

Cache.cs:

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

namespace Cachework
{
    class Cache
    {
        int[,] strings; // массив из строк кэша
        int[] tags; // тэги

        public Cache(int n, int m)
        {
            strings = new int[n, m];
            tags = new int[n];

            for (int i = 0; i < n; i++)
            {
                tags[i] = -1;
            }

            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < m; j++)
                {
                    strings[i, j] = 0;
                }
            }
        }

        // Доступ к элементам массива
        кэша public int this [int i,
        int j]
        {
            get
            {
                return strings[i, j];
            }

            set
            {
                strings[i, j] = value;
            }
        }

        //Доступ к тэгам
        public int this [int i]
        {
            get
            {
                return tags[i];
            }

            set
            {
                tags[i] = value;
            }
        }
    }
}
```

```

/ метод, проверяющий, есть ли строка j страницы i в
кэше public bool isThereATag(int i, int j)
{
    return tags[j] == i ? true : false;
}

/ Записать новую строку размерностью n в кэш память
public void writeLine(int [] str, int n, int indexLine)
{
    for (int i = 0; i < n; i++)
    {
        strings[indexLine, i] = str[i];
    }
}
}
}

```

MainMemory.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace Cachework
{
    class MainMemory
    {
        string filename;
        public int CountPages,
            CountLines,
            CountElements;

        BinaryWriter Write;
        BinaryReader Read;

        public MainMemory(string filename, int i, int j, int k)
        {
            this.filename = filename;
            CountPages = i;
            CountLines = j;
            CountElements = k;
        }

        public void RandomArray(int [, ,] arr, int page, int n, int m)
        {
            Random rnd = new Random();

            for (int i = 0; i < page; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    for (int k = 0; k < m; k++)
                    {
                        arr[i, j, k] = (rnd.Next(1000, 9999));
                    }
                }
            }
        }
    }
}

```

```

/ Записать заданную строку в массив
public void SetLineOnArray(ref int [,,] arr, int[] line, int
indexPage, int indexLine, int count)
{
    for (int i = 0; i < count; i++)
    {
        arr[indexPage, indexLine, i] = line[i];
    }
}

public void writeArray(int[,,] arr, int page, int n, int m)
{
    using (Write = new BinaryWriter(new FileStream(filename,
        FileMode.Create)))
    {
        for (int i = 0; i < page; i++)
        {
            Write.Write((char)10);
            for (int j = 0; j < n; j++)
            {
                for (int k = 0; k < m; k++)
                {
                    Write.Write(arr[i, j, k]);
                    Write.Write(' ');
                }
                Write.Write((char)10);
            }
        }
    }
}

void Positioning(int segment, int line, IDisposable writeRead)
{
    int position = (segment + 1) + //Отступы м\у сегментами
(segment * (CountLines * ((CountElements * 4) + 5))) + //Пропуск эл. до
нужного сегмента
(line * ((CountElements * 4) + 5)); //Пропуск эл. до нужной
строки

//Позиция каретки с учетом размеров
switch (writeRead)
{
    case BinaryWriter writer:
        writer.BaseStream.Position = position;
        break;
    case BinaryReader reader:
        reader.BaseStream.Position = position;
        break;
}
}

public int[] ReadLine(int segment, int line)
{
    int[] dataFromFile = new int [CountElements];

    using (Read = new BinaryReader(new FileStream(filename,
        FileMode.Open)))
    {
        Positioning(segment, line, Read);
        for (int i = 0; i < CountElements; i++)
        {
            dataFromFile[i] = Read.ReadInt32();
            Read.BaseStream.Position++;
        }
    }
    return dataFromFile;
}

```

```

//Записать строку temp в строку line в сегменте
segment public void WriteLine(int segment, int
line, int[] temp) {
    using (Write = new BinaryWriter(new FileStream(filename,
        FileMode.Open)))
    {
        Positioning(segment, line, write);

        for (int i = 0; i < CountElements; i++)
        {
            write.Write(temp[i]);
            write.Write(' ');
        }
    }
}
}
}

```

Controller.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Cachework
{
    class Controller
    {
        public static MainMemory memory;
        public static Cache cache;
        public static int[, ] arr;
        bool isCache = false; // данные загружены из Кэш памяти

        public Controller(int countPages, int countLines, int countElements,
string filename)
        {
            memory = new MainMemory(filename + ".txt", countPages,
countLines, countElements);
            arr = new int[countPages, countLines, countElements];
            memory.RandomArray(arr, countPages, countLines,
countElements); memory.WriteArray(arr, countPages,
countLines, countElements); cache = new Cache(countLines,
countElements);
        }

        public int this[int i, int j, int k]
        {
            get
            {
                return arr[i, j, k];
            }
        }

        public int this [int i, int j]
        {
            get
            {
                return cache[i, j];
            }
        }

        public int this[int i]
        {

```

```

        get
        {
            return cache[i];
        }
    }

    public bool IsCache
    {
        get
        {
            return isCache;
        }
    }

    /  <summary>
    /  Поиск строки в кэше, либо в ОП
    /  </summary>
    public int[] SearchLine(int indexPage, int indexLine)
    {
        int[] buf = new int[memory.CountElements];

        // если строка с индексом indexLine находится в кэше с тэгом,
        // равному
        indexPage
        / считываем строку из кэша
        if (cache.isThereATag(indexPage, indexLine))
        {
            for (int i = 0; i < memory.CountElements; i++)
            {
                buf[i] = cache[indexLine, i];
            }
            isCache = true;
            return buf;
        }
        isCache = false;

        / иначе считываем строку из ОП
        buf = memory.ReadLine(indexPage, indexLine);

        / если данный тэг уже занят другой строкой, то эту строку нужно
        / скопировать и записать в файл (и в массив)
        if (cache[indexLine] != -1)
        {
            int[] old_str = new
            int[memory.CountElements]; for (int i = 0; i
            < memory.CountElements; i++)
            {
                old_str[i] = cache[indexLine, i];
            }
            / вернем строку в файл в нужную страницу
            memory.WriteLine(cache[indexLine], indexLine,
            old_str);
            / вернем строку в массив в нужную страницу
            memory.SetLineOnArray(ref arr, old_str, cache[indexLine],
            indexLine, memory.CountElements);
        }
        cache[indexLine] = indexPage; // присваиваем новому тэгу значение
        SetLineOnCache(buf, memory.CountElements, indexLine); // добавляем
        строку в
        кэш

        return buf;
    }

    /  <summary>
    /  Записать строку в кэш
    /  </summary>
    public void SetLineOnCache(int [] buf, int countElements, int indexLine)
    {

```

```

        cache.WriteLine(buf, memory.CountElements, indexLine);
    }
}
}

```

Form1.cs:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;

namespace Cachework
{
    public partial class Form1 : Form
    {
        Controller controller;
        int AdresI, AdresJ, AdresK;
        int CountPages,
            CountLines,
            CountElements;
        int Value;
        int[] str;

        Stopwatch Time = new Stopwatch();
        public Form1()
        {
            InitializeComponent();
            CountPages = Convert.ToInt32(textBox_Page.Text);
            CountLines = Convert.ToInt32(textBox_String.Text);
            CountElements = Convert.ToInt32(textBox_Items.Text);
            controller = new Controller(CountPages, CountLines,
CountElements, "MainMemory");
            label_whereFrom.Text = "Ни один элемент пока не был
загружен"; label_whereFrom.ForeColor = Color.Black;
            button_Create.Enabled = false;
            WriteToTextBox_OP();
            WriteToTextBox_Cache();
            textBox_OP.ReadOnly = true;
            textBox_Cache.ReadOnly = true;
        }

        private void button_change_Click(object sender, EventArgs e)
        {
            try
            {
                AdresI =
                    Convert.ToInt32(textBox_PageSearch.Text);
                AdresJ =
                    Convert.ToInt32(textBox_StringSearch.Text);
                AdresK =
                    Convert.ToInt32(textBox_ItemSearch.Text); Value
                    = Convert.ToInt32(textBox_ItemChange.Text);
            }
            catch
            {
                MessageBox.Show("Данные для поиска введены
некорректно!"); return;
            }

            Time.Start();

```

```

str = controller.SearchLine(AdresI,
AdresJ); Time.Stop();

if (controller.IsCache)
{
    label_whereFrom.Text = "Элемент загружен из
Кэша"; label_whereFrom.ForeColor = Color.Red;
}
else
{
    label_whereFrom.Text = "Элемент загружен из
Он"; label_whereFrom.ForeColor =
Color.Blue;
}

str[AdresK] = value; // сохраняем нужный элемент
controller.SetLineOnCache(str, CountElements,
AdresJ); label_ItemFrom.Text = value.ToString();
label_StringFrom.Text = " ";
for (int i = 0; i < CountElements; i++)
{
    label_StringFrom.Text += str[i].ToString() + " ";
}

WriteToTextBox_Cache();
WriteToTextBox_OP();
label_TimeFrom.Text =
Time.Elapsed.ToString(); Time.Reset();
}

private void button_Search_Click(object sender, EventArgs e)
{
    try
    {
        AdresI = Convert.ToInt32(textBox_PageSearch.Text);
        AdresJ =
        Convert.ToInt32(textBox_StringSearch.Text); AdresK
        = Convert.ToInt32(textBox_ItemSearch.Text);
    } catch
    {
        MessageBox.Show("Данные для поиска введены
некорректно!"); return;
    }

    Time.Start();
    str = controller.SearchLine(AdresI,
AdresJ); Time.Stop();

    if (controller.IsCache)
    {
        label_whereFrom.Text = "Элемент загружен из
Кэша"; label_whereFrom.ForeColor = Color.Red;
    } else
    {
        label_whereFrom.Text = "Элемент загружен из
Он"; label_whereFrom.ForeColor =
Color.Blue;
    }

    value = str[AdresK]; // загружаем нужный
элемент label_ItemFrom.Text =
value.ToString(); label_StringFrom.Text = "
";
    for (int i = 0; i < CountElements; i++)
    {
        label_StringFrom.Text += str[i].ToString() + " ";
    }
}

```

```

        WriteToTextBox_Cache();
        WriteToTextBox_OP();
        label_TimeFrom.Text =
        Time.Elapsed.ToString(); Time.Reset();
    }

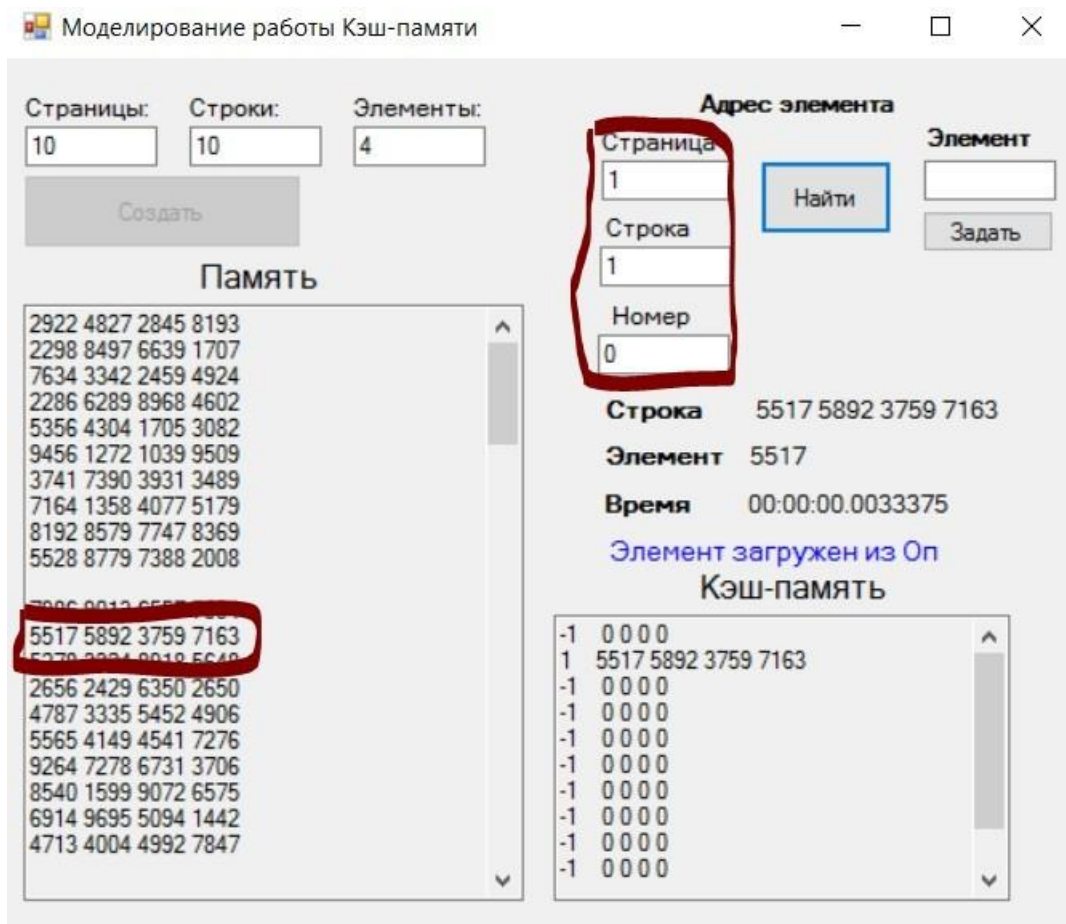
    private void button_Create_Click(object sender, EventArgs e)
    {
        WriteToTextBox_OP();
    }

    private void WriteToTextBox_OP()
    {
        textBox_OP.Text = "";
        for (int i = 0; i < CountPages; i++)
        {
            for (int j = 0; j < CountLines; j++)
            {
                for (int k = 0; k < CountElements; k++)
                {
                    textBox_OP.Text += (controller[i, j, k].ToString() + " ");
                }
                textBox_OP.Text += Environment.NewLine;
            }
            textBox_OP.Text += Environment.NewLine;
        }
        textBox_OP.Text += Environment.NewLine;
    }

    private void WriteToTextBox_Cache()
    {
        textBox_Cache.Text = "";
        for (int i = 0; i < CountLines; i++)
        {
            textBox_Cache.Text += controller[i] + "
"; for (int j = 0; j < CountElements; j+
+) {
                textBox_Cache.Text += controller[i, j].ToString() + " ";
            }
            textBox_Cache.Text += Environment.NewLine;
        }
        textBox_Cache.Text += Environment.NewLine;
    }
}
}
}

```


При запуске программы файл ОП заполняется случайными значениями



[illegible]

