Московский авиационный институт
(Национальный исследовательский университет)

Факультет «Информационные технологии и прикладная математика»

Курсовая работа
по курсу
«Системы программирования»
по теме
«Разработка и реализация автомата с магазинной памятью»
IV семестр

Студент:	Шевчук П.В.
Группа:	М8О-204Б
Преподаватель:	Семенов А.С.
Оценка:	
Дата:	
Подпись:	

Москва
2018

Задание

Построение МП-автомата, допускающего язык $L = \{a^n b^n \mid n \geq 0\}$.
Входная цепочка: aaabbb

МП = ($\{q_0, q_1, q_2, q_f\}$, {a,b}, {z_0,a}, δ, q_0, z_0, { q_f })
1 $\delta(q_0,a, z_0) = \{(q_1,az_0)\}$
2 $\delta(q_1,a, a) = \{(q_1,aa)\}$
3 $\delta(q_1,b, a) = \{(q_2, \varepsilon)\}$
4 $\delta(q_2,b, a) = \{(q_2, \varepsilon)\}$
5 $\delta(q_2, \varepsilon, z_0) = \{(q_f, \varepsilon)\}$

Последовательность тактов:

$(q_0, aaabbb, z_0) \vdash_1 (q_1, aabbb, az_0) \vdash_2 (q_1, abbb, aaz_0) \vdash_2$
$(q_1, bbb, aaaz_0) \vdash_3 (q_2, bb, aaz_0) \vdash_3 (q_1, b, az_0) \vdash_3$
$(q_2, \varepsilon, z_0) \vdash_4 (q_f, \varepsilon, \varepsilon)$

Теория

МП автомат – это семерка объектов

МП = $(Q, \sum, \Gamma, \delta, q_0, z_0, F)$

Q – конечное множество состояний устройства управления;

$\sum$ - конечный алфавит входных символов;

Γ - конечный алфавит магазинных символов;

δ - функция переходов, отображает множества $Q\times(\sum \cup \{\varepsilon\})\times\Gamma$ в множество конечных подмножеств множества $Q\times\Gamma^*$;

q_0 - начальное состояние, $q_0 \in Q$;

z_0 - начальный символ магазина, $z_0 \in \Gamma$;

F - множество заключительных состояний, $F \subseteq Q$.

Конфигурацией МП-автомата называется тройка $(q, \omega, z) \in Q\times\sum^*\times\Gamma^*$, где

q – текущее состояние управляющего устройства;

ω – необработанная часть входной цепочки (первый символ цепочки ω находится под входной головкой; если $\omega = \varepsilon$, то считается, что вся входная цепочка прочитана);

z – содержимое магазина (самый левый символ цепочки z считается верхним символом магазина; если $z = \varepsilon$, то магазин считается пустым).

Такт МП-автомата будем описывать бинарным отношением $\vdash$, определенным на множестве конфигураций. Будем писать:

$(q, a\omega, z) \vdash (q', \omega, z\gamma)$, если $\delta(q, a, z) = (q', \gamma)$, где

$q, q' \in Q$, $a \in \sum \cup \{\varepsilon\}$, $\omega \in \sum^*$, $z \in \Gamma$ и $\gamma \in \Gamma^*$

Если $a \neq \varepsilon$, то и входная цепочка прочитана не вся, то запись $(q, a\omega, z\gamma) \vdash (q', \omega, a\gamma)$ означает, что МП-автомат в состоянии q, обозревая символ в входной цепочки и имея символ z в верхушке магазина, может перейти в новое состояние q', сдвинуть входную головку на один символ вправо и заменить верхний символ магазина z цепочкой магазинных символов γ. Если z = a, то верхний символ удаляется из магазина.

Если $a = \varepsilon$, то текущий входной символ в этом такте называется ε-таком, не принимается во внимание и входная головка остается неподвижной.

ε-такты могут выполняться также в случае, когда вся входная цепочка прочитана, но если магазин пуст, то такт МП-автомата невозможен по определению.

Код программы

Приведены основные фрагменты кода.

Program.cs

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Data;
using System.Text;

namespace myCompiler
{
    class Program
    {

            switch (Console.ReadLine())
               {
                  ...

                       case "7": //МП - автоматы

                       myMp Mp = new myMp(new ArrayList() { "q0", "q1", "q2", "qf"
}, new ArrayList() { "a", "b" }, new ArrayList() { "z0", "a" }, "q0", new ArrayList()
{ "qf" });

                       Mp.addDeltaRule("q0", "a", "z0", new ArrayList() { "q1" },
new ArrayList() { "a", "z0" });
                       Mp.addDeltaRule("q1", "a", "a", new ArrayList() { "q1" }, new
ArrayList() { "a", "a" });
                       Mp.addDeltaRule("q1", "b", "a", new ArrayList() { "q2" }, new
ArrayList() { "e" });
                       Mp.addDeltaRule("q2", "b", "a", new ArrayList() { "q2" }, new
ArrayList() { "e" });
                       Mp.addDeltaRule("q2", "e", "z0", new ArrayList() { "qf" },
new ArrayList() { "e" });
                       Console.Write("Debug Mp ");
                       Mp.debugDelta();

                       Console.WriteLine("\nEnter the line :");

Console.WriteLine(Mp.Execute_(Console.ReadLine()).ToString());
                       break;


               }
    }
}
```

MyMP.cs

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Text;

namespace myCompiler
```

```
{

    class DeltaQSigmaGamma
    {
        // структура Delta отображения
        private string LeftQ = null;  // исходное состояние
        private string LeftT = null;  // символ входной цепочки
        private string LeftZ = null;  // верхний символ магазин
        private ArrayList RightQ = null;  // множество следующих состояний
        private ArrayList RightZ = null;  // множество символов магазина

        public string leftQ { get { return LeftQ; } set { LeftQ = value; } }
        public string leftT { get { return LeftT; } set { LeftT = value; } }
        public string leftZ { get { return LeftZ; } set { LeftZ = value; } }
        public ArrayList rightQ { get { return RightQ; } set { RightQ = value; } }
        public ArrayList rightZ { get { return RightZ; } set { RightZ = value; } }

        // Delta (  q1   ,   a    ,   z   ) = {  {q}   ,   {z1z2...} }
        //         LeftQ    LeftT   LeftZ       RightQ       RightZ
        public DeltaQSigmaGamma(string LeftQ, string LeftT, string LeftZ, ArrayList
RightQ, ArrayList RightZ)
        {
            this.LeftQ = LeftQ;
            this.LeftT = LeftT;
            this.LeftZ = LeftZ;
            this.RightQ = RightQ;
            this.RightZ = RightZ;
        }
    } // end class Delta

    class myMp : Automate    //МП = {}
    {
        // Q - множество состояний МП - автомата
        // Sigma - алфавит входных символов
        // DeltaList - правила перехода
        // Q0 - начальное состояние
        // F - множество конечных состояний
        public ArrayList Gamma = null;     //алфавит магазинных символов
        public string z0 = null;           //начальный символ магазина

        public Stack Z = null;
        private int c = 1;

        //
        public myMp(ArrayList Q, ArrayList Sigma, ArrayList Gamma, string Q0,
ArrayList F)
            : base(Q, Sigma, F, Q0)
        {
            this.Gamma = Gamma;

            this.Z = new Stack();
            Q0 = Q[0].ToString();  // начальное состояние
            Z.Push(Q0);  // начальный символ в магазине
            this.F = F;       // пустое множество заключительных состояний
        }
        //

        public myMp(myGrammar KCgrammar)
            : base(new ArrayList() { "q" }, KCgrammar.T, new ArrayList() { }, "q")
        {
            this.Gamma = new ArrayList();
            this.Z = new Stack();
```

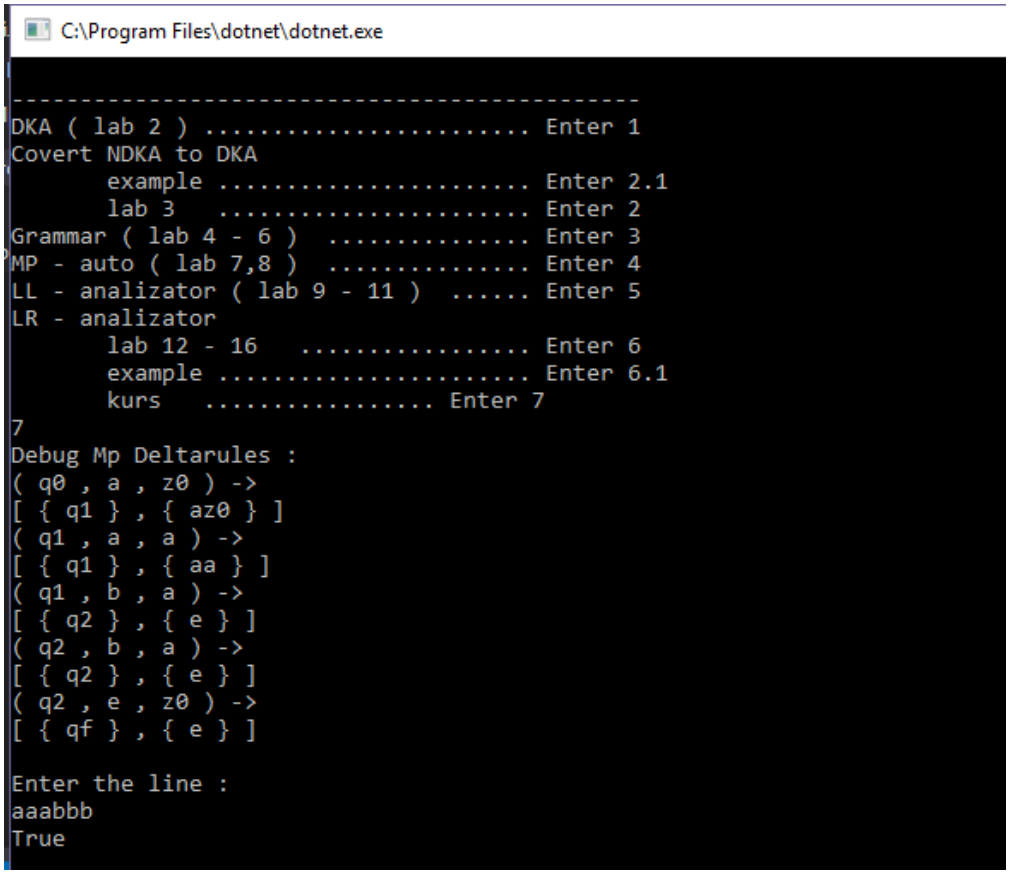
```
            foreach (string v1 in KCgrammar.V)   // магазинные символы
                Gamma.Add(v1);
            foreach (string t1 in KCgrammar.T)
                Gamma.Add(t1);
            Q0 = Q[0].ToString();  // начальное состояние
            Z.Push(KCgrammar.S0);  // начальный символ в магазине
            F = new ArrayList();  // пустое множество заключительных состояний

            DeltaQSigmaGamma delta = null;
            foreach (string v1 in KCgrammar.V)
            {            // сопоставление правил с отображениями
                ArrayList q1 = new ArrayList();
                ArrayList z1 = new ArrayList();
                foreach (Prule rule in KCgrammar.Prules)
                {
                    if (rule.leftNoTerm == v1)
                    {
                        Stack zb = new Stack();
                        ArrayList rr = new ArrayList(rule.rightChain);
                        rr.Reverse();
                        foreach (string s in rr)
                            zb.Push(s);
                        z1.Add(zb);
                        q1.Add(Q0);
                    }
                }
                delta = new DeltaQSigmaGamma(Q0, "e", v1, q1, z1);
                DeltaList.Add(delta);
            }
            foreach (string t1 in KCgrammar.T)
            {
                Stack e = new Stack();
                e.Push("e");
                delta = new DeltaQSigmaGamma(Q0, t1, t1, new ArrayList() { Q0 }, new
ArrayList() { e });
                DeltaList.Add(delta);
            }
        }

        public void addDeltaRule(string LeftQ, string LeftT, string LeftZ, ArrayList
RightQ, ArrayList RightZ)
        {
            DeltaList.Add(new DeltaQSigmaGamma(LeftQ, LeftT, LeftZ, RightQ, RightZ));
        }

        public bool Execute_(string str)
        {

            string currState = this.Q0;
            DeltaQSigmaGamma delta = null;
            int i = 0;
            str = str + "e";
            for (; ; )  // empty step
            {
                delta = findDelta(currState, str[i].ToString());
                if (delta == null) return false;
                if (delta.leftT != "e")
                {
                    for (; i < str.Length;)
                    {
                        this.Q = delta.rightQ;
                        currState = arrToStr(delta.rightQ);
```

```
                        if (delta.leftZ == Z.Peek().ToString() &&
delta.rightZ[0].ToString() == "e")
                        {
                            this.Z.Pop();
                        }
                        else this.Z.Push(delta.leftT);
                        i++;
                        break;
                    }
                }
                else if (delta.leftT == "e")
                {
                    this.Q = delta.rightQ;
                    this.Z.Pop();
                    if (this.Z.Count == 0) return true;
                    else return false;
                }
            } // end for
        } // end Execute_

        //
        // поиск правила по состоянию.
        private DeltaQSigmaGamma findDelta(string Q, string a)
        {
            foreach (DeltaQSigmaGamma delta in this.DeltaList)
            {
                if (delta.leftQ == Q && delta.leftT == a) return delta;
            }
            return null; // not find
        }


        //*** вспомогательные процедуры ***

        //объединение множеств A or B
        public ArrayList Unify(ArrayList A, ArrayList B)
        {
            ArrayList unify = A;
            foreach (string s in B)
                if (!A.Contains(s))
                    unify.Add(s);
            return unify;
        }

        //преобразование элементов массива в строку
        public string arrToStr(ArrayList array)
        {
            if (array.Equals(null)) return null;
            else
            {
                string newLine = "";
                foreach (string s in array)
                    newLine += s;
                return newLine;
            }
        }

        public string StackToString(Stack Z)
        {
            if (Z.Count == 0) return null;
            else
            {
```

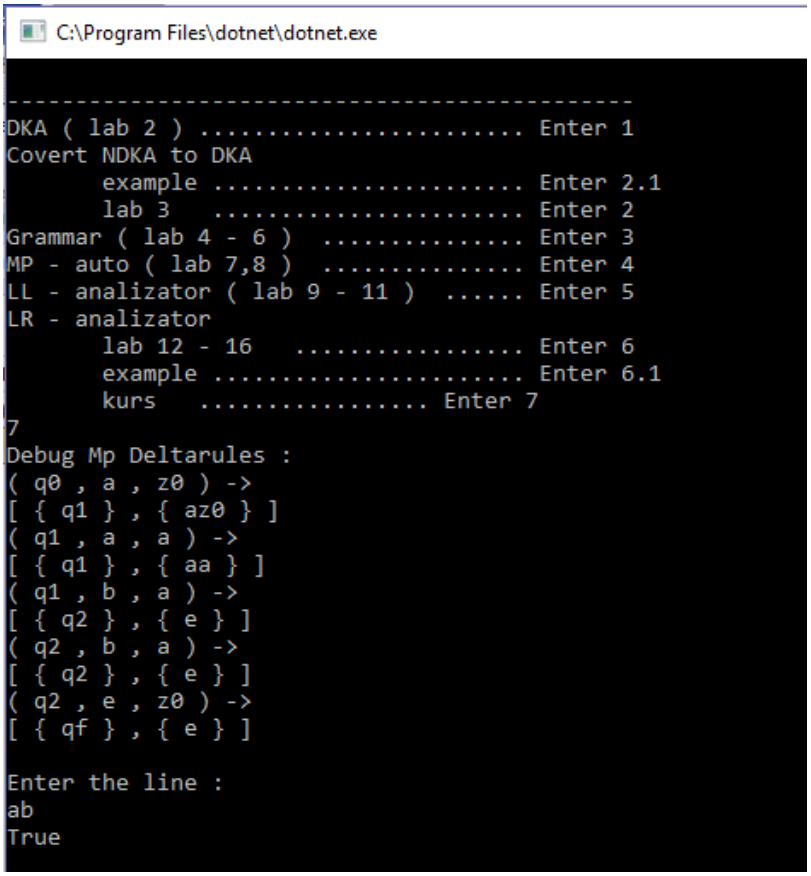
```
                string newLine = "";
                Stack temp = new Stack();
                for (int i = 0; i < Z.Count; i++)
                {
                    temp.Push(Z.Pop());
                    newLine += Z.Peek();
                }
                for (int i = 0; i < temp.Count; i++)
                    Z.Push(temp.Pop());
                return newLine;
            }
        }



        // **   Debug   **
        public string DebugStack(Stack s)
        { // печать текущего состояния магазина
            string p = "|";
            Stack s1 = new Stack();
            while (s.Count != 0)
            {
                s1.Push(s.Pop());
                p = p + s1.Peek().ToString();
            }
            while (s1.Count != 0) s.Push(s1.Pop());
            return p;
        }

        public void debugDelta()
        {
            Console.WriteLine("Deltarules :");
            if (this.DeltaList == null) Console.WriteLine("null");
            else
                foreach (DeltaQSigmaGamma d in this.DeltaList)
                { // тут
                    Console.Write("( " + d.leftQ + " , " + d.leftT + " , " + d.leftZ
+ " )");
                    Console.Write(" -> \n");
                    Console.WriteLine("[ { " + arrToStr(d.rightQ) + " } , { " +
arrToStr(d.rightZ) + " } ]");
                }
        }
    }
}
```

Пример работы

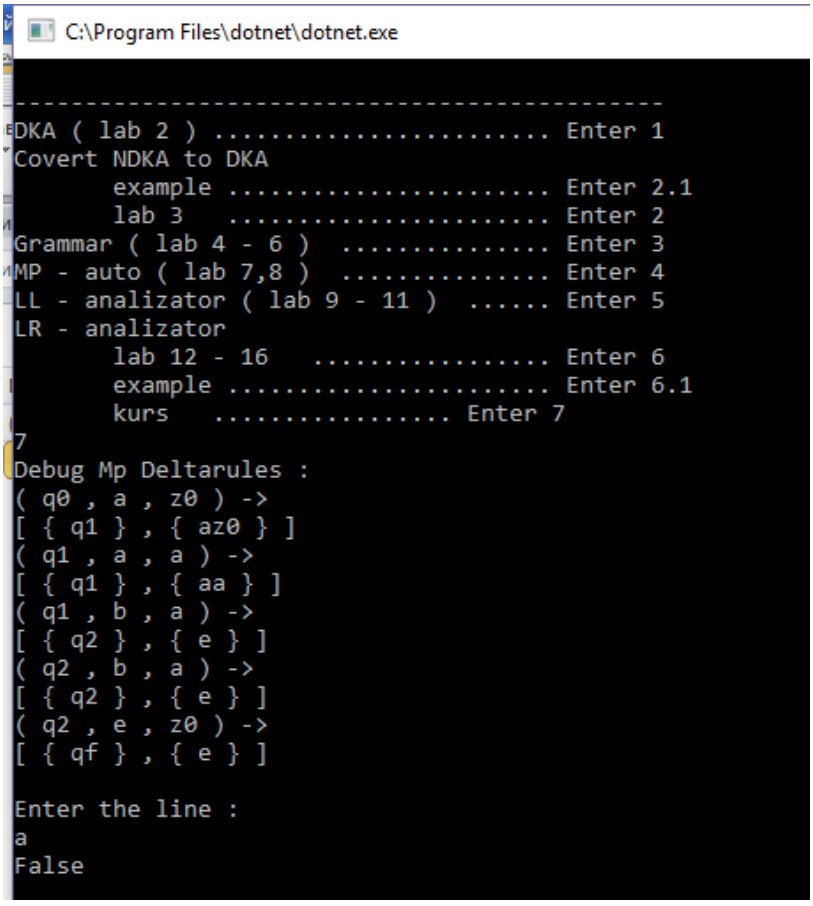
Входная строка: aaabbb – принадлежит языку L.



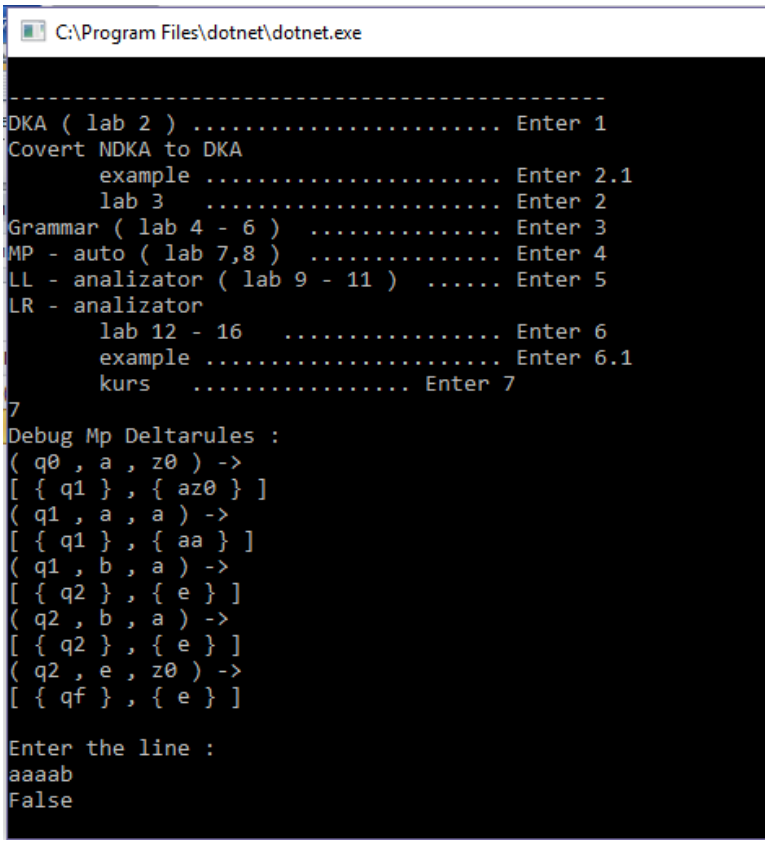
Входная строка: ab – принадлежит языку L.



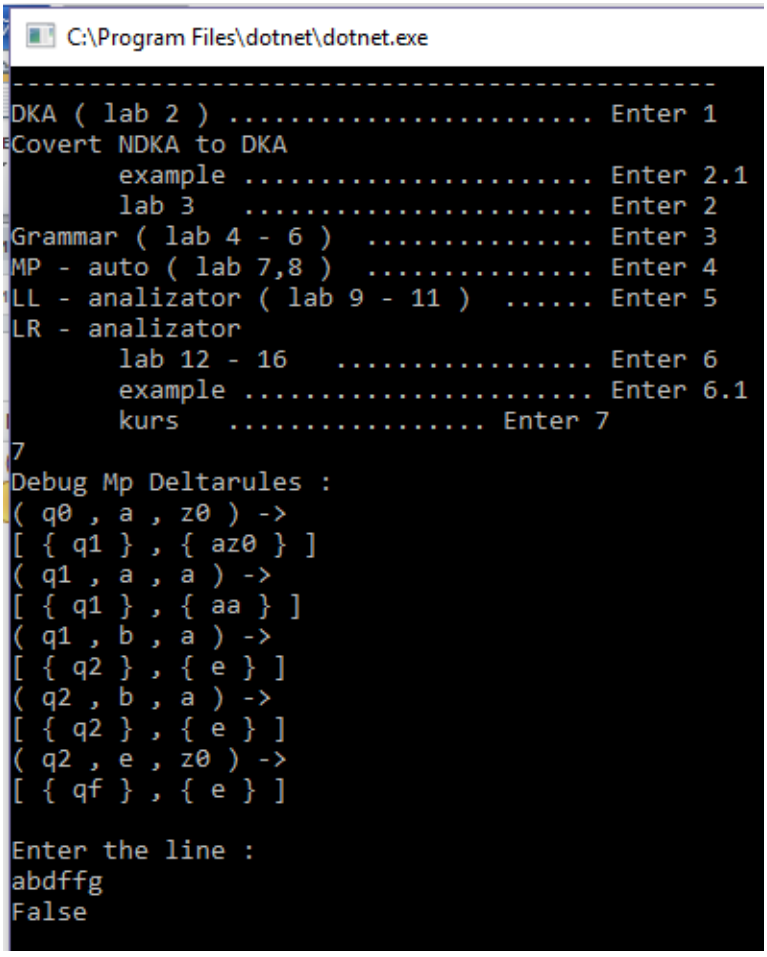
Входная строка: a – не принадлежит языку L.



Входная строка: aaaab – не принадлежит языку L.



Входная строка: abdffg – принадлежит языку L.



Заключение

Во время выполнения курсовой работы были отточены навыки работы с мп-автоматами. Это несомненно было полезно, так как поможет в дальнейшем обучении. Так же были приобретены знания в программировании на c# и повторены основные принципы ООП.
В процессе реализации построения мп-автомата был написан код, осуществляющий проверку принадлежности строки заданному языку.