

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 памяти
        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

Адрес элемента

Страница

Строка

Номер

Элемент

Задать

Найти

Ни один элемент пока не был загружен

Строка

Элемент

Время

00:00:00

Кэш-память

-1 0000

-1 0000

-1 0000

-1 0000

-1 0000

-1 0000

-1 0000

-1 0000

-1 0000

-1 0000

Основная память

9622 4028 3386 4010

4565 8099 1229 1527

8139 2182 4660 7145

7467 2704 3165 8327

6327 5865 9039 5816

5729 4322 4641 8142

8437 6653 4517 3192

9686 5257 5505 2631

9474 4974 5517 2940

3962 1949 1880 4690

7787 5895 9842 4534

5027 3786 9764 1941

9904 1685 9504 6498

1247 5868 7509 6300

9386 5243 5572 7911

4581 8036 1839 3882

6675 3029 7669 3493

7185 1225 5365 5377

4985 5348 8028 9832

Загрузили элемент из ОП

Моделирование работы Кэш-памяти

Страницы: 10

Строки: 10

Элементы: 4

Создать

Адрес элемента

Страница

Строка

Номер

Элемент

Задать

Найти

Элемент загружен из Оп

Строка

Элемент

Время

00:00:00.0047476

Кэш-память

-1 0000

1 5027 3786 9764 1941

-1 0000

-1 0000

-1 0000

-1 0000

-1 0000

-1 0000

-1 0000

-1 0000

Основная память

9622 4028 3386 4010

4565 8099 1229 1527

8139 2182 4660 7145

7467 2704 3165 8327

6327 5865 9039 5816

5729 4322 4641 8142

8437 6653 4517 3192

9686 5257 5505 2631

9474 4974 5517 2940

3962 1949 1880 4690

7787 5895 9842 4534

5027 3786 9764 1941

9904 1685 9504 6498

1247 5868 7509 6300

9386 5243 5572 7911

4581 8036 1839 3882

6675 3029 7669 3493

7185 1225 5365 5377

4985 5348 8028 9832

Загрузили элемент из ОП.

[illegible]

Задали новое значение элементу, загруженного из Кэша

[illegible]

Загрузили элемент строки из ОП, при этом место в Кэше, занятое другой строкой, освободилось и измененная строка в Кэше поместилась обратно в ОП.

Моделирование работы Кэш-памяти

Страницы: 10

Строки: 10

Элементы: 4

Создать

Адрес элемента

Страница 2

Строка 0

Номер 0

Элемент 1111

Задать

Найти

Элемент загружен из Оп

Строка 6888 3576 3286 9001

Элемент 6888

Время 00:00:00.0238131

Кэш-память

2 6888 3576 3286 9001

1 5027 3786 9764 1941

-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

Основная память

3962 1949 1880 4690

1111 5895 9842 4534

5027 3786 9764 1941

9904 1685 9504 6498

1247 5868 7509 6300

9386 5243 5572 7911

4581 8036 1839 3882

6675 3029 7669 3493

7185 1225 5365 5377

4985 5348 8028 9832

4478 1106 4114 5650

6888 3576 3286 9001

5752 9005 2383 4606

7464 9216 3552 8718

6375 3967 1109 9749

5056 6960 7857 2998

6115 7270 4754 9679

6319 5126 2512 5897

Загрузим измененный ранее элемент из ОП.

Моделирование работы Кэш-памяти

Страницы: 10

Строки: 10

Элементы: 4

Создать

Адрес элемента

Страница 1

Строка 0

Номер 0

Элемент

Задать

Найти

Элемент загружен из Оп

Строка 1111 5895 9842 4534

Элемент 1111

Время 00:00:00.0081301

Кэш-память

1 1111 5895 9842 4534

1 5027 3786 9764 1941

-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

Основная память

9622 4028 3386 4010

4565 8099 1229 1527

8139 2182 4660 7145

7467 2704 3165 8327

6327 5865 9039 5816

5729 4322 4641 8142

8437 6653 4517 3192

9686 5257 5505 2631

9474 4974 5517 2940

3962 1949 1880 4690

1111 5895 9842 4534

5027 3786 9764 1941

9904 1685 9504 6498

1247 5868 7509 6300

9386 5243 5572 7911

4581 8036 1839 3882

6675 3029 7669 3493

7185 1225 5365 5377

4985 5348 8028 9832