

Лабораторная работа №2
Математическое обеспечение ЭВМ
«Построение лексического блока»

Работу выполнил:

Солонин Егор А-14-19

Вариант 14

Работу принял:

Князев А. В.

Задание:

Разработать программу, реализующую лексический блок для заданного языка.

Максимальная длина идентификатора - не менее 10 литер.

Лексический блок реализуется как подпрограмма, вызываемая синтаксическим анализатором.

Лексический блок для каждой лексемы должен указывать ее класс и значение.

Лексический блок должен строить таблицу(ы) идентификаторов и констант.

Возможны следующие виды лексем:

- идентификаторы;
- ключевые слова;
- целые числа;
- разделители.

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; endf</pre>
----	--------------	---

Описание работы программы:

В поле Code вносится фрагмент кода программы. При нажатии на кнопку Parse происходит построение двух таблиц – таблицы лексем и таблицы идентификаторов.

Грамматика для лексем:

<идент>::= буква|<идент> буква|<идент> цифра

<целое>::= цифра|<целое> цифра

<разделитель>::= +|*|=|(|)|,|>|<|;

<ключ>::= if|else|endif|for|end

<не лекс.>::= <пробел> | \n | \r

<цифра>::= 0|1|2|3|4|5|6|7|8|9

<буква>::= a|...|z|A|...|Z

Граф лексического блока:

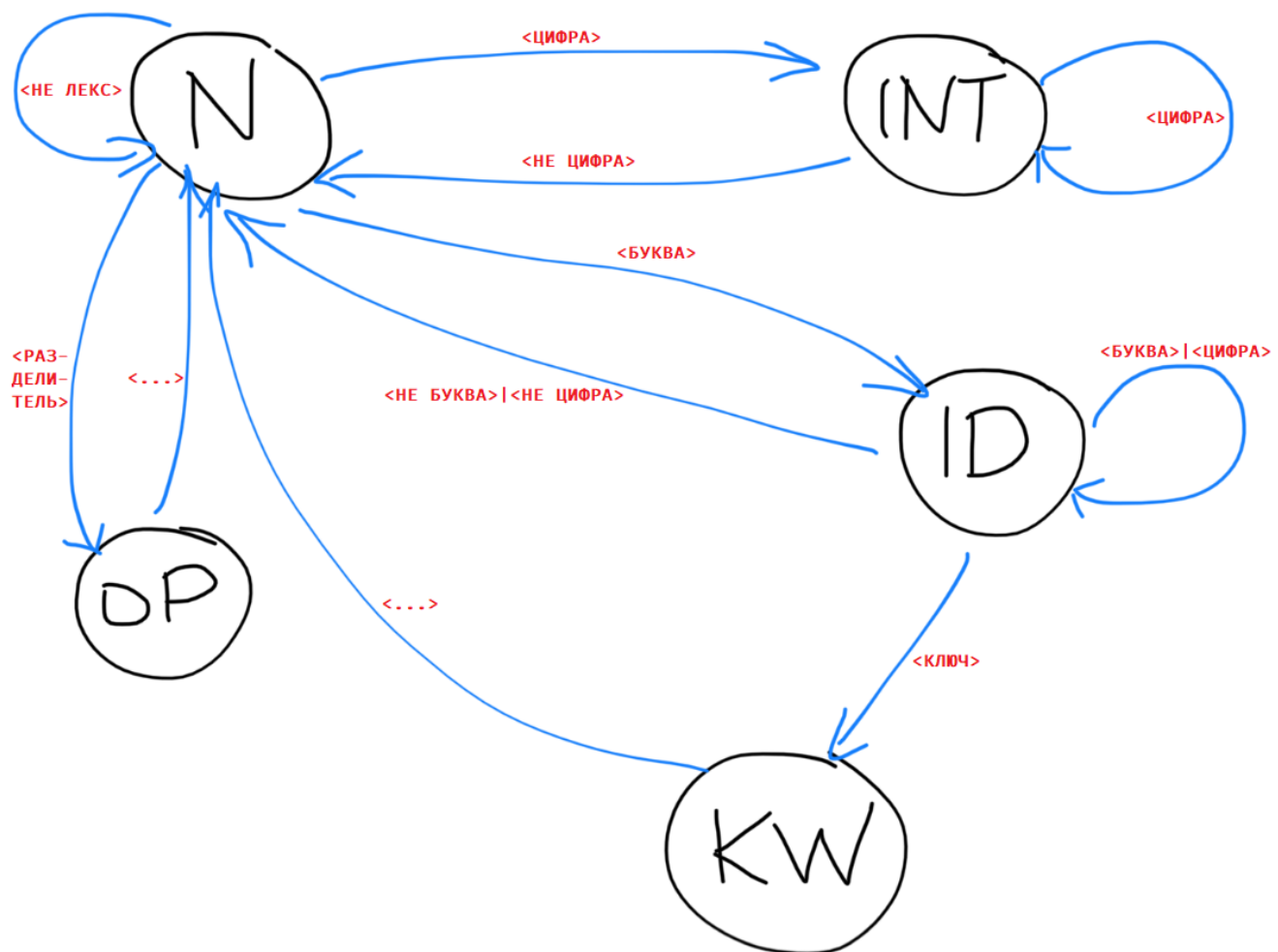
N – начальное состояние

INT – целое число

ID – идентификатор

KW – ключевое слово

OP – разделитель



Алгоритмы операций на псевдокоде:

`class LexBlock` – класс лексического блока, содержит:

- `enum LexType` – перечисление всех лексем
- `char[] Terminal` – все терминальные символы
- `LinkedListNode table` – таблица идентификаторов
- `int index` – свойство, указывает на текущий индекс в строке
- `TerminalExists(char item)` – функция, устанавливает принадлежность символа терминалу
- `isLetter(char l)` – функция, устанавливает является ли символ буквой
- `GetLexem(string data)` – функция, позволяющая получить значение лексемы
- `LexType GetLexemType(string lex)` – функция, позволяющая получить тип лексемы

`class Form1 : Form` – содержит:

- `void FillTable_Lexems()` – процедура, заполняющая таблицу лексем
- `void FillTable_Ids()` – процедура, заполняющая таблицу идентификаторов

`GetLexem(string data):`

1. Если встречаем не лексему – переходим к следующему символу
2. Если дошли до конца строки – возвращаем `\0`
3. Идем по строке начиная с `index`
 - a. Если встретили терминальный символ
 - i. Если `i == index` – возвращаем текущий символ из строки
 - ii. Иначе возвращаем подстроку – все что идет после `index` до терминального символа
 - b. Если дошли до конца строки, то возвращаем подстроку все что идет после `index` до конца строки
4. Возвращаем `\0`

`LexType GetLexemType(string lex):`

Возвращаем соответствующий тип лексемы. Отдельно обрабатываем случаи для чисел, не валидных символов и идентификаторов

`void FillTable_Lexems():`

1. Получаем очередную лексему `Lexem` пока не дошли до конца строки.
2. Получаем ее тип (атрибут)
3. Добавляем новую строку в `dataDridView` (таблицу лексем)
4. Если (тип атрибута – целое или идентификатор) и (таблица идентификаторов `LinkedListNode table` не содержит `Lexem`), то:
 - a. Создаем экземпляр идентификатора
 - b. Добавляем его в `LinkedListNode table`.

`void FillTable_Ids():`

Идем по `LinkedListNode table` и добавляем каждую запись в `dataDridView` (таблицу идентификаторов)

Тесты работы программы:

тест №1

Lab2

Code

```

a1=16*3+1;
b22=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
                    
```

Parse

Result

Lexems

Value	Class
a1	ID
=	OP_SET
16	INT
*	OP_MULT
3	INT
+	OP_PLS
1	INT
;	DELIM_SEMI
b22	ID
=	OP_SET
11	INT
+	OP_PLS
2	INT
*	OP_MULT
a	ID
;	DELIM_SEMI
c	ID
=	OP_SET
3	INT
*	OP_MULT
a	ID
+	OP_PLS
2	INT
;	DELIM_SEMI
if	KW_IF
(BRA_OPN
b	ID
>	OP_MORE

Table of identifiers

Value	Class
a1	ID
16	INT
3	INT
1	INT
b22	ID
11	INT
2	INT
a	ID
c	ID
b	ID
4	INT
12	INT
k	ID
0	INT
s	ID
i	ID
10	INT

Lab2

Code

```

a1=16*3+1;
b22=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
                    
```

Parse

Result

Lexems

Value	Class
>	OP_MORE
c	ID
)	BRA_CLS
a	ID
=	OP_SET
4	INT
*	OP_MULT
b	ID
;	DELIM_SEMI
b	ID
=	OP_SET
12	INT
;	DELIM_SEMI
else	KW_ELSE
a	ID
=	OP_SET
2	INT
*	OP_MULT
b	ID
+	OP_PLS
3	INT
;	DELIM_SEMI
endif	KW_ENDIF
k	ID
=	OP_SET
0	INT
;	DELIM_SEMI
s	ID

Table of identifiers

Value	Class
a1	ID
16	INT
3	INT
1	INT
b22	ID
11	INT
2	INT
a	ID
c	ID
b	ID
4	INT
12	INT
k	ID
0	INT
s	ID
i	ID
10	INT

Lab2

Code

```

a1=16*3+1;
b22=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

```

Parse

Result

Lexems

Value	Class
s	ID
=	OP_SET
0	INT
;	DELIM_SEMI
for	KW_FOR
(BRA_OPN
i	ID
=	OP_SET
1	INT
;	DELIM_SEMI
i	ID
<	OP_LESS
10	INT
;	DELIM_SEMI
i	ID
=	OP_SET
i	ID
+	OP_PLS
1	INT
)	BRA_CLS
k	ID
=	OP_SET
k	ID
+	OP_PLS
1	INT
;	DELIM_SEMI
s	ID
=	OP_SET
	ID

Table of identifiers

Value	Class
a1	ID
16	INT
3	INT
1	INT
b22	ID
11	INT
2	INT
a	ID
c	ID
b	ID
4	INT
12	INT
k	ID
0	INT
s	ID
i	ID
10	INT

Lab2

Code

```

a1=16*3+1;
b22=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

```

Parse

Result

Lexems

Value	Class
(BRA_OPN
i	ID
=	OP_SET
1	INT
;	DELIM_SEMI
i	ID
<	OP_LESS
10	INT
;	DELIM_SEMI
i	ID
=	OP_SET
i	ID
+	OP_PLS
1	INT
)	BRA_CLS
k	ID
=	OP_SET
k	ID
+	OP_PLS
1	INT
;	DELIM_SEMI
s	ID
=	OP_SET
s	ID
+	OP_PLS
k	ID
;	DELIM_SEMI
end	KW_FOREND

Table of identifiers

Value	Class
a1	ID
16	INT
3	INT
1	INT
b22	ID
11	INT
2	INT
a	ID
c	ID
b	ID
4	INT
12	INT
k	ID
0	INT
s	ID
i	ID
10	INT

Тест №2(проверка на валидность идентификатора)

Lab2

Code

w123443
qwer
11qqr2
11qrr
_qq
qq_
test_1
qwe1
qw%
if1
for89

Parse

Result

Lexems

	Value	Class
►	w123443	ID
	qwer	ID
	11qqr2	INVALID
	11qrr	INVALID
	_qq	INVALID
	qq_	INVALID
	test_1	INVALID
	qwe1	ID
	qw%	INVALID
	if1	ID
	for89	ID

Table of identifiers

	Value	Class
►	w123443	ID
	qwer	ID
	qwe1	ID
	if1	ID
	for89	ID

Листинг программы:

```
public class Identifier {
    public string Name { get; set; }
    public string Attr { get; set; }
}
public class LinkedListNode {
    private LinkedListNode _head;
    private LinkedListNode _tail;
    public LinkedListNode() {}
    public LinkedListNode(Identifier value) {
        Value = value;
    }

    public Identifier Value { get; internal set; }
    public LinkedListNode Next { get; internal set; }

    public int Count {
        get;
        private set;
    }

    public LinkedListNode Head { get { return _head; } }

    public void Add(Identifier item) {
        LinkedListNode node = new LinkedListNode(item);

        if(_head == null) {
            _head = node;
            _tail = node;
        } else {
            _tail.Next = node;
            _tail = node;
        }
        Count++;
    }

    public bool Contains(string name) {
        LinkedListNode node = _head;
        while(node != null) {
            if(node.Value.Name == name)
                return true;
            node = node.Next;
        }
        return false;
    }

    public bool Remove(string name) {
        LinkedListNode prev = null;
        LinkedListNode curr = _head;

        while(curr != null) {
            if(curr.Value.Name == name) {
                if(prev != null) {
                    prev.Next = curr.Next;
                    if (curr.Next == null)
                        _tail = prev;
                } else {
                    _head = _head.Next;
                    if(_head == null)
                        _tail = null;
                }
                Count--;
                return true;
            }
            prev = curr;
            curr = curr.Next;
        }
    }
}
```



```

        curr = curr.Next;
    }
    return false;
}

public void Print()
{
    LinkedListNode node = _head;
    while(node != null) {
        Console.WriteLine($"{node.Value.Name}, {node.Value.Attr}");
        node = node.Next;
    }
}

}

static class LexBlock
{
    public enum LexType
    {
        INVALID,
        ID,
        INT,
        OP_SET,
        OP_MULT,
        OP_PLS,
        OP_LESS,
        OP_MORE,
        DELIM_COM,
        DELIM_SEMI,
        BRA_OPN,
        BRA_CLS,
        KW_IF,
        KW_ELSE,
        KW_ENDIF,
        KW_FOR,
        KW_FOREND,
        END,
    }

    public static int index { get; set; }
    public static char[] Terminal = { '+', '*', '(', ')', ',', ';', '=', '<', '>', ' ', '\n', '\r' };
    public static LinkedListNode table = new LinkedListNode();

    private static bool TerminalExists(char item)
    {
        foreach (char lex in Terminal)
            if (lex == item) return true;
        return false;
    }

    private static bool isLetter(char l) => (l >= 'a' && l <= 'z') || (l >= 'A' && l <= 'Z');

    public static string GetLexem(string data)
    {
        for(; (index < data.Length)
            && (data[index] == ' '
            || data[index] == '\0'
            || data[index] == '\n'
            || data[index] == '\r'); index++);
        if (index == data.Length)
            return "\0";

        for (int i = index; i < data.Length; i++)
        {
            if (TerminalExists(data[i]))
            {
                if (index == i)
                {
                    index = i + 1;

```

```

        return data[i].ToString();
    }
    string temp = data.Substring(index, i - index);
    index = i;
    return temp;
}
if (i == data.Length - 1)
{
    string temp = data.Substring(index, i - index + 1);
    index = i + 1;
    return temp;
}
}
return "\\0";
}
public static LexType GetLexemType(string lex)
{
    switch (lex)
    {
        case "if": return LexType.KW_IF;
        case "else": return LexType.KW_ELSE;
        case "endif": return LexType.KW_ENDIF;
        case "for": return LexType.KW_FOR;
        case "end": return LexType.KW_FOREND;
        case "(": return LexType.BRA_OPN;
        case ")": return LexType.BRA_CLS;
        case "<": return LexType.OP_LESS;
        case ">": return LexType.OP_MORE;
        case "=": return LexType.OP_SET;
        case "+": return LexType.OP_PLS;
        case "*": return LexType.OP_MULT;
        case ";": return LexType.DELIM_SEMI;
        case ",": return LexType.DELIM_COM;
        case "\\0": return LexType.END;
        default: break;
    }

    int num = 0;
    if (int.TryParse(lex, out num))
        return LexType.INT;

    if (isLetter(lex[0]))
        foreach (char l in lex)
        {
            if (isLetter(l) || int.TryParse(l.ToString(), out num))
                continue;
            return LexType.INVALID;
        }
    else return LexType.INVALID;

    return LexType.ID;
}

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

    private void button1_Click(object sender, EventArgs e)
    {
        LexBlock.table = new LinkedListNode();
        LexBlock.index = 0;

        FillTable_Lexems();
        FillTable_Ids();
    }
}

```

```

    }

private void FillTable_Lexems()
{
    string Lexem = "";
    LexemsTable.Rows.Clear();

    while ((Lexem = LexBlock.GetLexem(inputData.Text)) != "\0")
    {
        string Attr = LexBlock.GetLexemType(Lexem).ToString();
        LexemsTable.Rows.Add(Lexem, Attr);

        if(!LexBlock.table.Contains(Lexem) && (Attr == "ID" || Attr == "INT"))
        {
            Identifier id = new Identifier
            {
                Name = Lexem,
                Attr = Attr
            };
            LexBlock.table.Add(id);
        }
    }
}

private void FillTable_Ids()
{
    TableOfIds.Rows.Clear();
    LinkedListNode node = LexBlock.table.Head;
    while (node != null)
    {
        TableOfIds.Rows.Add(node.Value.Name, node.Value.Attr);
        node = node.Next;
    }
}
}

```