

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;
    }

```



```

        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
            Оп"; 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
            Оп"; 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;
    }
}
}
}

```

Результаты работы

При запуске программы файл ОП заполняется случайными значениями

Моделирование работы Кэш-памяти

Страницы: 10

Строки: 10

Элементы: 4

Создать

Память

2922 4827 2845 8193
2298 8497 6639 1707
7634 3342 2459 4924
2286 6289 8968 4602
5356 4304 1705 3082
9456 1272 1039 9509
3741 7390 3931 3489
7164 1358 4077 5179
8192 8579 7747 8369
5528 8779 7388 2008

7986 9013 6557 7034
5517 5892 3759 7163
5278 3324 8918 5648
2656 2429 6350 2650
4787 3335 5452 4906
5565 4149 4541 7276
9264 7278 6731 3706
8540 1599 9072 6575
6914 9695 5094 1442
4713 4004 4992 7847

Адрес элемента

Страница

Строка

Номер

Найти

Задать

Элемент

Строка -----
Элемент ----
Время 00:00:00
Ни один элемент пока не был загр

Кэш-память

-1 0 0 0 0
-1 0 0 0 0
-1 0 0 0 0
-1 0 0 0 0
-1 0 0 0 0
-1 0 0 0 0
-1 0 0 0 0
-1 0 0 0 0
-1 0 0 0 0
-1 0 0 0 0

Загрузили элемент из ОП

Загру

Моделирование работы Кэш-памяти

[illegible]

Задали новое значение элементу, загруженного из Кэша

