

Лабораторная работа №3
Математическое обеспечение ЭВМ
«Построение синтаксического анализатора»

Работу выполнил:

Солонин Егор А-14-19

Вариант 14

Работу принял:

Князев А. В.

Задание :

1. Преобразовать заданную грамматику в LL(1)-грамматику.
2. Разработать МП-автомат для нисходящего грамматического разбора предложений данного языка.
3. Разработать функцию, реализующую МП-автомат.
4. 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.

Индивидуальный вариант (№14):

14	Солонин Е.В.	<p>Оператор присваивания: <ид.>=<ар.выр.>;</p> <p>Условный оператор: if(<лог.выр.>) <совок. операторов> [else <совок. операторов>] endif</p> <p>Оператор цикла: for (<ид.>=<E> ; <лог.выр.>; <ид.>=<E>) <совок. операторов> end</p> <p>Арифметическое выражение: <E>::=<T><E-список> <E-список>::= +<T><E-список> <E-список>::= ε <T>::=<F><T-список> <T-список>::= *<F><T-список> <T-список>::= ε <F>::=<Id> <F>::=<Int></p> <p>Логическое выражение: <лог.выр.>::=<F><лог.опер.><F> <лог.опер.>::= > <лог.опер.>::= <</p> <p>Пример программы:</p> <pre>a=16*3+1; b=11+2*a; c=3*a+2; if(b>c) a=4*b; b=12; else a=2*b+3; endif k=0; s=0; for(i=1; i<10; i=i+1) k=k+1; s=s+k; end</pre>
----	--------------	--

LL(1)-грамматика:

1. $\langle \text{программа} \rangle ::= \langle \text{оператор} \rangle \langle \text{спис. операторов} \rangle$
2. $\langle \text{спис. операторов} \rangle ::= \langle \text{оператор} \rangle \langle \text{спис. операторов} \rangle$
3. $\langle \text{программа} \rangle ::= \varepsilon$
4. $\langle \text{оператор} \rangle ::= \langle \text{идент.} \rangle = \langle E \rangle ;$
5. $\langle \text{оператор} \rangle ::= \text{for}(\langle \text{идент.} \rangle = \langle E \rangle ; \langle \text{лог.выр.} \rangle ; \langle \text{идент.} \rangle = \langle E \rangle) \langle \text{спис. операторов} \rangle \text{end}$
6. $\langle \text{оператор} \rangle ::= \text{if}(\langle \text{лог.выр.} \rangle) \langle \text{спис. операторов} \rangle \langle \text{иначе} \rangle \text{endif}$
7. $\langle \text{иначе} \rangle ::= \text{else} \langle \text{спис. операторов} \rangle$
8. $\langle \text{иначе} \rangle ::= \varepsilon$
9. $\langle \text{лог.выр.} \rangle ::= \langle F \rangle \langle \text{лог.опер.} \rangle \langle F \rangle$
10. $\langle \text{лог.опер.} \rangle ::= >$
11. $\langle \text{лог.опер.} \rangle ::= <$
12. $\langle E \rangle ::= \langle T \rangle \langle E\text{-список} \rangle$
13. $\langle E\text{-список} \rangle ::= + \langle T \rangle \langle E\text{-список} \rangle$
14. $\langle E\text{-список} \rangle ::= \varepsilon$
15. $\langle T \rangle ::= \langle F \rangle \langle T\text{-список} \rangle$
16. $\langle T\text{-список} \rangle ::= * \langle F \rangle \langle T\text{-список} \rangle$
17. $\langle T\text{-список} \rangle ::= \varepsilon$
18. $\langle F \rangle ::= \langle \text{идент.} \rangle$
19. $\langle F \rangle ::= \langle \text{целое} \rangle$

Множества выбора:

Выбор(1) = { $\langle \text{идент.} \rangle$, if, for}

Выбор(2) = { $\langle \text{идент.} \rangle$, if, for}

Выбор(3) = {"- |"}}

Выбор(4) = { $\langle \text{идент.} \rangle$ }

Выбор(5) = { for }

Выбор(6) = {if}

Выбор(7) = {else}

Выбор(8) = {"- |"}}

Выбор(9) = { $\langle \text{идент.} \rangle$, $\langle \text{целое} \rangle$ }

Выбор(10) = {>}

Выбор(11) = {<}

Выбор(12) = { $\langle \text{идент.} \rangle$, $\langle \text{целое} \rangle$ }

Выбор(13) = {+}

Выбор(14) = {;}

Выбор(15) = { $\langle \text{идент.} \rangle$, $\langle \text{целое} \rangle$ }

Выбор(16) = {*}

Выбор(17) = {+, ;}

Выбор(18) = { $\langle \text{идент.} \rangle$ }

Выбор(19) = { $\langle \text{целое} \rangle$ }

Управляющая таблица МП-автомата:

Магаз. символы	<идент>	<целое>	for	if	else	endif	end	+	*	()	>	<	=	;	-
<программа>	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
<спис. операторов>	2	0	2	2	3	3	3	0	0	0	0	0	0	0	0	3
<оператор>	4	0	5	6	0	0	0	0	0	0	0	0	0	0	0	0
<else>	8	0	8	8	7	8	8	0	0	0	0	0	0	0	0	8
<лог.выр.>	9	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<лог. опер.>	0	0	0	0	0	0	0	0	0	0	0	10	11	0	0	0
<E>	12	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<E список>	0	0	0	0	0	0	0	13	0	0	0	0	0	0	14	0
<T>	15	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<T список>	0	0	0	0	0	0	0	16	16	0	0	0	0	0	17	0
<F>	18	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<идент.>	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<endif>	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0
<end>	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
if	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0
for	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	Д

Начальное содержимое магазина: <программа>✓

1. Заменить(<оператор><спис. операторов>), Держать
2. Заменить(<оператор><спис. операторов>), Держать
3. Вытолкнуть, Держать
4. Заменить(=<E>;), Сдвиг
5. Заменить(for(<идент.>=<E>; <лог.выр.>; <идент.>=<E>) <спис. операторов> end), Сдвиг
6. Заменить((<лог. выр.>) <спис. операторов><else>), Сдвиг
7. Заменить(<спис. операторов>), Сдвиг
8. Вытолкнуть, Держать
9. Заменить(<F><лог.опер.><F>), Держать
10. Вытолкнуть, Сдвиг
11. Вытолкнуть, Сдвиг
12. Заменить(<T><E-список>), Держать
13. Заменить(<T><E-список>), Сдвиг
14. Вытолкнуть, Держать
15. Заменить(<F><T-список>), Держать
16. Заменить(<F><T-список>), Сдвиг
17. Вытолкнуть, Держать
18. Вытолкнуть, Сдвиг
19. Вытолкнуть, Сдвиг

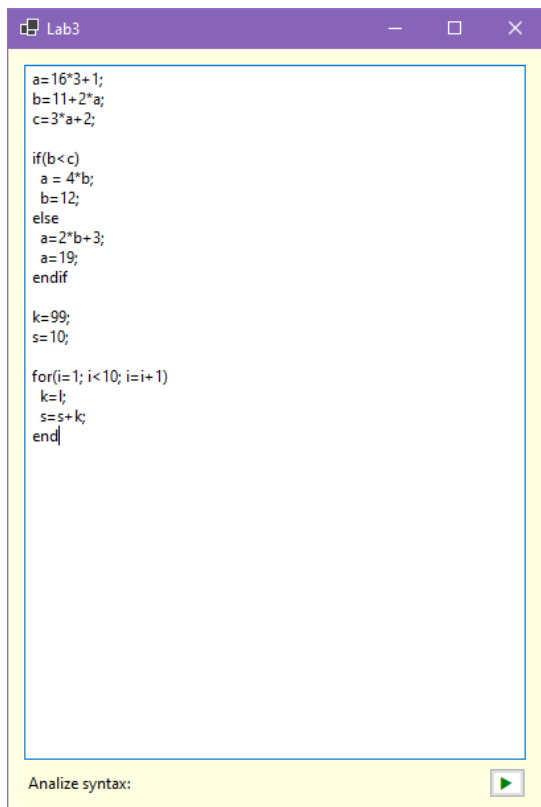
- Вытолкнуть, Сдвиг
Д - Допустить
0 - Отвергнуть

Алгоритм работы синтаксического анализатора:

1. Заводим автомат с магазинной памятью PDA на основе стека
2. Изначально добавляем <.> (признак окончания программы) и <program> в стек
3. Пока стек не пуст и пока не получен отрицательный результат работы анализатора:
 - а. Получить очередной тип лексемы с помощью функции из класса лексического блока (лабораторная работа 2)
 - б. Получить значение с вершины стека
 - с. Обработать каждый возможный случай на основе управляющей таблицы

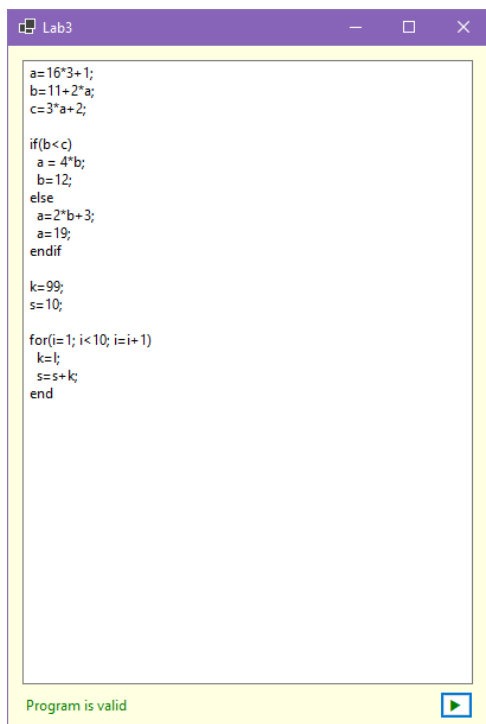
Описание интерфейса программы:

В текстовое поле заносится код. После нажатия на кнопку пользователю сообщается о корректности синтаксиса в представленном коде

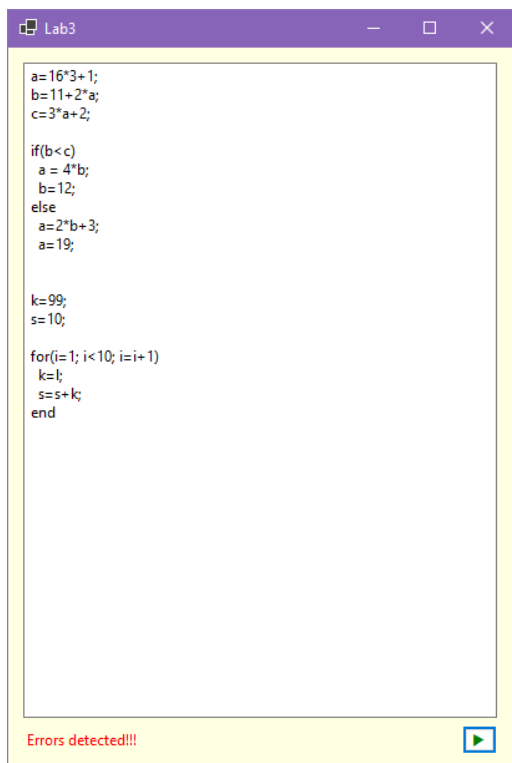


Тесты работы программы:

Тест 1 (корректный код)



Тест 2 (ошибка – отсутствует endif)



```
a=16*3+1;
b=11+2*a;
c=3*a+2;

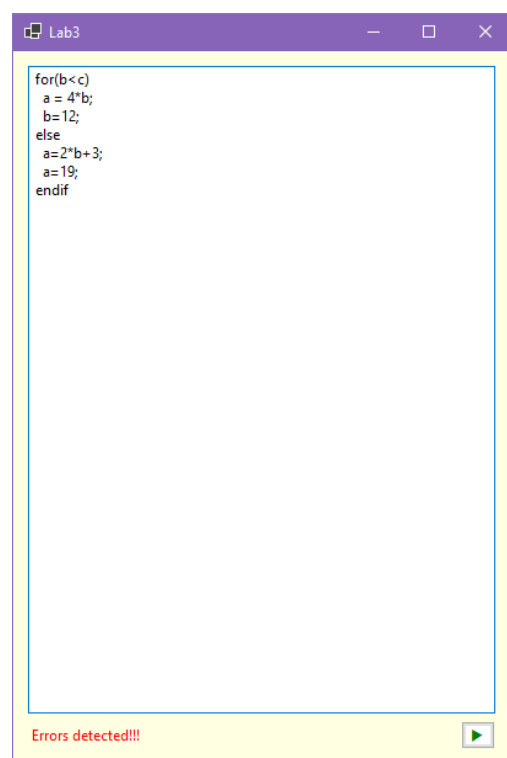
if(b<c)
    a = 4*b;
    b=12;
else
    a=2*b+3;
    a=19;

k=99;
s=10;

for(i=1; i<10; i=i+1)
    k=i;
    s=s+k;
end
```

Errors detected!!!

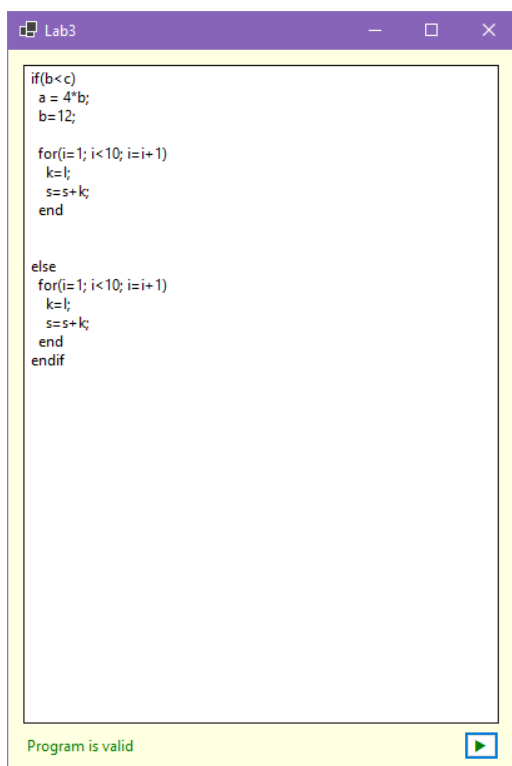
Тест 3 (ошибка – for вместо if)



```
for(b<c)
    a = 4*b;
    b=12;
else
    a=2*b+3;
    a=19;
endif
```

Errors detected!!!

Тест 5 (проверка - for внутри if/else)



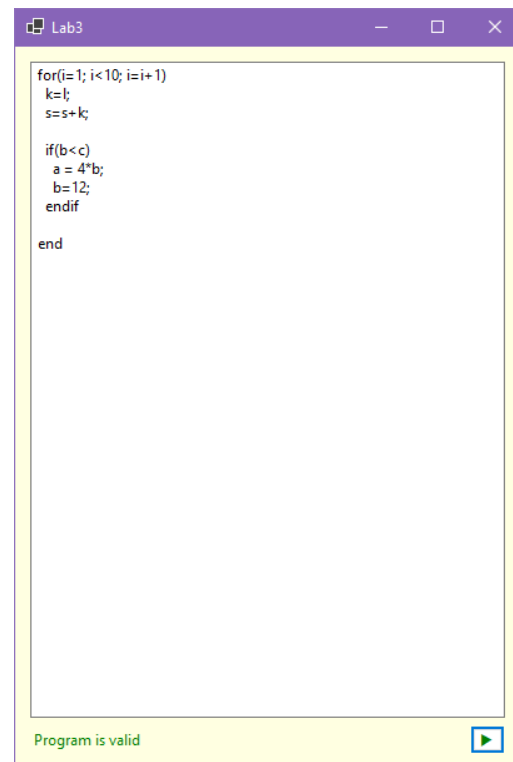
```
if(b<c)
    a = 4*b;
    b=12;

    for(i=1; i<10; i=i+1)
        k=i;
        s=s+k;
    end

else
    for(i=1; i<10; i=i+1)
        k=i;
        s=s+k;
    end
endif
```

Program is valid

Тест 6 (if внутри for)



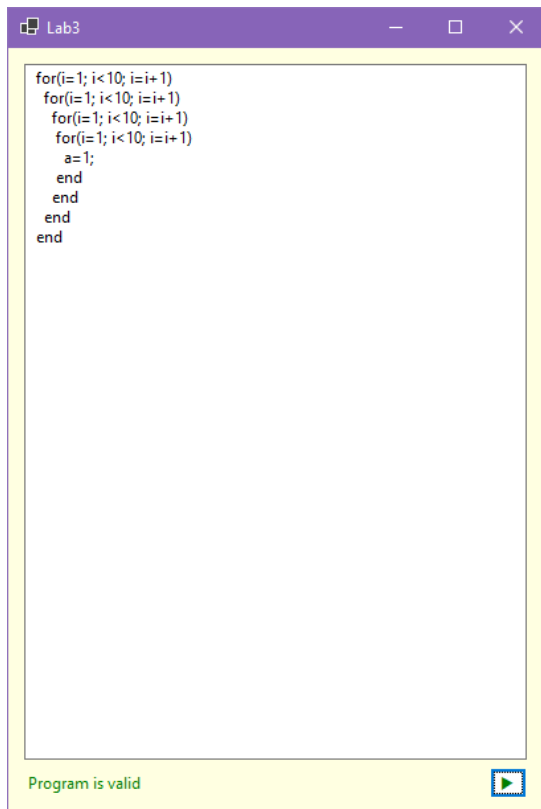
```
for(i=1; i<10; i=i+1)
    k=i;
    s=s+k;

    if(b<c)
        a = 4*b;
        b=12;
    endif

end
```

Program is valid

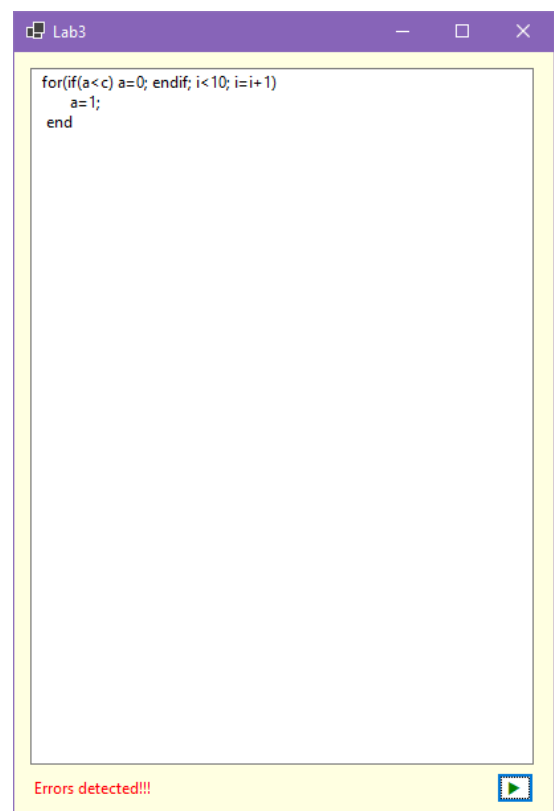
Тест 7 (вложенные циклы)



```
for(i=1; i<10; i=i+1)
  for(j=1; j<10; j=j+1)
    for(k=1; k<10; k=k+1)
      a=1;
    end
  end
end
end
```

Program is valid

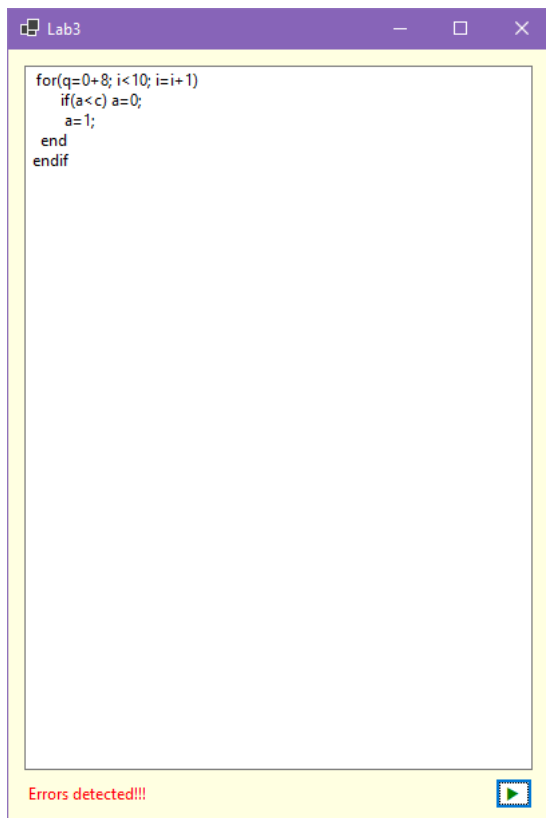
Тест 8 (блок if внутри шапки цикла for)



```
for(if(a<c) a=0; endif; i<10; i=i+1)
  a=1;
end
```

Errors detected!!!

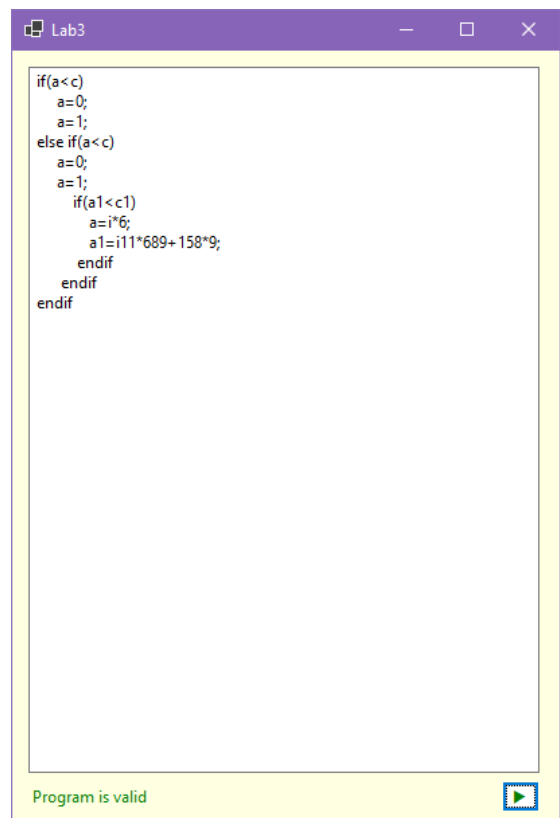
Тест 9 (ошибка – отсутствует endif)



```
for(q=0+8; i<10; i=i+1)
  if(a<c) a=0;
  a=1;
end
endif
```

Errors detected!!!

Тест 10 (вложенные if)



```
if(a<c)
  a=0;
  a=1;
else if(a<c)
  a=0;
  a=1;
  if(a1<c1)
    a=i*6;
    a1=i11*689+158*9;
  endif
endif
endif
```

Program is valid

Листинг программы:

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

    private void button1_Click(object sender, EventArgs e)
    {
        resultMsg.Text = "";
        Thread.Sleep(500);

        bool result = SyntaxAnalyzer.programIsValid(data.Text);
        resultMsg.Text = result ? "Program is valid" : "Errors detected!!!";
        resultMsg.ForeColor = result ? Color.Green : Color.Red;
    }
}

class SyntaxAnalyzer
{
    public static bool programIsValid(string data)
    {
        bool result = true;
        bool keep = false;
        bool bra_opened = false;
        bool inside_for = false;

        LexBlock.index = 0;
        string LexemType = "";
        Stack<string> PDA = new Stack<string>();

        PDA.Push("<.>");
        PDA.Push("<program>");

        while (result && !PDA.IsEmpty())
        {
            if (!keep)
            {
                string Lexem = LexBlock.GetLexem(data);
                LexemType = LexBlock.GetLexemType(Lexem).ToString();
                if (LexemType == "INVALID")
                {
                    result = false;
                    continue;
                }
            }

            switch (PDA.Peek())
            {
                case "<program>":
                    if ((LexemType == "ID") || (LexemType == "KW_IF") || (LexemType ==
"KW_FOR"))
                    {
                        PDA.Pop();
                        PDA.Push("<operators list>");
                        PDA.Push("<operator>");
                    }
                    else result = false;
                    keep = true;
                    break;

                case "<operators list>":
                    if ((LexemType == "ID") || (LexemType == "KW_IF") || (LexemType ==
"KW_FOR"))
                    {

```



```

        PDA.Pop();
        PDA.Push("<operators list>");
        PDA.Push("<operator>");
    }
    else if ((LexemType == "KW_ENDIF") ||
             (LexemType == "END") ||
             (LexemType == "KW_FOREND") ||
             (LexemType == "KW_ELSE"))
    {
        PDA.Pop();
    }
    else result = false;
    keep = true;
    break;

case ("<operator>"):
    if ((LexemType == "ID"))
    {
        PDA.Pop();
        PDA.Push("<;>");
        PDA.Push("<E>");
        PDA.Push("<=>");
    }
    else if ((LexemType == "KW_FOR"))
    {
        PDA.Pop();
        PDA.Push("<end>");
        PDA.Push("<operators list>");
        PDA.Push("<>");
        PDA.Push("<E>");
        PDA.Push("<=>");
        PDA.Push("<ID>");
        PDA.Push("<;>");
        PDA.Push("<logical expression>");
        PDA.Push("<;>");
        PDA.Push("<E>");
        PDA.Push("<=>");
        PDA.Push("<ID>");
        PDA.Push("<(>");
        inside_for = true;
    }
    else if ((LexemType == "KW_IF"))
    {
        PDA.Pop();
        PDA.Push("<endif>");
        PDA.Push("<else block>");
        PDA.Push("<operators list>");
        PDA.Push("<>");
        PDA.Push("<logical expression>");
        PDA.Push("<(>");
    }
    else result = false;
    keep = false;
    break;

case ("<else block>"):
    if (LexemType == "KW_ELSE")
    {
        PDA.Pop();
        PDA.Push("<operators list>");
        keep = false;
    }
    else if ((LexemType == "ID") ||
             (LexemType == "KW_IF") ||
             (LexemType == "KW_FOR") ||
             (LexemType == "KW_ENDIF") ||
             (LexemType == "END") ||
             (LexemType == "KW_FOREND"))
    {

```

```

        PDA.Pop();
        keep = true;
    }
    else result = false;
    break;

case ("<logical expression>"):
    if ((LexemType == "ID") || (LexemType == "INT"))
    {
        PDA.Pop();
        PDA.Push("<F>");
        PDA.Push("<logical operator>");
        PDA.Push("<F>");
        keep = true;
    }
    else result = false;
    break;

case ("<logical operator>"):
    if ((LexemType == "OP_LESS") || (LexemType == "OP_MORE"))
        PDA.Pop();
    else result = false;
    keep = false;
    break;

case ("<E>"):
    if ((LexemType == "ID") || (LexemType == "INT"))
    {
        PDA.Pop();
        PDA.Push("<E list>");
        PDA.Push("<T>");
    }
    else result = false;
    keep = true;
    break;

case ("<E list>"):
    if ((LexemType == "OP_PLS"))
    {
        PDA.Pop();
        PDA.Push("<E list>");
        PDA.Push("<T>");
        keep = false;
    }
    else if((LexemType == "DELIM_SEMI") || (LexemType == "BRA_CLS" && bra_opened
&& inside_for))
    {
        PDA.Pop();
        keep = true;
    }
    else result = false;
    break;

case ("<T>"):
    if ((LexemType == "ID") || (LexemType == "INT"))
    {
        PDA.Pop();
        PDA.Push("<T list>");
        PDA.Push("<F>");
    }
    else result = false;
    keep = true;
    break;

case ("<T list>"):
    if ((LexemType == "OP_PLS") || (LexemType == "OP_MULT"))
    {
        PDA.Pop();
        PDA.Push("<T list>");

```

```

        PDA.Push("<F>");
        keep = false;
    }
    else if ((LexemType == "DELIM_SEMI") || (LexemType == "BRA_CLS" &&
bra_opened && inside_for))
    {
        PDA.Pop();
        keep = true;
    }
    else result = false;
    break;

case ("<F>"):
    if ((LexemType == "ID") || (LexemType == "INT"))
        PDA.Pop();
    else result = false;
    keep = false;
    break;

case ("<=>"):
    Pop_Shift(ref LexemType, "OP_SET", ref PDA, ref result, ref keep);
    break;

case ("<;>"):
    Pop_Shift(ref LexemType, "DELIM_SEMI", ref PDA, ref result, ref keep);
    break;

case ("<(>)"):
    bra_opened = true;
    Pop_Shift(ref LexemType, "BRA_OPN", ref PDA, ref result, ref keep);
    break;

case ("<>>"):
    bra_opened = false;
    Pop_Shift(ref LexemType, "BRA_CLS", ref PDA, ref result, ref keep);
    break;

case ("<end>"):
    inside_for = false;
    Pop_Shift(ref LexemType, "KW_FOREND", ref PDA, ref result, ref keep);
    break;

case ("<endif>"):
    Pop_Shift(ref LexemType, "KW_ENDIF", ref PDA, ref result, ref keep);
    break;

case ("<.>"):
    Pop_Shift(ref LexemType, "END", ref PDA, ref result, ref keep);
    break;

case ("<ID>"):
    Pop_Shift(ref LexemType, "ID", ref PDA, ref result, ref keep);
    break;

default:
    result = false;
    break;
    }
}

return result;
}

private static void Pop_Shift(ref string LexemType, string GotLexType, ref Stack<string>
PDA, ref bool result, ref bool keep)
{
    if ((LexemType == GotLexType))
        PDA.Pop();
    else result = false;

```

```

        keep = false;
    }
}

class Node<T>
{
    public Node(T data)
    {
        Data = data;
    }
    public T Data { get; set; }
    public Node<T> Next { get; set; }
}

class Stack<T>
{
    Node<T> head;
    int count;
    private InvalidOperationException err411 = new InvalidOperationException("Error 411: Stack
is empty");
    public bool IsEmpty() => count == 0;
    public int Count() => count;
    public void Push(T item)
    {
        Node<T> node = new Node<T>(item);
        node.Next = head;
        head = node;
        count++;
    }
    public T Pop()
    {
        if (IsEmpty())
            throw err411;
        Node<T> temp = head;
        head = head.Next;
        count--;
        return temp.Data;
    }
    public T Peek() => IsEmpty() ? throw err411 : head.Data;
    public Stack<T> Reverse()
    {
        Stack<T> copy = new Stack<T>();
        Node<T> node = head;
        for (; node != null; node = node.Next)
            copy.Push(node.Data);
        return copy;
    }
    public T[] ToArray()
    {
        T[] arr = new T[count];
        int i = 0;
        Node<T> node = head;
        for (; node != null; node = node.Next, i++)
            arr[i] = node.Data;
        return arr;
    }
}

```