

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ  
КАФЕДРА МАТЕМАТИЧЕСКОГО И КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ

ЛАБОРАТОРНАЯ РАБОТА №4  
по дисциплине “МО ЭВМ”  
“Разработка интерпретатора”

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

Москва 2023

### 1. Исходное задание:

- a. Разработать объектный язык для заданного исходного языка.
- b. Разработать программу-интерпретатор для созданного объектного языка. Программа разрабатывается как приложение с графическим интерфейсом на языке C# в среде Visual Studio

### 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. LL(1)-грамматика:

- 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>

## 3. 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.

## 4. Описание интерфейса программы:

Пользователь открывает текстовый файл с кодом, который нужно запустить(код появляется в левой части экрана) и нажимает кнопку “Анализ кода”. После чего пользователь видит значения переменных в памяти, а также операции, примененные в вычислениях.

## 5. Тесты:

Входные данные

```
a:=16*3+1;
```

Память

Имя	Атрибут
a	49
16	16
*	48
3	3
+	49
1	1
*	

Операции

Операция
:= (p: 0, q: 4, r: 0)
* (p: 1, q: 3, r: 2)
+ (p: 2, q: 5, r: 4)
*

Анализ кода

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;
```

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)
*

Анализ кода

Входные данные

```
k:=0; s:=0;
repeat
k:=k+1;
s:=s+1;
until k=10;
```

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)

Анализ кода

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

ОК

## 7. 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 ("<Е-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("<Е-список>");
    AnalyzerStack.Push("<T>");
}

```



```

    }
    private void Rule8()
    {
        AnalyzerStack.Push("<Е-список>");
        AnalyzerStack.Push("Сложить");
        AnalyzerStack.Push("<T>");
    }
    private void Rule9()
    {
        AnalyzerStack.Push("<Т-список>");
        AnalyzerStack.Push("<F>");
    }
    private void Rule10()
    {
        AnalyzerStack.Push("<Т-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>");
    }
}
}

```

