repeat,if}
2. Перв{<Оператор>} = {Id ,if, repeat}
3. Перв{<Совокупность операторов>} = {Id,if,repeat}
4. Перв{<Конец if>} = {else}
5. Перв{<E>} = {Id,Int}
6. Перв{<E-спис>} = {+}
7. Перв{<T>} = {Id,Int}
8. Перв{<T-список>} = {*}
9. Перв{<F>} = {Id,Int}
10. Перв{<Логическое выражение>} = {Id,Int}
11. Перв{<Логическая операция>} = {=,!=}

Построение множества Перв для правил

1. Перв{1} = {Id,repeat,if}
2. Перв{3} = {Id,Int}
3. Перв{4} = {Id,if,repeat,Int,else}
4. Перв{5} = {Id,Int,if,repeat}
5. Перв{6} = {Id,repeat,if}
6. Перв{8} = {Id,repeat,if}
7. Перв{10} = {Id,Int,+}
8. Перв{11} = {Id,Int,+}
9. Перв{13} = {Id,Int,*}
10. Перв{14} = {Id,Int,*}
11. Перв{16} = {Id}
12. Перв{17} = {Int}
13. Перв{18} = {Int,Id,=,!=}
14. Перв{19} = {Id,Int}
15. Перв{20} = {Id,Int}
16. Перв{2} = Перв{7} = Перв{9} = Перв{12} = Перв{15} = {}

Прямо Перед

3	Id	int	if	until	else	+	*	(	)	;	:=	=	!=	repeat	оператор	совокупность операторов	Конец if	E	E-список	T	T-список	F	Логическая операция	Логическое выражение
Оператор					1				1															
Совокупность операторов														1	1									
Конец if															1									
E											1													
E-список																				1				
T						1																		
T-список																						1		
F							1						1	1										
Логическое выражение				1				1																
Логическая операция																						1		
Программа																								
Id																								
int																								
if																								
until																1								
else																								
+																								
*																								
(			1																					
)																								1
;																		1						1
:=	1																							
=																								
!=																								

Прямо на Конце

4	Id	int	if	until	else	+	*	(	)	;	:=	=	!=	repeat	оператор	совокупность операторов	Конец if	E	E-список	T	T-список	F	Логическая операция	Логическое выражение	Программа
Оператор																1									1
Совокупность операторов																									
Конец if																									
E																									
E-список																									
T																	1								
T-список																									
F																			1					1	
Логическое выражение																									
Логическая операция																									
Программа																									
Id															1						1				
int																					1				
if															1										
until																									
else																	1								
+																			1						
*																					1				
(																									
)																									
;																									
:=																									
=																							1		
!=																							1		
repeat															1										

На конце

5	Id	int	if	until	else	+	*	(	)	;	:=	=	!=	repeat	оператор	совокупность операторов	Конец if	E	E-список	T	T-список	F	Логическая операция	Логическое выражение	Программа
Оператор															1	1									1
Совокупность операторов																1									
Конец if																	1								
E																		1							
E-список																			1						
T																		1		1					
T-список																					1				
F																			1		1		1		
Логическое выражение																								1	
Логическая операция																							1		
Программа																									1
Id	1														1						1				
int		1																			1				
if			1												1										
until				1																					
else					1												1								
+						1													1						
*							1														1				
(								1																	
)									1																
;										1															
:=											1														
=												1											1		
!=													1										1		
repeat														1	1										

Перед

	Id	int	if	until	else	+	*	(	)	;	:=	=	!=	repeat	оператор	совокупность операторов	Конец if	E	E-список	T	T-список	F	Логическая операция	Логическое выражение	Программа
Оператор	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Совокупность операторов	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Конец if	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E-список	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T-список	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Логическое выражение	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Логическая операция	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Программа	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Id	1	1	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
int	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
if	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
until	1	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
else	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
+	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
)	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0
;	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
:=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
!=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
repeat	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Множество След() для аннулирующих нетерминалов:
След(<Программа>) = {⊥}
След(<Сов. оп.>) = {⊥}
След(<E-сп>) = {;, ⊥}
След(<T-сп>) = {;, ⊥}
След(<конец if>) = { <Id>,if,repeat,⊥}

Множество выбора для LL(1)-грамматики:
Обозначение: ~ это концевой маркер.
Выбор(1)={<Id>,if,repeat}
Выбор(2)= {-|}
Выбор(3)= {<Id>}
Выбор(4)= {if}
Выбор(5)= {repeat}
Выбор(6)={<Id>,if,repeat}
Выбор(7)= {-|}
Выбор(8)= {else}
Выбор(9)= {<Id>,if,repeat,-|,}
Выбор(10)= {<Id>,<Int>}
Выбор(11)= {+}
Выбор(12)= {;,⊥}
Выбор(13)= {<Id>,<Int>}
Выбор(14)= {*}
Выбор(15)= {;, ⊥}
Выбор(16)= {<Id>}
Выбор(17)= {<Int>}
Выбор(18)= {Id, <Int>}
Выбор(19)= {=}
Выбор(20)= {!=}

4. **Транслирующая грамматика:**

1 <Программа> -> < Оператор >< Совокупность операторов >
2 < Программа > -> !
3 < Оператор > -> <Id> := <E>{ := };
4 <Оператор> -> if (<логическое выражение>) {условный переход по нулю}<оператор> <конец if>
5 <Оператор> -> repeat {Метка}<Совокупность операторов>{Безусловный переход}{метка} until <логическое выражение>{Условный переход};
6 < Совокупность операторов > ->< Оператор >< Совокупность операторов>
7 < Совокупность операторов > -> !
8 <конец if> -> else{Безусловный переход}{Метка}<оператор>{Метка}
9 <конец if> -> {Метка} !
10 <E > -> <T><E-список>
11 <E-список> -> +<T>{ + }<E-список>
12 <E-список> -> !
13 <T> -> <F><T-список>
14 <T-список> -> *<F>{ * }<T-список>
15 <T-список>->!
16 <F> -> <Id> { Id }
17 <F> -> <Int> { Int }
18<логическое выражение> -> <F><логический оператор>
19 <логический оператор> -> =<F> { = }
20<логический оператор>-> !=<F> { != }

5. **Управляющая таблица МП-автомата:**

	Id-1	int-2	if-3	until-6	else-4	+ -8	*-9	(-10	)-11	;-7	:=-16	=-14	!=-17	repeat-5	\
Программа	13	err	13	err	err	err	err	err	err	err	err	err	err	13	
Оператор	1	err	2	err	err	err	err	err	err	err	err	err	err	3	Отв
Совокупность операторов	13	err	13	6	5	err	err	err	err	err	err	err	err	13	6
Конец if	4	err	4	4	5	err	err	err	err	err	err	err	err	4	Отв
E	7	7	err	err	err	err	err	err	err	err	err	err	err	err	Отв
E-список	err	err	err	err	err	8	err	err	err	6	err	err	err	err	Отв
T	9	9	err	err	err	err	err	err	err	err	err	err	err	err	Отв
T-список	err	err	err	err	err	6	10	err	err	6	err	err	err	err	Отв
F	11	11	err	err	err	err	err	err	err	err	err	err	err	err	Отв
Логическое выражение	12	12	err	err	err	err	err	err	err	err	err	err	err	err	Отв
Логическая операция	err	err	err	err	err	err	err	err	err	err	err	11	11	err	Отв
:=	err	err	err	err	err	err	err	err	err	err	11	err	err	err	Отв
;	err	err	err	err	err	err	err	err	err	11	err	err	err	err	Отв
(	err	err	err	err	err	err	err	11	err	err	err	err	err	err	Отв
)	err	err	err	err	err	err	err	err	11	err	err	err	err	err	Отв
until	err	err	err	11	err	err	err	err	err	err	err	err	err	err	Отв
Id	11	err	err	err	err	err	err	err	err	err	err	err	err	err	Отв
Int	err	11	err	err	err	err	err	err	err	err	err	err	err	err	Отв
!=	err	err	err	err	err	err	err	err	err	err	err	err	11	err	Отв
=	err	err	err	err	err	err	err	err	err	err	err	11	err	err	Отв
Δ															Доп

6. Атрибутная транслирующая грамматика:

1 <Программа> -> < Оператор >< Совокупность операторов >

2 < Программа > -> !

3 < Оператор > -> <Id>$_{p1}$:= <E>$_{q1}$\{ Присвоить$_{p2,q2}$ };

p2:=p1, q2:= q1

4 <Оператор> -> if (<логическое выражение>$_{t1}$) {условный переход по нулю$_{t2,z1}$}<оператор> <конец if>$_{z2}$

(z1,z2):= НовМет, t2:=t1;

5 <Оператор> -> repeat {Метка}$_{z2}$<Совокупность операторов> until <логическое выражение>$_{p1}$\{Условный переход по 1\}$_{p2,z1}$;

z2:=z1, p2:=p1;

6 < Совокупность операторов > ->< Оператор >< Совокупность операторов>

7 < Совокупность операторов > -> !

8 <конец if>$_{z1}$ -> else\{Безусловный переход$_{w1}$\}\{Метка$_{z2}$\}<оператор>\{Метка$_{w2}$\}

z2:=z1, w2:=w1

9 <конец if>$_{z1}$ -> \{Метка$_{z2}$\} !

10 <E >$_{r2}$ -> <T>$_{p1}$<E-список>$_{p2,r1}$

p2:=p1,t2:=t1

11 <E-список>$_{p1,r3}$ -> +<T>$_{q1}$\{ + \}$_{p2,q2,r1}$<E-список>$_{p3,r2}$

q2:=p1, p2:=q1,(r1,r2):=Нов,r3:=r2

12 <E-список>$_{p1,r2}$ -> !

13 <T>$_{r2}$ -> <F>$_{p1}$<T-список>$_{p2,r1}$

p2:=p1,t2:=t1

14 <T-список>$_{p1,r3}$ -> *<F>$_{q1}$\{ * \}$_{p2,q2,r1}$<T-список>$_{p3,r2}$

q2:=p1, p2:=q1,(r1,r2):=Нов,r3:=r2

15 <T-список>$_{p1,r2}$->!

16 <F>$_{r2}$ -> <Id>$_{r1}$

r2:= r1

17 <F>$_{r2}$ -> <Int>$_{r1}$

r2:= r1

18<логическое выражение>$_{t2}$ -> <F>$_{p1}$<логический оператор>$_{p2,t1}$

p2:=p1, t2:= t1;

19 <логический оператор>$_{p1,t2}$ -> =<F>$_{q1}$ { =$_{p2,q2,t1}$ }

p2:= p1, t2:= t1;

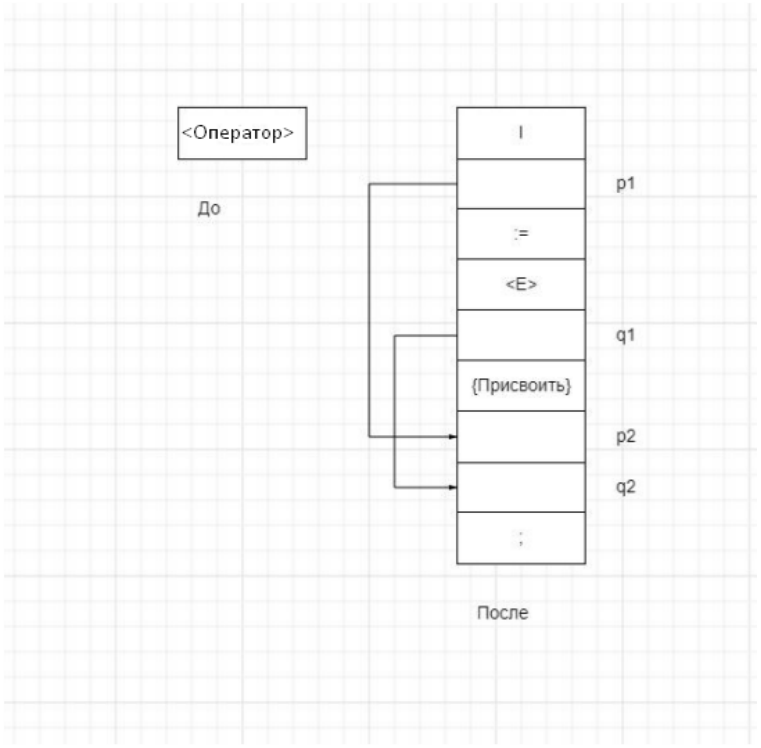
20<логический оператор>$_{p1,t2}$-> !=<F>$_{q1}$ { !=$_{p2,q2,t1}$ }

p2:= p1, t2:= t1;

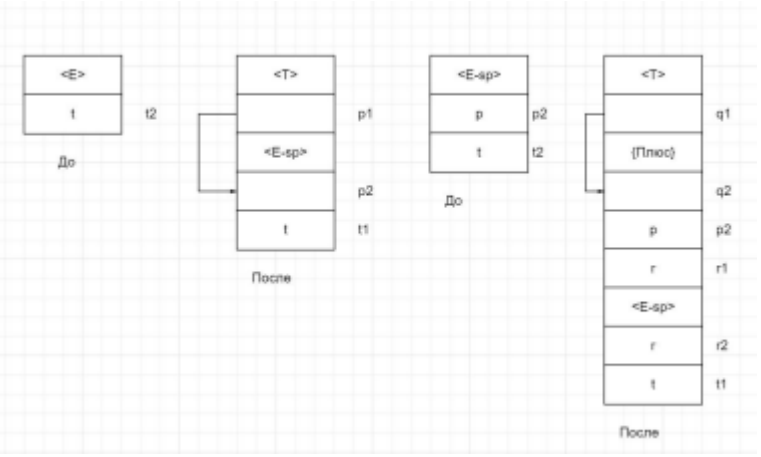
7. Правила замены магазинных символов:

1.Замен(:=<Ариф. выражение>{Присвоить};), Сдвиг
2.Замен((<Логическое выражение>)<Условный переход по 0><Оператор><Конец if>), Сдвиг
3.Замен({Метка}<Совокупность операторов> until <логическое выражение>{Условный переход};), Сдвиг
4.{Метка}Вытолк, Держать
5.Замен(<Безусловный переход><Метка><Оператор><Метка>),Сдвиг
6.Вытолк, Держать
7.Замен (<T><E-список>), Держать
8.Замен(<T>{Сложить}<E-список>), Сдвиг
9.Замен (<F><T-список>), Держать
10.Замен (<F>{Умножить}<T-список>), Сдвиг
11.Вытолк, Сдвиг
12.Замен (<F><Логическая операция><F>{сравнить}), Держать
13.Замен(<Оператор><Совокупность операторов>),Держать
{Присвоить} Действие({Присвоить}),Вытолкнуть,Держать
{Условный переход} Действие({Условный переход}),Вытолкнуть,Держать
{Безусловный переход} Действие({Безусловный переход}),Вытолкнуть,Держать
{Метка} Действие({Метка}),Вытолкнуть,Держать
{Умножить} Действие({Умножить}),Вытолкнуть,Держать
{Сложить} Действие({Сложить}),Вытолкнуть,Держать
{Сравнить} Действие({Сравнить}),Вытолкнуть,Держать

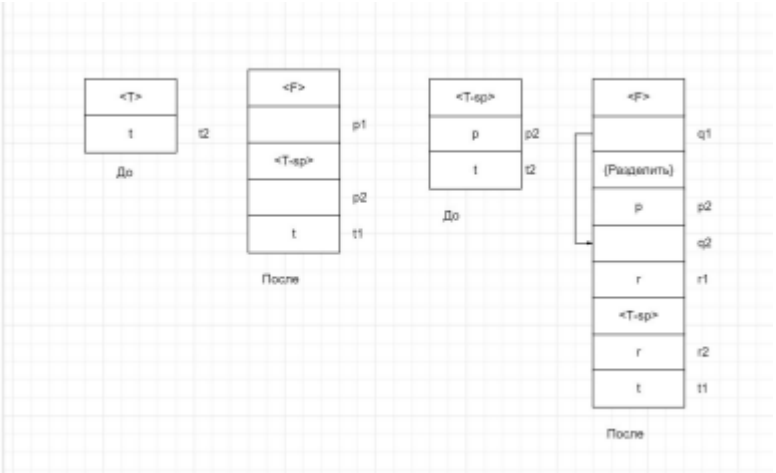
Оператор присваивания



Замена E и E-список



Замена T и T-список



8. **Объектный язык:**

1. Присвоить(v1, v2) – копирует значение из ячейки v1 в v2

2. Сложить(v1, v2, r) – записывает результат сложения значений ячеек v1 и v2 в ячейку r

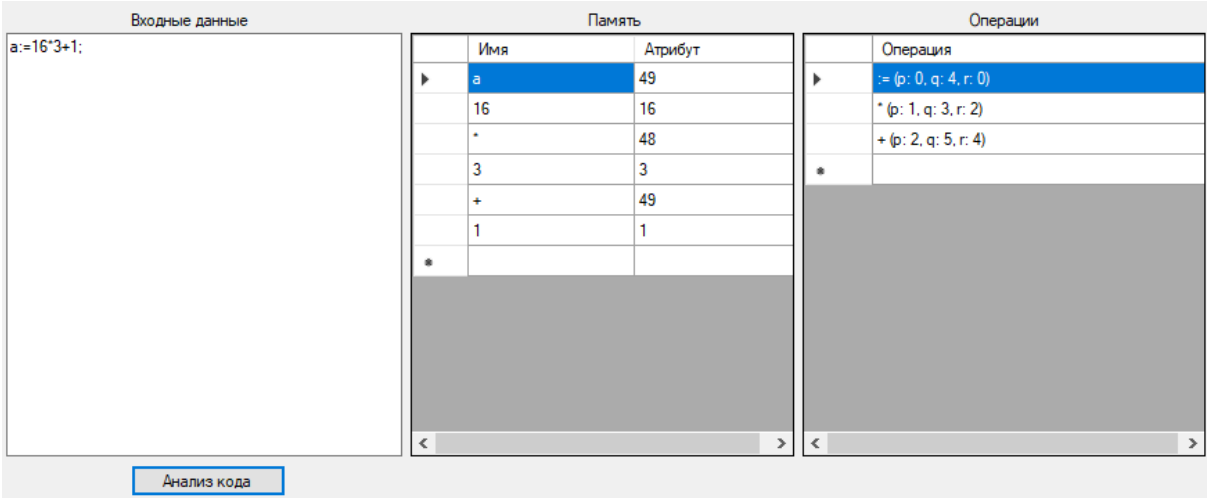
3.Умножить(v1, v2, r) – записывает результат умножения значений ячеек v1 и v2 в ячейку r

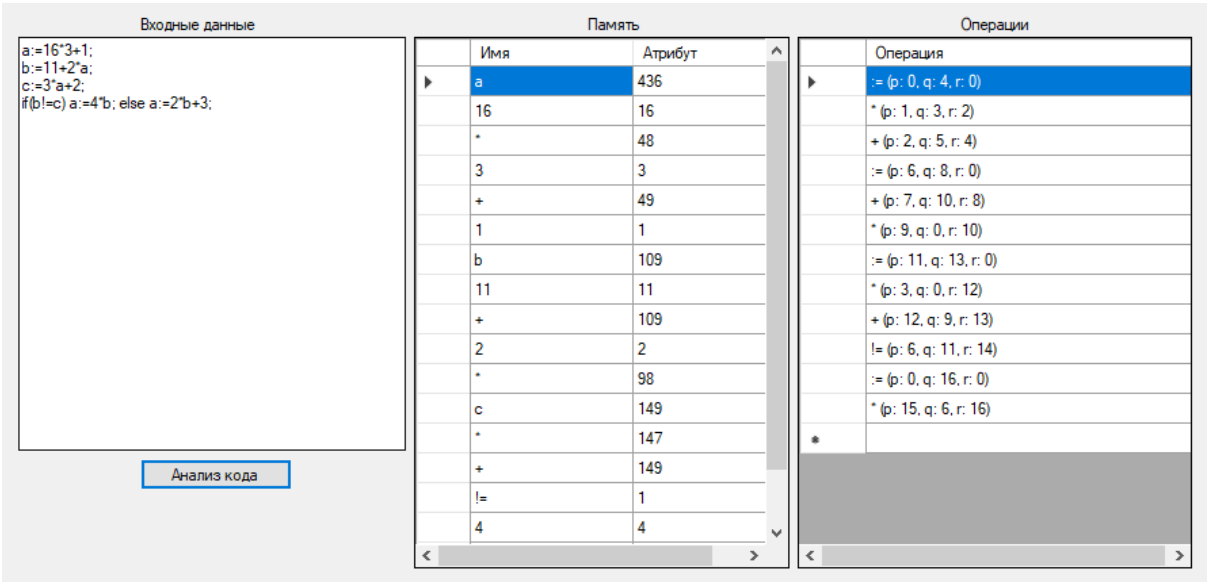
3. Равно(v1, v2, r) – записывает результат сравнения значений ячеек v1 и v2 в ячейку r, если значение первой ячейки равно второй, то записывается true и наоборот

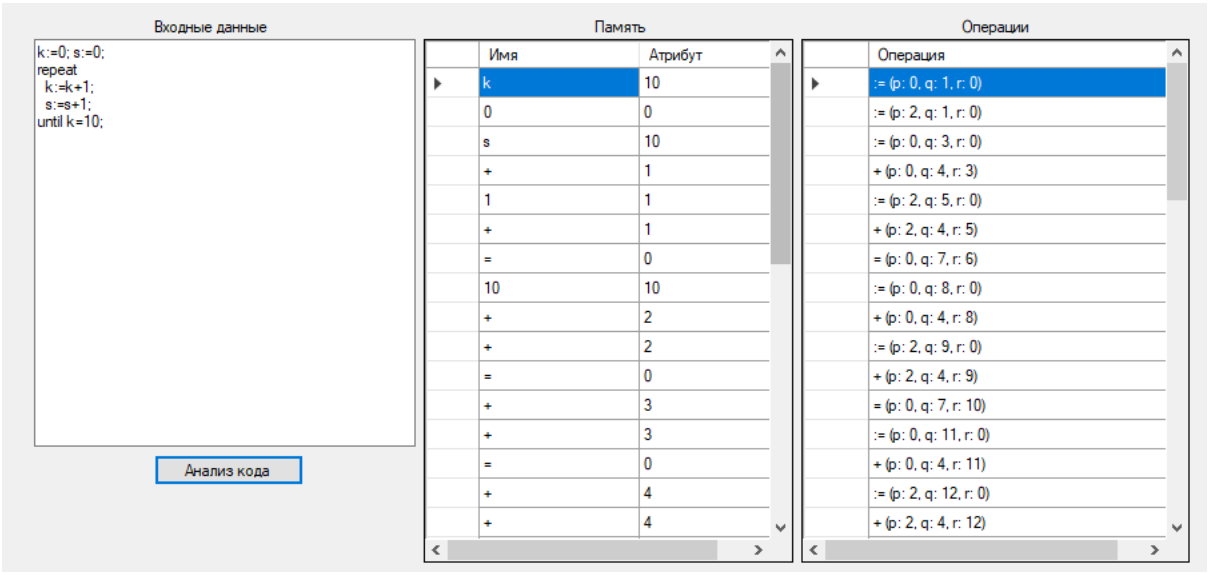
4. Условный переход (if) (v1, num) – если значение ячейки v1 = false, то продолжаем обработку команд с num-той команды

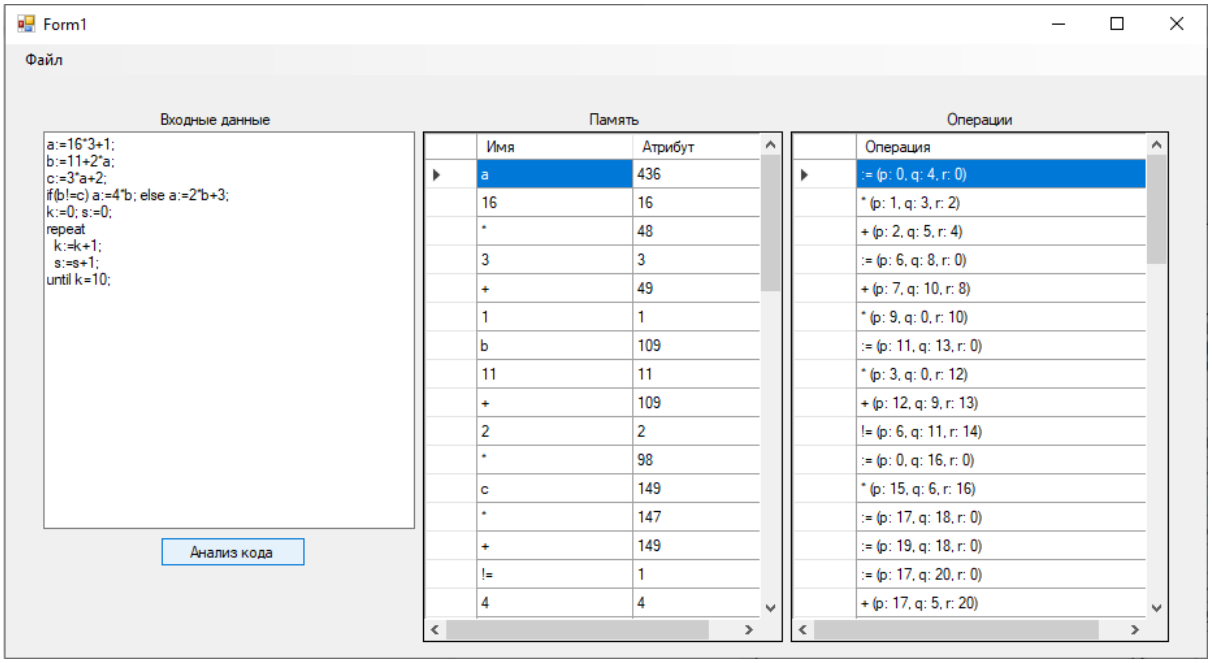
5. Безусловный переход (if) (v1, num) – если значение ячейки v1 = true, то продолжаем обработку команд с num-той команды.

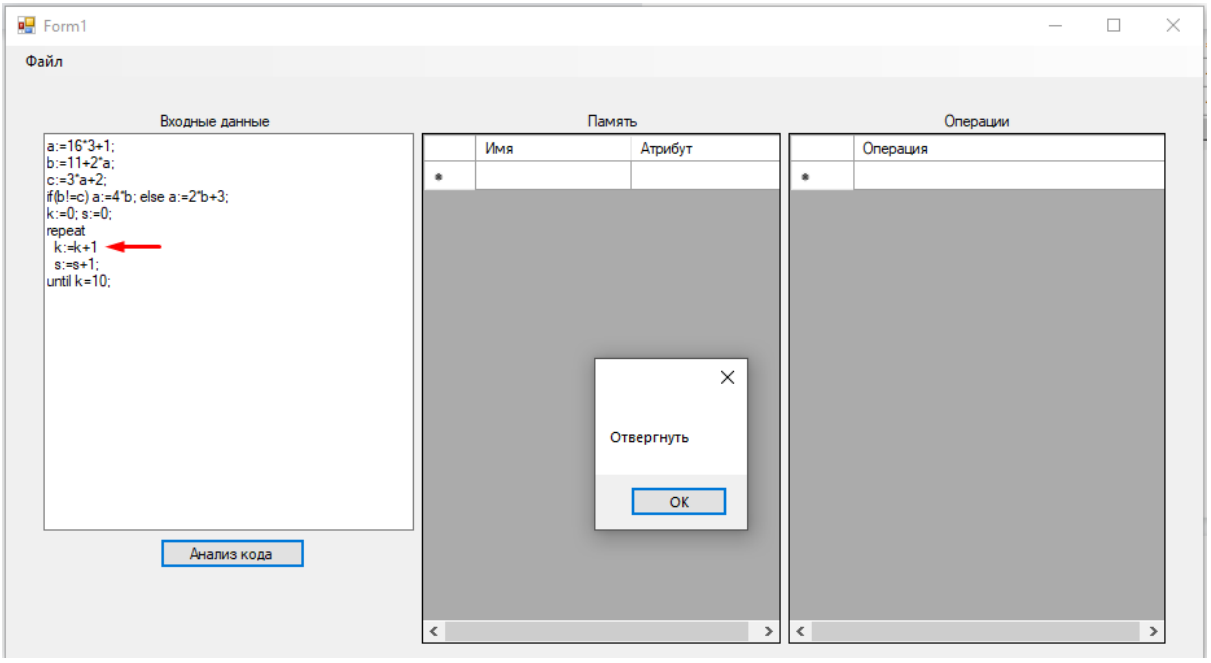
9. **Описание интерфейса программы:**
   Пользователь открывает текстовый файл с кодом, который нужно запустить и нажимает кнопку “Анализ кода”. После чего пользователь видит значения переменных в памяти, а также операции, примененные в вычислениях.
10. **Тесты:**




Входные данные
a:=16*3+1;
b:=11+2*a;
c:=3*a+2;
if(b!=c) a:=4*b; else a:=2*b+3;
Анализ кода
Память
Имя
Атрибут
a 436
16 16
* 48
3 3
+ 49
1 1
b 109
11 11
+ 109
2 2
* 98
c 149
* 147
+ 149
!= 1
4 4
Операции
Операция
:= (p: 0, q: 4, r: 0)
* (p: 1, q: 3, r: 2)
+ (p: 2, q: 5, r: 4)
:= (p: 6, q: 8, r: 0)
+ (p: 7, q: 10, r: 8)
* (p: 9, q: 0, r: 10)
:= (p: 11, q: 13, r: 0)
* (p: 3, q: 0, r: 12)
+ (p: 12, q: 9, r: 13)
!= (p: 6, q: 11, r: 14)
:= (p: 0, q: 16, r: 0)
* (p: 15, q: 6, r: 16)


Входные данные
k:=0; s:=0;
repeat
k:=k+1;
s:=s+1;
until k=10;
Анализ кода
Память
Имя
Атрибут
k 10
0 0
s 10
+ 1
1 1
+ 1
= 0
10 10
+ 2
+ 2
= 0
+ 3
+ 3
= 0
+ 4
+ 4
Операции
Операция
:= (p: 0, q: 1, r: 0)
:= (p: 2, q: 1, r: 0)
:= (p: 0, q: 3, r: 0)
+ (p: 0, q: 4, r: 3)
:= (p: 2, q: 5, r: 0)
+ (p: 2, q: 4, r: 5)
= (p: 0, q: 7, r: 6)
:= (p: 0, q: 8, r: 0)
+ (p: 0, q: 4, r: 8)
:= (p: 2, q: 9, r: 0)
+ (p: 2, q: 4, r: 9)
= (p: 0, q: 7, r: 10)
:= (p: 0, q: 11, r: 0)
+ (p: 0, q: 4, r: 11)
:= (p: 2, q: 12, r: 0)
+ (p: 2, q: 4, r: 12)


Form1
Файл
Входные данные
a:=16*3+1;
b:=11+2*a;
c:=3*a+2;
if(b!=c) a:=4*b; else a:=2*b+3;
k:=0; s:=0;
repeat
k:=k+1;
s:=s+1;
until k=10;
Анализ кода
Память
Имя
Атрибут
a 436
16 16
* 48
3 3
+ 49
1 1
b 109
11 11
+ 109
2 2
* 98
c 149
* 147
+ 149
!= 1
4 4
Операции
Операция
:= (p: 0, q: 4, r: 0)
* (p: 1, q: 3, r: 2)
+ (p: 2, q: 5, r: 4)
:= (p: 6, q: 8, r: 0)
+ (p: 7, q: 10, r: 8)
* (p: 9, q: 0, r: 10)
:= (p: 11, q: 13, r: 0)
* (p: 3, q: 0, r: 12)
+ (p: 12, q: 9, r: 13)
!= (p: 6, q: 11, r: 14)
:= (p: 0, q: 16, r: 0)
* (p: 15, q: 6, r: 16)
:= (p: 17, q: 18, r: 0)
:= (p: 19, q: 18, r: 0)
:= (p: 17, q: 20, r: 0)
+ (p: 17, q: 5, r: 20)


Form1
Файл
Входные данные
a:=16*3+1;
b:=11+2*a;
c:=3*a+2;
if(b!=c) a:=4*b; else a:=2*b+3;
k:=0; s:=0;
repeat
k:=k+1
s:=s+1;
until k=10;
Анализ кода
Память
Имя
Атрибут
Операции
Операция
Отвергнуть
OK

11. **Листинг программы:**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using LexicalAnalyzer;
using ClassID;
using Struct;
namespace SyntacsycalAnalyzer
{
  internal class SyntacticalBlock
  {
    public Stack<string> AnalyzerStack = new Stack<string>(); //правая часть
    public List<TreeNode> lexemList = new List<TreeNode>();
    String[] rules = {";":=","(",")","<Арифм.выражение>","<Логическое выражение>","<Оператор>","<Конец if>",
      "<Совокупность операторов>","until","<T>","<E-список>","<F>","<T-список>","<Логическая операция>"};
    public List<TreeNode> nodeList = new List<TreeNode>();
    public int lastCell = 0;
    public List<Structures> structList = new List<Structures>();
    public int k = 0;
    public int mark = -1;
    public bool jumpFlag = false;
    public int elseFlag = 0;

    public void RefreshTree(Stack<TreeNode> tree1)
    {
      while (tree1.Count != 0)
      {
        TreeNode node =  tree1.Pop();
        lexemList.Insert(0, node);
      }

    }
    public bool SyntacticalAnalyzer()
    {
      AnalyzerStack.Push("The End");
      AnalyzerStack.Push("Программа");
      while (lexemList.Count > k)
      {
        TreeNode node = lexemList[k];
        string magazine = AnalyzerStack.Pop();
        int data = node.Data;

        switch (magazine)
        {
          case ("Программа"):
            {
              switch (data)
              {
                case 1:
                  {
                    Rule13();
                    break;
                  }
                case 3:
                  {
                    Rule13();
                    break;
                  }
                case 5:
```

```
            {
              Rule13();
              break;
            }
            default: return false;
        }
        break;
      }
    case ("<Оператор>"):
      {
        switch (data)
        {
          case 1:
            {
              Rule1();
              if (elseFlag != 1 && !jumpFlag)
              {
                if (nodeList.FindIndex(i => i.Name == node.Name) == -1)
                {
                  nodeList.Add(new TreeNode(node.Name));
                  lastCell = nodeList.Count - 1;
                }
                else
                {
                  lastCell = nodeList.FindIndex(i => i.Name == node.Name);
                }
              }
              k++;
              break;
            }
          case 3:
            {
              Rule2();
              k++;
              break;
            }
          case 5:
            {
              Rule3();
              k++;
              break;
            }
          default: return false;
        }
        break;
      }
    case ("<Совокупность операторов>"):
      {
        switch (data)
        {
          case 1:
            {
              Rule13();
              break;
            }
          case 3:
            {
              Rule13();
              break;
            }
          case 4:
            {
              Rule5();
```

```
                k++;
                break;
            }
          case 5:
            {
                Rule13();
                break;
            }
          case 6:
            {
                Rule6();
                break;
            }
          default: return false;

      }
      break;
    }
case ("<Конец if>"):
    {
      switch (data)
      {
          case 1:
            {
                Rule6();
                break;
            }
          case 3:
            {
                Rule6();
                break;
            }
          case 4:
            {
                Rule5();
                if (elseFlag == 2)
                {
                   elseFlag--;
                }
                jumpFlag = false;
                k++;
                break;
            }
          case 5:
            {
                Rule6();
                break;
            }
          case 6:
            {
                Rule6();
                break;
            }
          default: return false;
      }
      break;
    }
case ("<Арифм.выражение>"):
    {
      switch (data)
      {
          case 1:
            {
```

```
                Rule7();
                break;
            }
          case 2:
            {
                Rule7();
                break;
            }
          default: return false;
        }
        break;
      }
    case ("<E-список>"):
      {
        switch(data)
        {
          case 7:
            {
                Rule6();
                break;
            }
          case 8:
            {
                Rule8();
                if (elseFlag != 1 && !jumpFlag)
                {
                   nodeList.Add(new TreeNode("+"));
                   structList.Add(new Structures());
                   structList[structList.Count - 1].r = nodeList.Count - 1;
                   structList[structList.Count - 1].p = lastCell;
                   structList[structList.Count - 1].name = "+";
                }
                k++;
                break;
            }
          default: return false;
        }
        break;
      }
    case ("<T>"):
      {
        switch(data)
        {
          case 1:
            {
                Rule9();
                break;
            }
          case 2:
            {
                Rule9();
                break;
            }
          default: return false;
        }
        break;
      }
    case ("<T-список>"):
      {
        switch (data)
        {
          case 7:
            {
```

```
                Rule6();
                break;
            }
          case 8:
            {
                Rule6();
                break;
            }
          case 9:
            {
                Rule10();
                if (elseFlag != 1 && !jumpFlag)
                {
                    nodeList.Add(new TreeNode("*"));
                    structList.Add(new Structures());
                    structList[structList.Count - 1].r = nodeList.Count - 1;
                    structList[structList.Count - 1].p = lastCell;
                    structList[structList.Count - 1].name = "*";
                }
                k++;
                break;
            }
          default: return false;
        }
        break;
    }
  case ("<F>"):
    {
        switch(data)
        {
          case 1:
            {
                Rule11();
                if (elseFlag != 1 && !jumpFlag)
                {
                    lastCell = nodeList.FindIndex(i => i.Name == node.Name);
                }
                k++;
                break;
            }
          case 2:
            {
                if (elseFlag != 1 && !jumpFlag)
                {
                    if (nodeList.FindIndex(i => i.Name == node.Name) == -1)
                    {
                        nodeList.Add(new TreeNode(node.Name, int.Parse(node.Name)));
                        lastCell = nodeList.Count - 1;
                    }
                    else
                    {
                        lastCell = nodeList.FindIndex(i => i.Name == node.Name);
                    }
                }
                Rule11();
                k++;
                break;
            }
          default: return false;
        }
        break;
    }
  case ("<Логическое выражение>"):
```

```
  {
    switch (data)
    {
      case 1:
        {
          Rule12();
          break;
        }
      case 2:
        {
          Rule12();
          break;
        }
      default: return false;
    }
    break;
  }
case ("<Логическая операция>"):
  {
    switch (data)
    {
      case 14:

      case 17:
        {
          Rule11();
          if (elseFlag != 1 && !jumpFlag)
          {
            nodeList.Add(new TreeNode("!"));
            structList.Add(new Structures());
            structList[structList.Count - 1].p = lastCell;
            if (data == 14)
            {
              structList[structList.Count - 1].name = "=";
              nodeList[nodeList.Count - 1].Name = "=";
            }
            else
            {
              structList[structList.Count - 1].name = "!=";
              nodeList[nodeList.Count - 1].Name = "!=";
            }
            structList[structList.Count - 1].r = nodeList.Count - 1;
          }
          k++;
          break;
        }
      default: return false;
    }
    break;
  }
case (":="):
  {
    switch (data)
    {
      case 16:
        {
          Rule11();
          if (elseFlag != 1 && !jumpFlag)
          {
            structList.Add(new Structures());
            structList[structList.Count - 1].p = lastCell;
            structList[structList.Count - 1].name = ":=";
          }
```

```
                k++;
                break;
            }
          default: return false;
      }
      break;
    }
  case (";"):
    {
      switch (data)
      {
        case 7:
          {
            Rule11();
            if (elseFlag == 1)
            {
              elseFlag--;
            }
            jumpFlag = false;
            k++;
            break;
          }
        default: return false;
      }
      break;
    }
  case ("("):
    {
      switch (data)
      {
        case 10:
          {
            Rule11();
            k++;
            break;
          }
        default: return false;
      }
      break;
    }
  case (")"):
    {
      switch (data)
      {
        case 11:
          {
            Rule11();
            k++;
            break;
          }
        default: return false;
      }
      break;
    }
  case ("until"):
    {
      switch (data)
      {
        case 6:
          {
            Rule11();
            k++;
            break;
```

```
                    }
                default: return false;
            }
            break;
        }
    case ("Сложить"):
        {
            if (elseFlag != 1 && !jumpFlag)
            {
                int target = -1;
                for (int i = structList.Count - 1; i >= 0; i--)
                {
                    if (structList[i].name == "+")
                    {
                        target = i;
                        break;
                    }
                }
                structList[target].q = lastCell;
                nodeList[structList[target].r].Data = nodeList[structList[target].p].Data +
nodeList[structList[target].q].Data;
                lastCell = structList[target].r;
            }
            break;
        }
    case ("Умножить"):
        {
            if (elseFlag != 1 && !jumpFlag)
            {
                structList[structList.Count - 1].q = lastCell;
                nodeList[structList[structList.Count - 1].r].Data = nodeList[structList[structList.Count - 1].p].Data
* nodeList[structList[structList.Count - 1].q].Data;
                lastCell = structList[structList.Count - 1].r;
            }
            break;
        }
    case ("Присвоить"):
        {
            if (elseFlag != 1 && !jumpFlag)
            {
                for (int i = structList.Count - 1; i >= 0; i--)
                {
                    if (structList[i].name == ":=")
                    {
                        structList[i].q = lastCell;
                        nodeList[structList[i].p].Data = nodeList[structList[i].q].Data;
                        break;
                    }
                }
            }
            break;
        }
    case ("Сравнить"):
        {
            if (elseFlag != 1 && !jumpFlag)
            {
                structList[structList.Count - 1].q = lastCell;
                if (structList[structList.Count - 1].name == "=")
                {
                    if (nodeList[structList[structList.Count - 1].p].Data == nodeList[structList[structList.Count -
1].q].Data)
                    {
                        nodeList[structList[structList.Count - 1].r].Data = 1;
```

```
                    }
                    else
                    {
                        nodeList[structList[structList.Count - 1].r].Data = 0;
                    }
                }
                else
                {
                    if (nodeList[structList[structList.Count - 1].p].Data == nodeList[structList[structList.Count -
1].q].Data)
                    {
                        nodeList[structList[structList.Count - 1].r].Data = 0;
                    }
                    else
                    {
                        nodeList[structList[structList.Count - 1].r].Data = 1;
                    }
                }
                lastCell = structList[structList.Count - 1].r;
            }
            break;
        }
    case ("Условный переход"):
        {
            if (elseFlag != 1 && !jumpFlag)
            {
                if (nodeList[lastCell].Data == 0)
                {
                    k = mark;
                    AnalyzerStack.Pop();
                }
            }
            break;
        }
    case ("Метка"):
        {
            if (elseFlag != 1 && !jumpFlag)
            {
                mark = k - 1;
            }
            break;
        }
    case ("Условный переход по 0"):
        {
            if (elseFlag != 1 && !jumpFlag)
            {
                if (nodeList[lastCell].Data == 1)
                {
                    elseFlag = 2;
                }
                else
                {
                    jumpFlag = true;
                }
            }
            break;
        }
    }
}
string str1 = AnalyzerStack.Pop();
string str2 = AnalyzerStack.Pop();
if (str1 == "<Совокупность операторов>" && str2 == "The End")
{
```

```
        return true;
    }
    else
        return false;
}
private void Rule1()
{
    AnalyzerStack.Push(";");
    AnalyzerStack.Push("Присвоить");
    AnalyzerStack.Push("<Арифм.выражение>");
    AnalyzerStack.Push(":=");

}
private void Rule2()
{
    AnalyzerStack.Push("<Конец if>");
    AnalyzerStack.Push("<Оператор>");
    AnalyzerStack.Push("Условный переход по 0");
    AnalyzerStack.Push(")");
    AnalyzerStack.Push("<Логическое выражение>");
    AnalyzerStack.Push("(");
}
private void Rule3()
{
    AnalyzerStack.Push(";");
    AnalyzerStack.Push("Условный переход");
    AnalyzerStack.Push("<Логическое выражение>");
    AnalyzerStack.Push("until");
    AnalyzerStack.Push("<Совокупность операторов>");
    AnalyzerStack.Push("Метка");
}
private void Rule4()
{
    AnalyzerStack.Push(";");
    AnalyzerStack.Push("<Логическое выражение>");
}
private void Rule5()
{
    AnalyzerStack.Push("<Оператор>");
}
private void Rule6()
{
}
private void Rule7()
{
    AnalyzerStack.Push("<E-список>");
    AnalyzerStack.Push("<T>");
}
private void Rule8()
{
    AnalyzerStack.Push("<E-список>");
    AnalyzerStack.Push("Сложить");
    AnalyzerStack.Push("<T>");
}
private void Rule9()
{
    AnalyzerStack.Push("<T-список>");
    AnalyzerStack.Push("<F>");
}
private void Rule10()
{
    AnalyzerStack.Push("<T-список>");
    AnalyzerStack.Push("Умножить");
```

```
            AnalyzerStack.Push("<F>");
        }
        private void Rule11()
        {
        }
        private void Rule12()
        {
            AnalyzerStack.Push("Сравнить");
            AnalyzerStack.Push("<F>");
            AnalyzerStack.Push("<Логическая операция>");
            AnalyzerStack.Push("<F>");
        }
        private void Rule13()
        {
            AnalyzerStack.Push("<Совокупность операторов>");
            AnalyzerStack.Push("<Оператор>");
        }
    }
}
using System;
using System.CodeDom.Compiler;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ClassID;
using System.Runtime.Remoting.Messaging;

namespace LexicalAnalyzer
{
    internal class LexicalBlock
    {
        private string[] keywords = {"if", "else", "repeat", "until"};
        private static char[] separators = {';', '+', '*', '(', ')', ':',' ', '=', '!','\n'};
        private static string[] separatorstring = { ":=", "!=" };
        public Stack<TreeNode> tree1 = new Stack<TreeNode>();
        public string allTextProgram = null;
        private void LexicalAnalyzer(char code,ref string temp,ref string type)
        {
            if ((code >= 'a' && code <= 'z') || (code >= 'A' && code <= 'Z') || code == '_') //если символ - латинница
            {
                type += '1';
                temp += code;
                return;
                }
            if (code >= '0' && code <= '9') //если символ - цифра
            {
                type += '2';
                temp += code;
                return;
            }
            foreach (var item in separators) //если разделитель
            {
                if (code == item)
                {
                    type += '3';
                    temp += code;
                    return;
                }
            }
        }
        public bool AllTextAnalyser()
        {
```

```
string temp = null;
string type = null;
string str = null;
for (int c = 0; c < allTextProgram.Length; c++)
{
   LexicalAnalyzer(allTextProgram[c],ref temp,ref type);
}
string allTextProgram1 = allTextProgram.Replace("\r\n", "");
char[] chr = allTextProgram1.ToCharArray();
int i = 0;
while (i < type.Length - 1)
{
   if (type[i] == '2') //обработка констант (тип 2)
   {
      while (type[i] != '3')
      {
         if (type[i] == '1')
         {
            Console.WriteLine("Error");
            return false;
         }
         str += temp[i];
         i++;
      }
      tree1.Push(new TreeNode(str, 2));
      str = null;
   }
   if (type[i] == '1') //обработка переменных (тип 1)
   {
      while (type[i] != '3')
      {
         str += temp[i];
         i++;
      }
      bool flag = false;
         for (int j = 0; j < keywords.Length; j++)
         {
            if (str == keywords[j])
            {
               tree1.Push(new TreeNode(str, 3 + j)); //обработка ключевых слов (3-6)
               flag = true;
            }
         }
         if (flag == false)
         {
            tree1.Push(new TreeNode(str, 1));
         }
         str = null;
         flag = false;
   }
   if (type[i] == '3')
   {
      while ((i < type.Length) && (type[i] == '3'))
      {
         if (temp[i] != ' ' && temp[i] != '\n')
         {
            str += temp[i];
         }
         i++;
      }
      if (str != null)
      {
         if (str.Length == 1)
```

```
                {
                    for (int j = 0; j < separators.Length; j++)
                    {
                        if (str == separators[j].ToString())
                        {
                            tree1.Push(new TreeNode(str, 7 + j)); //7 - 15
                            break;
                        }
                    }
                }
                if (str.Length == 2)
                {
                    for (int j = 0; j < separatorstring.Length; j++)
                    {
                        if (str == separatorstring[j])
                        {
                            tree1.Push(new TreeNode(str, 16 + j)); //16-17
                            break;
                        }
                    }
                }
            }
            str = null;
        }
    }
    return true;
  }
 }
}
```