НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ
КАФЕДРА МАТЕМАТИЧЕСКОГО И КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ

ЛАБОРАТОРНАЯ РАБОТА №3
по дисциплине “МО ЭВМ”
“Построение синтаксического анализатора”

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

1. **Исходное задание**
    a. Преобразовать заданную грамматику в LL(1)-грамматику.
    b. Разработать МП-автомат для нисходящего грамматического разбора предложений данного языка.
    c. Разработать функцию, реализующую МП-автомат.
    d. Разработать программу, иллюстрирующую работу созданного автомата для данного языка. Программа разрабатывается как приложение с графическим интерфейсом на языке C# в среде Visual Studio. Не должны использоваться коллекции. Не должны использоваться регулярные выражения и другие средства разбора строк.

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. **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. **Разработать МП-автомат для нисходящего грамматического разбора предложений данного языка.**

	Id-1	int-2	if-3	until-6	else-4	-41	*-5	(-61	)-7	;-3	:=-9	-10	!=-101	repeat-51	\
Программа	13	Отв	13	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	13	
Оператор	1	Отв	2	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	3	Отв
Совокупность операторов	13	Отв	13	6	5	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	13	6
Конец if	6	Отв	6	6	5	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	6	Отв
E	7	7	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв
E-список	Отв	Отв	Отв	Отв	Отв	8	Отв	Отв	Отв	6	Отв	Отв	Отв	Отв	Отв
T	9	9	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв
T-список	Отв	Отв	Отв	Отв	Отв	6	10	Отв	Отв	6	Отв	Отв	Отв	Отв	Отв
F	11	11	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв
Логическое выражение	12	12	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв
Логическая операция	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	11	11	Отв	Отв

:=	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	11	Отв	Отв	Отв	Отв
;	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	11	Отв	Отв	Отв	Отв	Отв
(	Отв	Отв	Отв	Отв	Отв	Отв	Отв	11	Отв	Отв	Отв	Отв	Отв	Отв	Отв
)	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	11	Отв	Отв	Отв	Отв	Отв	Отв
until	Отв	Отв	Отв	11	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв
Id	11	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв
Int	Отв	11	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв
!=	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	11	Отв	Отв
=	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	Отв	11	Отв	Отв	Отв
∆															Доп

Таблица замен.

1.Замен(:=<Ариф. выражение>;), Сдвиг
2.Замен((<Логическое выражение>)<Оператор><Конец if>), Сдвиг
3.Замен(<Совокупность операторов> until <логическое выражение>;), Сдвиг
4.Замен(<Логическое выражение>;), Сдвиг
5.Замен(<Оператор>),Сдвиг
6.Вытолк, Держать
7.Замен (<T><E-список>), Держать
8.Замен(<T><E-список>), Сдвиг
9.Замен (<F><T-список>), Держать
10.Замен (<F><T-список>), Сдвиг
11.Вытолк, Сдвиг
12.Замен (<F><Логическая операция><F>), Держать
13.Замен(<Оператор><Совокупность операторов>),Держать

5. Алгоритм работы синтаксического анализатора на псевдокоде:

```
public bool SyntacticalAnalyzer()
{
   AnalyzerStack.Push("The End");
   AnalyzerStack.Push("Программа");
   ЦИКЛ_ПОКА (lexemStack.Count != 0)
   {
      TreeNode node = lexemStack.Pop();
      string magazine = AnalyzerStack.Pop();
      int data = node.Data;
      ВЫБОР (magazine)
```

```
{
  СЛУЧАЙ ("Программа"):
    {
      ВЫБОР (data)
      {
        СЛУЧАЙ 1:
          {
            Rule13(node);
            break;
          }
        СЛУЧАЙ 3:
          {
            Rule13(node);
            break;
          }
        СЛУЧАЙ 5:
          {
            Rule13(node);
            break;
          }
          ИНАЧЕ: return false;
      }
      break;
    }
  СЛУЧАЙ ("<Оператор>"):
    {
      ВЫБОР (data)
      {
        СЛУЧАЙ 1:
          {
            Rule1();
            break;
          }
        СЛУЧАЙ 3:
          {
            Rule2();
            break;
          }
        СЛУЧАЙ 5:
          {
            Rule3();
            break;
          }
        ИНАЧЕ: return false;
      }
      break;
    }
  СЛУЧАЙ ("<Совокупность операторов>"):
```

```
  {
    ВЫБОР (data)
    {
      СЛУЧАЙ 1:
        {
          Rule13(node);
          break;
        }
      СЛУЧАЙ 3:
        {
          Rule13(node);
          break;
        }
      СЛУЧАЙ 4:
        {
          Rule5();
          break;
        }
      СЛУЧАЙ 5:
        {
          Rule13(node);
          break;
        }
      СЛУЧАЙ 6:
        {
          Rule6(node);
          break;
        }
      ИНАЧЕ: return false;

    }
    break;
  }
СЛУЧАЙ ("<Конец if>"):
  {
    ВЫБОР (data)
    {
      СЛУЧАЙ 1:
        {
          Rule6(node);
          break;
        }
      СЛУЧАЙ 3:
        {
          Rule6(node);
          break;
        }
      СЛУЧАЙ 4:
```

```
            {
               Rule5();
               break;
            }
         СЛУЧАЙ 5:
            {
               Rule6(node);
               break;
            }
         СЛУЧАЙ 6:
            {
               Rule6(node);
               break;
            }
         ИНАЧЕ: return false;
      }
      break;
   }
СЛУЧАЙ ("<Арифм.выражение>"):
   {
      ВЫБОР (data)
      {
         СЛУЧАЙ 1:
            {
               Rule7(node);
               break;
            }
         СЛУЧАЙ 2:
            {
               Rule7(node);
               break;
            }
         ИНАЧЕ: return false;
      }
      break;
   }
СЛУЧАЙ ("<Е-список>"):
   {
      ВЫБОР(data)
      {
         СЛУЧАЙ 7:
            {
               Rule6(node);
               break;
            }
         СЛУЧАЙ 8:
            {
               Rule8();
```

```
            break;
          }
        ИНАЧЕ: return false;
      }
      break;
    }
  СЛУЧАЙ ("<T>"):
    {
      ВЫБОР(data)
      {
        СЛУЧАЙ 1:
          {
            Rule9(node);
            break;
          }
        СЛУЧАЙ 2:
          {
            Rule9(node);
            break;
          }
        ИНАЧЕ: return false;
      }
      break;
    }
  СЛУЧАЙ ("<T-список>"):
    {
      ВЫБОР (data)
      {
        СЛУЧАЙ 7:
          {
            Rule6(node);
            break;
          }
        СЛУЧАЙ 8:
          {
            Rule6(node);
            break;
          }
        СЛУЧАЙ 9:
          {
            Rule10();
            break;
          }
        ИНАЧЕ: return false;
      }
      break;
    }
  СЛУЧАЙ ("<F>"):
```

```
  {
    ВЫБОР(data)
    {
      СЛУЧАЙ 1:
        {
          Rule11();
          break;
        }
      СЛУЧАЙ 2:
        {
          Rule11();
          break;
        }
      ИНАЧЕ: return false;
    }
    break;
  }
СЛУЧАЙ ("<Логическое выражение>"):
  {
    ВЫБОР (data)
    {
      СЛУЧАЙ 1:
        {
          Rule12(node);
          break;
        }
      СЛУЧАЙ 2:
        {
          Rule12(node);
          break;
        }
      ИНАЧЕ: return false;
    }
    break;
  }
СЛУЧАЙ ("<Логическая операция>"):
  {
    ВЫБОР (data)
    {
      СЛУЧАЙ 14:
        {
          Rule11();
          break;
        }
      СЛУЧАЙ 17:
        {
          Rule11();
          break;
```

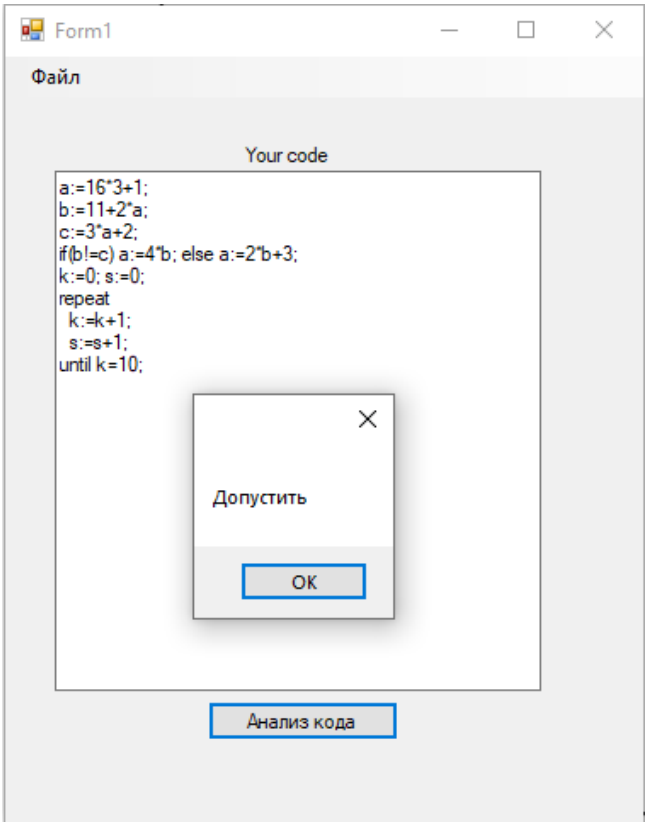
```
            }
          ИНАЧЕ: return false;
        }
        break;
    }
  СЛУЧАЙ (":="):
    {
      ВЫБОР (data)
      {
        СЛУЧАЙ 16:
          {
            Rule11();
            break;
          }
        ИНАЧЕ: return false;
      }
      break;
    }
  СЛУЧАЙ (";"):
    {
      ВЫБОР (data)
      {
        СЛУЧАЙ 7:
          {
            Rule11();
            break;
          }
        ИНАЧЕ: return false;
      }
      break;
    }
  СЛУЧАЙ ("("):
    {
      ВЫБОР (data)
      {
        СЛУЧАЙ 10:
          {
            Rule11();
            break;
          }
        ИНАЧЕ: return false;
      }
      break;
    }
  СЛУЧАЙ (")"):
    {
      ВЫБОР (data)
      {
```

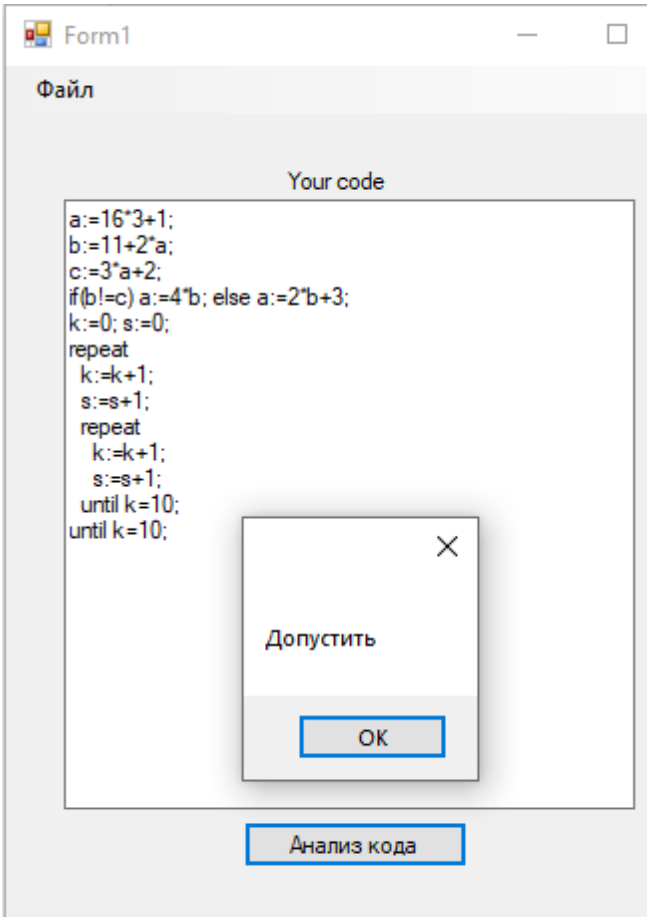
```
                СЛУЧАЙ 11:
                  {
                    Rule11();
                    break;
                  }
                ИНАЧЕ: return false;
              }
              break;
            }
          СЛУЧАЙ ("until"):
            {
              ВЫБОР (data)
              {
                СЛУЧАЙ 6:
                  {
                    Rule11();
                    break;
                  }
                ИНАЧЕ: return false;
              }
              break;
            }


        }


      } ЦИКЛ_КОНЕЦ
      string str1 = AnalyzerStack.Pop();
      string str2 = AnalyzerStack.Pop();
      ЕСЛИ (str1 == "<Совокупность операторов>" && str2 == "The End") ТОГДА
        return true;
      В_ПРОТИВНОМ_СЛУЧАЕ
        return false;
      КОНЕЦ_ЕСЛИ
    }
```

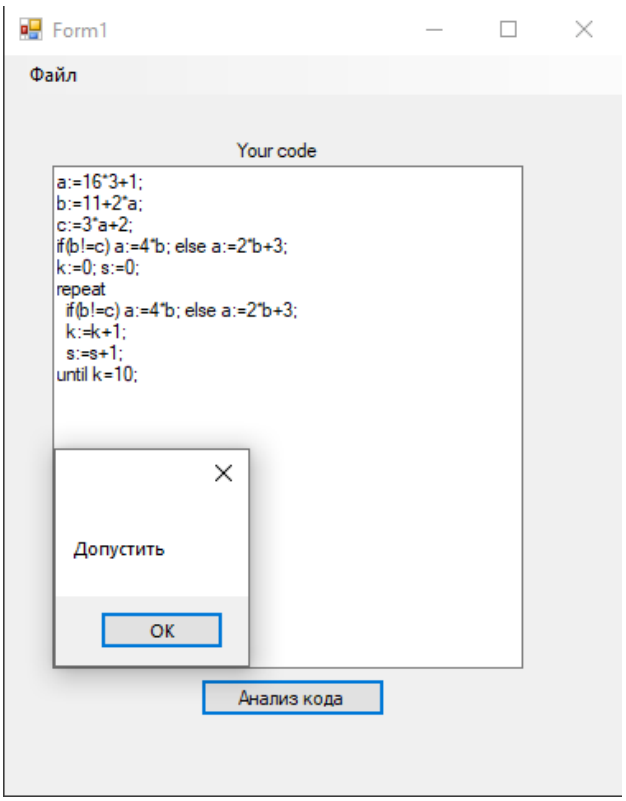
6. **Описание интерфейса программы:**

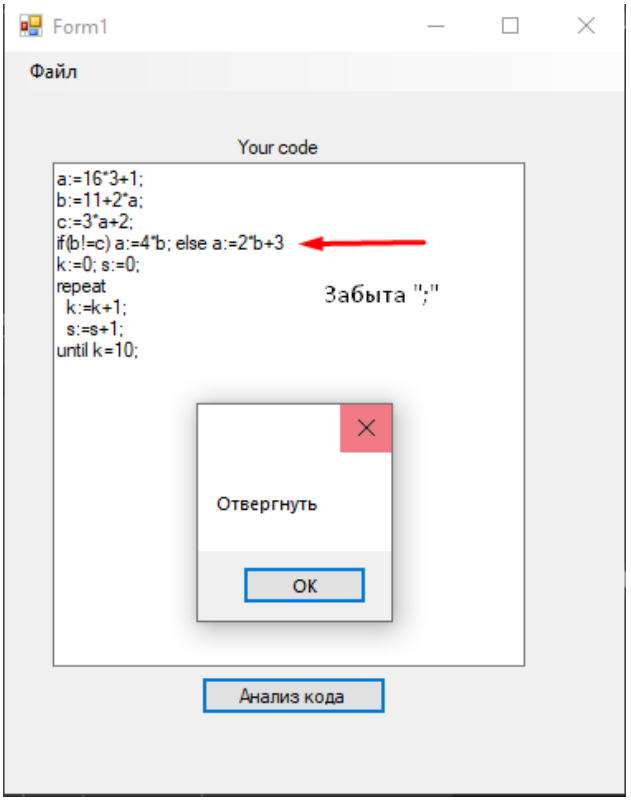
Пользователь открывает файл с кодом после чего,на экране появляется текст открытой программы. После чего пользователь нажимает кнопку “Анализ кода”, после чего пользователь получает результат работы программы - “Допустить”, если анализ прошел успешно и “Отвергнуть”, если во время синтаксического разбора программы произошла ошибка.

7. Тесты:


Form1
Файл
Your code
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;
Допустить
OK
Анализ кода


Form1
Файл
Your code
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;
repeat
k:=k+1;
s:=s+1;
until k=10;
until k=10;
Допустить
OK
Анализ кода


Form1
Файл
Your code
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
if(b!=c) a:=4*b; else a:=2*b+3;
k:=k+1;
s:=s+1;
until k=10;
Допустить
OK
Анализ кода


Form1
Файл
Your code
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;
Забыта ";"
Отвергнуть
OK
Анализ кода

8. Листинг программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using LexicalAnalyzer;
using ClassID;
namespace SyntacsycalAnalyzer
{
   internal class SyntacticalBlock
   {
      private Stack<string> AnalyzerStack = new Stack<string>(); //правая часть
      public Stack<TreeNode> lexemStack = new Stack<TreeNode>(); //левая часть
      String[] rules = {";",":=","(",")","<Арифм.выражение>","<Логическое
выражение>","<Оператор>","<Конец if>",
         "<Совокупность
операторов>","until","<T>","<E-список>","<F>","<T-список>","<Логическая операция>"};

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

      }
      public bool SyntacticalAnalyzer()
      {
         AnalyzerStack.Push("The End");
         AnalyzerStack.Push("Программа");
         while (lexemStack.Count != 0)
         {
            TreeNode node = lexemStack.Pop();
            string magazine = AnalyzerStack.Pop();
            int data = node.Data;
            switch (magazine)
            {
               case ("Программа"):
                  {
                     switch (data)
                     {
                        case 1:
                           {
                              Rule13(node);
                              break;
                           }
                        case 3:
                           {
                              Rule13(node);
                              break;
                           }
                        case 5:
                           {
                              Rule13(node);
                              break;
                           }
                           default: return false;
                     }
                     break;
                  }
               case ("<Оператор>"):
                  {
                     switch (data)
```

```
      {
         case 1:
            {
               Rule1();
               break;
            }
         case 3:
            {
               Rule2();
               break;
            }
         case 5:
            {
               Rule3();
               break;
            }
         default: return false;
      }
      break;
   }
case ("<Совокупность операторов>"):
   {
      switch (data)
      {
         case 1:
            {
               Rule13(node);
               break;
            }
         case 3:
            {
               Rule13(node);
               break;
            }
         case 4:
            {
               Rule5();
               break;
            }
         case 5:
            {
               Rule13(node);
               break;
            }
         case 6:
            {
               Rule6(node);
               break;
            }
         default: return false;

      }
      break;
   }
case ("<Конец if>"):
   {
      switch (data)
      {
         case 1:
            {
               Rule6(node);
               break;
            }
         case 3:
            {
               Rule6(node);
```

```
                break;
            }
          case 4:
            {
              Rule5();
              break;
            }
          case 5:
            {
              Rule6(node);
              break;
            }
          case 6:
            {
              Rule6(node);
              break;
            }
          default: return false;
        }
        break;
      }
    case ("<Арифм.выражение>"):
      {
        switch (data)
        {
          case 1:
            {
              Rule7(node);
              break;
            }
          case 2:
            {
              Rule7(node);
              break;
            }
          default: return false;
        }
        break;
      }
    case ("<E-список>"):
      {
        switch(data)
        {
          case 7:
            {
              Rule6(node);
              break;
            }
          case 8:
            {
              Rule8();
              break;
            }
          default: return false;
        }
        break;
      }
    case ("<T>"):
      {
        switch(data)
        {
          case 1:
            {
              Rule9(node);
              break;
            }
```

```
            case 2:
               {
                  Rule9(node);
                  break;
               }
            default: return false;
         }
         break;
      }
   case ("<Т-список>"):
      {
         switch (data)
         {
            case 7:
               {
                  Rule6(node);
                  break;
               }
            case 8:
               {
                  Rule6(node);
                  break;
               }
            case 9:
               {
                  Rule10();
                  break;
               }
            default: return false;
         }
         break;
      }
   case ("<F>"):
      {
         switch(data)
         {
            case 1:
               {
                  Rule11();
                  break;
               }
            case 2:
               {
                  Rule11();
                  break;
               }
            default: return false;
         }
         break;
      }
   case ("<Логическое выражение>"):
      {
         switch (data)
         {
            case 1:
               {
                  Rule12(node);
                  break;
               }
            case 2:
               {
                  Rule12(node);
                  break;
               }
            default: return false;
         }
```

```
      break;
    }
case ("<Логическая операция>"):
    {
      switch (data)
      {
        case 14:
          {
            Rule11();
            break;
          }
        case 17:
          {
            Rule11();
            break;
          }
        default: return false;
      }
      break;
    }
case (":="):
    {
      switch (data)
      {
        case 16:
          {
            Rule11();
            break;
          }
        default: return false;
      }
      break;
    }
case (";"):
    {
      switch (data)
      {
        case 7:
          {
            Rule11();
            break;
          }
        default: return false;
      }
      break;
    }
case ("("):
    {
      switch (data)
      {
        case 10:
          {
            Rule11();
            break;
          }
        default: return false;
      }
      break;
    }
case (")"):
    {
      switch (data)
      {
        case 11:
          {
            Rule11();
```

```
                        break;
                    }
                default: return false;
            }
            break;
        }
    case ("until"):
        {
            switch (data)
            {
                case 6:
                    {
                        Rule11();
                        break;
                    }
                default: return false;
            }
            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("<Арифм.выражение>");
    AnalyzerStack.Push(":=");

}

private void Rule2()
{
    AnalyzerStack.Push("<Конец if>");
    AnalyzerStack.Push("<Оператор>");
    AnalyzerStack.Push(")");
    AnalyzerStack.Push("<Логическое выражение>");
    AnalyzerStack.Push("(");
}

private void Rule3()
{
    AnalyzerStack.Push(";");
    AnalyzerStack.Push("<Логическое выражение>");
    AnalyzerStack.Push("until");
    AnalyzerStack.Push("<Совокупность операторов>");
}

private void Rule4()
{
    AnalyzerStack.Push(";");
    AnalyzerStack.Push("<Логическое выражение>");
}
```

```
        private void Rule5()
        {
            AnalyzerStack.Push("<Оператор>");
        }

        private void Rule6(TreeNode node)
        {
            lexemStack.Push(node);
        }

        private void Rule7(TreeNode node)
        {
            AnalyzerStack.Push("<E-список>");
            AnalyzerStack.Push("<T>");
            lexemStack.Push(node);
        }

        private void Rule8()
        {
            AnalyzerStack.Push("<E-список>");
            AnalyzerStack.Push("<T>");
        }

        private void Rule9(TreeNode node)
        {
            AnalyzerStack.Push("<T-список>");
            AnalyzerStack.Push("<F>");
            lexemStack.Push(node);

        }

        private void Rule10()
        {
            AnalyzerStack.Push("<T-список>");
            AnalyzerStack.Push("<F>");
        }
        private void Rule11()
        {
        }

        private void Rule12(TreeNode node)
        {
            AnalyzerStack.Push("<F>");
            AnalyzerStack.Push("<Логическая операция>");
            AnalyzerStack.Push("<F>");
            lexemStack.Push(node);

        }

        private void Rule13(TreeNode node)
        {
            AnalyzerStack.Push("<Совокупность операторов>");
            AnalyzerStack.Push("<Оператор>");
            lexemStack.Push(node);
        }
    }

}
```