# ЛАБОРАТОРНАЯ РАБОТА 14. ФАЙЛОВЫЙ ВВОД-ВЫВОД. РАБОТА С КАТАЛОГАМИ. РАБОТА С ФАЙЛАМИ.

## 1. Цель и содержание

Цель лабораторной работы: научиться использовать механизмы файлового ввода-вывода.

Задачи лабораторной работы:

– научиться применять классы для работы с файлами;

– научиться применять классы для работы с каталогами;

– научиться использовать потоки ввода-вывода.

## 2. Формируемые компетенции

Лабораторная работа направлена на формирование следующих компетенций:

– способность к проектированию базовых и прикладных информационных технологий (ПК-11);

– способность разрабатывать средства реализации информационных технологий (методические, информационные, математические, алгоритмические, технические и программные) (ПК-12).

## 3. Теоретическая часть

### 3.1 Классы .NET Framework для реализации операций ввода-вывода.

Весь ввод и вывод в .NET Framework подразумевает использование потоков. Поток – это абстрактное представление последовательного устройства. Последовательное устройство – это нечто такое, что хранит данные в линейной структуре и точно таким же образом обеспечивает доступ к ним: считывает или записывает по одному байту за одну единицу времени.

Сохранение устройства абстрактным означает, что лежащие в основе источник/приемник данных могут быть скрыты. Такой уровень абстракции обеспечивает повторное использование кода и позволяет писать более обобщенные процедуры, потому что нет необходимости заботиться о действительной специфике передачи данных.

Для обработки файлов в C# необходима ссылка на пространство имен System.IO. При открытии файла создается объект, с которым ассоциируется поток. Потоки обеспечивают механизмы связи между файлами и программами.

Среда .NET Framework предоставляет все необходимые инструменты для эффективного использования файлов в приложениях.

Основные классы, необходимые программисту:

1. Object – исходный базовый класс для всех классов платформы .NET Framework и корень иерархии типов.

2. File – статический служебный класс, предоставляющий множество статических методов, которые дают возможность перемещать, создавать, копировать, удалять файлы, опрашивать и обновлять атрибуты, а также создавать объекты потоков FileStream.

3. Directory – статический служебный класс, предоставляющий множество статических методов для перемещения, копирования и удаления каталогов.

4. Path – служебный класс, используемый для манипулирования путевыми именами.

5. MarshalByRefObject – разрешает доступ к объектам через границы доменов приложения в приложениях, поддерживающих удаленное взаимодействие, это базовый класс для всех классов .NET, позволяющих удаленное взаимодействие.

6. FileInfo – представляет физический файл на диске, имеет методы для манипулирования этим файлом. Для любого объекта, который читает или пишет в этот файл, должен быть создан объект Stream. Все методы FileInfo доступны из объектной переменной, поэтому, если необходимо выполнить

только одно действие, более эффективным может оказаться использование метода File, а не соответствующего экземпляра метода FileInfo. Для всех методов FileInfo требуется путь к файлу, с которым проводится операция. Все статические методы класса FileInfo выполняют проверку безопасности для всех методов. Если необходимо использовать объект неоднократно, рекомендуется использовать соответствующий метод экземпляра FileInfo, поскольку в этом случае проверка безопасности будет требоваться не всегда.

7. DirectoryInfo – представляет физический каталог на диске и предоставляет методы уровня экземпляра для манипулирования каталогом. Класс DirectoryInfo работает точно так же, как класс FileInfo. Это объект, представляющий отдельный каталог на машине. Подобно классу FileInfo, многие из вызовов методов дублируются между Directory и DirectoryInfo.

8. FileSystemInfo – служит базовым классом для FileInfo и DirectoryInfo, обеспечивая возможность работы с файлами и каталогами одновременно, используя полиморфизм.

9. Stream – предоставляет универсальное представление последовательности байтов. Класс Stream является абстрактным базовым классом всех потоков.

10. FileStream – представляет файл, который может быть записан, прочитан или то и другое.

11. TextReader – представляет средство чтения, позволяющее считывать последовательные наборы знаков. Этот класс является абстрактным базовым классом для StreamReader, который считывает символы из потоков.

12. TextWriter – представляет средство записи, позволяющее записывать последовательные наборы символов. Этот класс является абстрактным базовым классом для StreamWriter, который записывают символы в потоки.

13. StreamReader – читает символьные данные из потока и может быть создан с использованием класса FileStream в качестве базового.

14. StreamWriter – пишет символьные данные в поток и может быть создан с использованием класса FileStream в качестве базового.

15. Component – предоставляет базовую реализацию интерфейса IComponent и делает возможным совместное использование объектов разными приложениями.

16. FileSystemWatcher – используется для мониторинга файлов и каталогов и представляет события, которые приложение может перехватить, когда в этих объектах происходят какие-то изменения.

Таким образом, эта система классов включает в себя классы для работы с файлами (File, FileInfo), каталогами (Directory, DirectoryInfo) и потоками (FileStream, StreamReader, StreamWriter).

В большинстве случаев для разработки бизнес-приложений достаточно лишь четырех классов для манипулирования файловой системой. Эти классы расположены пространстве имен System.IO и предназначены для работы с файловой системой компьютера, то есть для создания, удаления переноса файлов и каталогов.

Первые два типа – Directory и Fi1e реализуют свои возможности с помощью статических методов, поэтому данные классы можно использовать без создания соответствующих объектов (экземпляров классов).

Следующие типы – DirectoryInfo и FileInfo обладают схожими функциональными возможностями с Directory и Fi1e, но порождены от класса FileSystemInfo, поэтому реализуются путем создания соответствующих экземпляров классов.

Класс FileSystemInfo предоставляет базовый функционал. Значительная часть членов FileSystemInfo предназначена для работы с общими характеристиками файла или каталога (метками времени, атрибутами и т. п.). Рассмотрим некоторые свойства FileSystemInfo (таблица 19.1).

Таблица 19.1 – Свойства класса FileSystemInfo.

| Свойство | Описание |
|---|---|
| Attributes | Позволяет получить или установить атрибуты для данного объекта файловой системы. Для этого свойства используются значения и перечисления FileAttributes |

| Свойство | Описание |
|---|---|
| CreationTime | Позволяет получить или установить время создания объекта файловой системы |
| Exists | Может быть использовано для того, чтобы определить, существует ли данный объект файловой системы |
| Extension | Позволяет получить расширение для файла |
| FullName | Возвращает имя файла или каталога с указанием пути к нему в файловой системе |
| LastAccessTime | Позволяет получить или установить время последнего обращения к объекту файловой системы |
| LastWriteTime | Позволяет получить или установить время последнего внесения изменений в объект файловой системы |
| Name | Возвращает имя указанного файла. Это свойство доступно только для чтения. Для каталогов возвращает имя последнего каталога в иерархии, если это возможно. Если нет, возвращает полностью определенное имя |

В FileSystemInfo предусмотрен набор методов. Например, метод Delete() – позволяет удалить объект файловой системы с жесткого диска, а Refresh() – обновить информацию об объекте файловой системы.

3.2 Классы для работы с каталогами файловой системы.

Класс DirectoryInfo наследует члены класса FileSystemInfo и содержит дополнительный набор членов, которые предназначены для создания, перемещения, удаления, получения информации о каталогах и подкаталогах в файловой системе. Наиболее важные члены класса содержатся в таблице 2.2.

Таблица 19.2 – Доступные члены класса DirectoryInfo.

| Член | Описание |
|---|---|
| Create()<br>CreateSubDirectory() | Создают каталог (или подкаталог) по указанному пути в файловой системе |
| Delete() | Удаляет пустой каталог |
| GetDirectories() | Позволяет получить доступ к подкаталогам текущего каталога (в виде массива объектов DirectoryInfo ) |
| GetFiles() | Позволяет получить доступ к файлам текущего каталога |

| Член | Описание |
|---|---|
| | (в виде массива объектов FileInfo ) |
| MoveTo() | Перемещает каталог и все его содержимое на новый адрес в файловой системе |
| Parent | Возвращает родительский каталог в иерархии файловой системы |

Работа с типом DirectoryInfo начинается с того, что создается экземпляр класса (объект), при вызове конструктора в качестве параметра указывается путь к нужному каталогу. Если необходимо обратиться к текущему каталогу (то есть каталогу, в котором в настоящее время производится выполнение приложения), вместо параметра используется обозначение ".". Например:

```
// Создаем объект DirectoryInfo,
// которому будет обращаться к текущему каталогу
DirectoryInfo dir1 = new DirectoryInfo(".");

// Создаем объект DirectoryInfo,
// которому будет обращаться к каталогу d:\prim
DirectoryInfo dir2 = new DirectoryInfo(@"d:\prim");
```

Если создается объект DirectoryInfo, который связывается с несуществующим каталогом, то будет сгенерировано исключение System.IO.DirectoryNotFoundException.

Свойство Attributes класса DirectoryInfo позволяет получить информацию об атрибутах объекта файловой системы. Возможные значения данного свойства приведены в следующей таблице 19.3.

Таблица 19.3 – Значения свойства Attributes.

| Значение | Описание |
|---|---|
| Archive | Этот атрибут используется приложениями при проведении резервного копирования, а в некоторых случаях - удаления старых файлов |
| Compressed | Определяет, что файл является сжатым |
| Directory | Определяет, что объект файловой системы является каталогом |
| Encrypted | Определяет, что файл является зашифрованным |
| Hidden | Определяет, что файл является скрытым (такой файл не будет выводиться при обычном просмотре каталога) |
| Normal | Определяет, что файл находится в обычном состоянии и для него установлены любые другие атрибуты. Этот атрибут не может |

| Значение | Описание |
|---|---|
| | использоваться с другими атрибутами |
| Offline | Файл (расположенный на сервере) кэширован в хранилище off-line на клиентском компьютере. Возможно, что данные этого файла уже устарели |
| Readonly | Файл доступен только для чтения |
| System | Файл является системным (то есть файл является частью операционной системы или используется исключительно операционной системой) |

Через класс DirectoryInfo программист может собрать информацию о дочерних подкаталогах. Например:

```
static void printDirect(DirectoryInfo dir)
{
    Console.WriteLine("***** " + dir.Name + " *****");
    Console.WriteLine("FullName: {0}", dir.FullName);
    Console.WriteLine("Name: {0}", dir.Name);
    Console.WriteLine("Parent: {0}", dir.Parent);
    Console.WriteLine("Creation: {0}", dir.CreationTime);
    Console.WriteLine("Attributes: {0}", dir.Attributes.ToString());
    Console.WriteLine("Root: {0}", dir.Root);
}

static void Main(string[] args)
{
    DirectoryInfo dir = new DirectoryInfo(@"d:\prim");
    printDirect(dir);
    DirectoryInfo[] subDirects = dir.GetDirectories();
    Console.WriteLine("Найдено {0} подкаталогов", subDirects.Length)
    foreach (DirectoryInfo d in subDirects)
    {
        printDirect(d);
    }
}
```

Метод CreateSubdirectory() позволяет создать в выбранном каталоге как единственный подкаталог, так и множество подкаталогов (в том числе, и вложенных друг в друга). Например:

```
DirectoryInfo dir = new DirectoryInfo(@"d:\prim");

//создали подкаталог
dir.CreateSubdirectory("doc");

//создали вложенный подкаталог
dir.CreateSubdirectory(@"book\2008");
```

Метод MoveTo() позволяет переместить текущий каталог по заданному в качестве параметра адресу. При этом возможно произвести переименование каталога. Например:

```
DirectoryInfo dir = new DirectoryInfo(@"d:\prim\bmp");
dir.MoveTo(@"d:\prim\letter\Николаев");
```

В данном случае каталог «bmp» перемещается по адресу «d:\prim\letter\Николаев». Так как имя перемещаемого каталога не совпадает с крайним правым именем в адресе нового местоположения каталога, то производится переименование.

Работать с каталогами файловой системы компьютера можно и при помощи класса Directory, функциональные возможности которого во многом совпадают с возможностями DirectoryInfo. Следует учитывать, что члены данного класса реализованы статически, поэтому для их использования нет необходимости создавать объект. Например:

```
// Создание подкаталога Новый
Directory.CreateDirectory(@"d:\prim\Новый");

// Перемещение каталога Новый в каталог 2012
Directory.Move(@"d:\prim\Новый",
               @"d:\prim\2012\Новый");

//переименовали каталог Новый в Старый
Directory.Move(@"d:\prim\2012\Новый",
               @"d:\prim\2012\Старый");
```

Следует учитывать, что удаление каталога возможно только когда он пуст. На практике комбинируют использование классов Directory и DirectoryInfo.

### 3.3 Классы для работы с файлами.

Класс FileInfo предназначен для организации доступа к физическому файлу, который содержится на жестком диске компьютера. Он позволяет получать информацию об этом файле (например, о времени его создания, размере, атрибутах), а также производить различные операции, например, по созданию файла или его удалению. Класс FileInfo наследует члены класса FileSystemInfo и содержит дополнительный набор членов, который приведен в следующей таблице 19.4

Таблица 19.4 – Члены класса FileInfo.

| Член | Описание |
|---|---|
| AppendText() | Создает объект StreamWriter для добавления текста к файлу. |
| CopyTo() | Копирует уже существующий файл в новый файл. |
| Create() | Создает новый файл и возвращает объект FileStream для взаимодействия с этим файлом. |
| CreateText() | Создает объект StreamWriter для записи текстовых данных в новый файл. |
| Delete() | Удаляет файл, которому соответствует объект FileInfo. |
| Directory | Возвращает каталог, в котором расположен данный файл. |
| DirectoryName | Возвращает полный путь к данному файлу в файловой системе. |
| Length | Возвращает размер файла. |
| MoveTo() | Перемещает файл в указанное пользователем место (этот метод позволяет одновременно переименовать данный файл). |
| Name | Позволяет получить имя файла. |
| Open() | Открывает файл с указанными пользователем правами доступа на чтение, запись или совместное использование с другими пользователями. |
| OpenRead() | Создает объект FileStream, доступный только для чтения. |
| OpenText() | Создает объект StreamReader (о нем также будет рассказано ниже), который позволяет считывать информацию из существующего текстового файла. |
| OpenWrite() | Создает объект FileStream, доступный для чтения и записи. |

Большинство методов FileInfo возвращает объекты классов FileStream, StreamWriter, StreamReader и т. п., которые позволяют различным образом взаимодействовать с файлом, например, производить чтение или запись в него. Например:

```
// Создание объекта FileInfo
FileInfo f = new FileInfo("text.txt");

// Создание файла и потока
StreamWriter fOut =
    new StreamWriter(f.Create());

// Запись текста в файл
fOut.WriteLine("ОДИН ДВА ТРИ...");

// Закрытие файла
fOut.Close();

// Получение информации о файле
Console.WriteLine("Name: {0}", f.Name);
Console.WriteLine("File size: {0}",
    f.Length);
Console.WriteLine("Creation: {0}",
    f.CreationTime);
Console.WriteLine("Attributes: {0}",
    f.Attributes.ToString());
```

Доступ к физическим файлам можно получать и через статические методы класса File. Большинство методов объекта FileInfo представляют в этом смысле зеркальное отражение методов объекта File.

## 3.4 Потоки в системе ввода-вывода.

Программы на языке C# выполняют операции ввода-вывода посредством потоков, которые построены на иерархии классов. Поток (stream) – это абстракция, которая генерирует и принимает данные. С помощью потока можно читать данные из различных источников (клавиатура, файл, память) и записывать в различные источники (принтер, экран, файл, память).

Центральную часть потоковой системы C# занимает класс Stream пространства имен System.IO. Класс Stream представляет байтовый поток и является базовым для всех остальных потоковых классов. Производными от класса Stream являются классы потоков:

1. FileStream – байтовый поток, разработанный для файлового ввода-вывода.

2. BufferedStream – заключает в оболочку байтовый поток и добавляет буферизацию, которая во многих случаях увеличивает производительность программы.

3. MemoryStream – байтовый поток, который использует память для хранения данных.

Программист может реализовать собственные потоковые классы. Однако для подавляющего большинства приложений достаточно встроенных потоков.

### 3.4.1 Байтовый поток.

Чтобы создать байтовый поток, связанный с файлом, создается объект класса FileStream. При этом в классе определено несколько конструкторов. Чаще всего используется конструктор, который открывает поток для чтения и (или) записи:

```
public FileStream(string path, FileMode mode);
```

Параметр path определяет имя файла, с которым будет связан поток ввода-вывода данных. Параметр mode определяет режим открытия файла, который может принимать одно из возможных значений, определенных перечислением FileMode:

– FileMode.Append – предназначен для добавления данных в конец файла;

– FileMode.Create – предназначен для создания нового файла, при этом если существует файл с таким же именем, то он будет предварительно удален;

– FileMode.CreateNew – предназначен для создания нового файла, при этом файл с таким же именем не должен существовать;

– FileMode.Open – предназначен для открытия существующего файла;

– FileMode.OpenOrCreate – если файл существует, то открывает его, в противном случае создает новый;

– FileMode.Truncate – открывает существующий файл, но усекает его длину до нуля.

Если попытка открыть файл оказалась неудачной, то генерируется одно из исключений:

– FileNotFoundException – файл невозможно открыть по причине его отсутствия;

– IOException – файл невозможно открыть из-за ошибки ввода-вывода;

– ArgumentNullException – имя файла представляет собой null-значение;

– ArgumentException – некорректен параметр mode;

– SecurityException – пользователь не обладает правами доступа;

– DirectoryNotFoundException – некорректно задан каталог.

Другая версия конструктора позволяет ограничить доступ только чтением или только записью:

```
public FileStream(string path, FileMode mode, FileAccess access);
```

Параметры path и mode имеют то же назначение, что и в предыдущей версии конструктора. Параметр access, определяет способ доступа к файлу и может принимать одно из значений, определенных перечислением FileAccess:

– FileAccess.Read – только чтение;

– FileAccess.Write – только запись;

– FileAccess.ReadWrite – и чтение, и запись.

После установления связи байтового потока с физическим файлом внутренний указатель потока устанавливается на начальный байт файла.

Для чтения очередного байта из потока, связанного с физическим файлом, используется метод ReadByte( ). После прочтения очередного байта внутренний указатель перемещается на следующий байт файла. Если достигнут конец файла, то метод ReadByte( ) возвращает значение -1.

Для побайтовой записи данных в поток используется метод WriteByte( ).

По завершении работы с файлом его необходимо закрыть. Для этого достаточно вызвать метод Close( ). При закрытии файла освобождаются системные ресурсы, ранее выделенные для этого файла, что дает возможность использовать их для работы с другими файлами.

```
static void Main(string[] args)
{
    try
    {
        // Создание потока для чтения из файла
        FileStream fileIn =
            new FileStream("ФайлСТекстом.txt",
                FileMode.Open,
                FileAccess.Read);
        // Создание потока для записи в файл
        FileStream fileOut =
            new FileStream("ФайлКопия.txt",
                FileMode.Create,
                FileAccess.Write);
        int i;

        // В цикле производится чтение одного файла
        // и одновременная запись содержимого
        // во второй файл
        while ((i = fileIn.ReadByte()) != -1)
        {
            // запись очередного байта в поток
            fileOut.WriteByte((byte)i);
        }

        // Закрыте файлов
        fileIn.Close();
        fileOut.Close();
    }
    catch (Exception EX)
    {
        Console.WriteLine(EX.Message);
    }
}
```

В данном примере создаются два потока fileIn (для чтения байтов из файла в поток) и fileOut (для записи байтов из потока в файл). Каждый из потоков ассоциируется со своим файлом. Затем в цикле производится побайтовое чтение файла «ФайлСТекстом.txt» до момента, когда функция ReadByte( ) вернет значение -1 (то есть достигнут конец файла). При этом на каждой итерации цикла считывается один байт и на этой же итерации этот байт записывается в файл «ФайлКопия.txt». После всех операция оба потока закрываются. Весь код, осуществляющий работу с файлами заключен в конструкцию try … catch. Это позволяет повысить устойчивость программы к ошибкам.

### 3.4.2 Символьный поток.

Чтобы создать символьный поток нужно поместить объект класса Stream (например FileStream) внутрь объекта класса StreamWriter или объекта класса StreamReader. В этом случае байтовый поток будет автоматически преобразовываться в символьный.

Класс StreamWriter предназначен для организации выходного символьного потока. В нем определено несколько конструкторов. Один из них записывается следующим образом:

```
public StreamWriter(Stream stream);
```

Параметр stream определяет имя уже открытого байтового потока. Этот конструктор генерирует исключение типа ArgumentException, если поток stream не открыт для вывода, и исключение типа ArgumentNullException, если он (поток) имеет null-значение.

Другой вид конструктора позволяет открыть поток сразу через обращения к файлу:

```
public StreamWriter(string path);
```

Параметр path определяет имя открываемого файла.

Например, создать экземпляр класса StreamWriter можно следующим образом:

```
StreamWriter fileOut =
    new StreamWriter(
        new FileStream(
            "ФайлНиколаева.txt",
            FileMode.Create,
            FileAccess.Write));
```

И еще один вариант конструктора StreamWriter:

```
public StreamWriter(string path, bool append);
```

Параметр path определяет имя открываемого файла, а параметр append может принимать значение true – если нужно добавлять данные в конец файла, или false – если файл необходимо перезаписать.

Например:

```
// Добавляем символы в конец файла
StreamWriter fileOut_1 =
    new StreamWriter("МойФ.txt", true);
```

Объявив таким образом переменную fileOut_1 для записи данных в поток можно обратиться к методу WriteLine:

```
// Использование WriteLine
fileOut_1.WriteLine("Строка для записи");
```

В данном случае для записи используется метод, аналогичный статическому методу класса Console. Это действительно схожие механизмы ввода-вывода.

Класс StreamReader предназначен для организации входного символьного потока. Один из его конструкторов выглядит следующим образом:

```
public StreamReader(Stream stream);
```

Параметр stream определяет имя уже открытого байтового потока.

Этот конструктор генерирует исключение типа ArgumentException, если поток stream не открыт для ввода. Создать экземпляр класса StreamReader можно следующим образом:

```
StreamReader fileIn =
    new StreamReader(
        new FileStream(
            "text.txt",
            FileMode.Open,
            FileAccess.Read));
```

Как и в случае с классом StreamWriter у класса StreamReader есть и другой вид конструктора, который позволяет открыть файл напрямую:

```
public StreamReader(string path);
```

Параметр path определяет имя открываемого файла. Обратиться к данному конструктору можно следующим образом:

```
StreamReader fileIn =
    new StreamReader
        (@"c:\temp\Николаев.txt");
```

В C# символы реализуются кодировкой Unicode. Для того, чтобы можно было обрабатывать текстовые файлы, содержащие русские символы, созданные, например, в Блокноте, рекомендуется вызывать следующий вид конструктора StreamReader:

```
StreamReader fileIn =
    new StreamReader(
        @"c:\temp\t.txt",
        Encoding.GetEncoding(1251));
```

Параметр Encoding.GetEncoding(1251) говорит о том, что будет выполняться преобразование из кода Windows-1251 (одна из модификаций кода ASCII, содержащая русские символы) в Unicode. Тип Encoding реализован в пространстве имен System.Text.

Для чтения данных из потока fileIn можно воспользоваться методом ReadLine. При этом если будет достигнут конец файла, то метод ReadLine вернет значение null.

Рассмотрим пример, в котором данные из одного файла копируются в другой, но уже с использованием классов StreamWriter и StreamReader.

```
static void Main(string[] args)
{
    // Создание символьных потоков
    StreamReader fileIn =
        new StreamReader(
            "ФайлСТекстом.txt",
            Encoding.GetEncoding(1251));
    StreamWriter fileOut =
        new StreamWriter(
            "НовыйФайл.txt",
            false);
    string line;

    // Построчное считывание
    while ((line = fileIn.ReadLine()) != null)
    {
        // ... и запись строки
        fileOut.WriteLine(line);
    }

    fileIn.Close();
    fileOut.Close();
}
```

В данном примере осуществляется копирование содержимого одного символьного файла в другой.

Таким образом, данный способ копирования одного файла в другой, дает тот же результат, что и при использовании байтовых потоков. Однако, его работа будет менее эффективной, т.к. будет тратиться дополнительное время на преобразование байтов в символы. У символьных потоков есть и свои преимущества. Например, можно использовать регулярные выражения для поиска заданных фрагментов текста в файле.

3.4.3 Перенаправление стандартных потоков.

Тремя стандартными потоками, доступ к которым осуществляется через свойства Console.Out, Console.In и Console.Error, могут пользоваться все программы, работающие в пространстве имен System. Свойство Console.Out относится к стандартному выходному потоку. По умолчанию это консоль. Например, при вызове метода Console.WriteLine() информация автоматически передается в поток Console.Out. Свойство Console.In относится к стандартному входному потоку, источником которого по умолчанию является клавиатура. Например, при вводе данных с клавиатуры информация автоматически передается потоку Console.In, к которому можно обратиться с помощью метода Console.ReadLine(). Свойство Console.Error относится к ошибкам в стандартном потоке, источником которого также по умолчанию является консоль. Однако эти потоки могут быть перенаправлены на любое совместимое устройство ввода-вывода, например, на работу с физическими файлами.

Перенаправить стандартный поток можно с помощью методов SetIn(), SetOut() и SetError(), которые являются членами класса Console:

```
static void SetIn(TextReader input);
static void SetOut(TextWriter output);
static void SetError(TextWriter output);
```

Пример перенаправления потоков представлен в следующей программе, демонстрирующей, что стандартный поток вывода перенаправляется в один файл, а поток ввода – в другой:

```
static void Main(string[] args)
{
    int[,] MyArray;
    StreamReader file = new StreamReader("input.txt");
    // Перенаправление на file
    Console.SetIn(file);
    string line = Console.ReadLine();
    string[] mas = line.Split(' ');
    int n = int.Parse(mas[0]);
    int m = int.Parse(mas[1]);
    MyArray = new int[n, m];
    for (int i = 0; i < n; i++)
    {
        line = Console.ReadLine();
        mas = line.Split(' ');
        for (int j = 0; j < m; j++)
        {
            MyArray[i, j] = int.Parse(mas[j]);
        }
    }
    PrintArray("Исходный массив:", MyArray, n, m);
    file.Close();
}

static void PrintArray(string a, int[,] mas, int n, int m)
{
    // Перенаправление на file
    StreamWriter file=new StreamWriter("output.txt");
    Console.SetOut(file);
    Console.WriteLine(a);
    for (int i = 0; i < n; i++)
    {
        for (int j=0; j<m; j++) Console.Write("{0} ", mas[i,j]);
        Console.WriteLine();
    }
    file.Close();
}
```

4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

Для выполнения лабораторной работы необходимо спроектировать многомодульное приложение, использующее файлы для ввода и вывода информации.

Например, требуется разработать приложение, которое позволяет:

– генерировать матрицу случайных чисел с сохранением ее в файл;

– считывать матрицу из файла;

– выводить матрицу на экран.

Для выполнения данного задания-примера достаточно в отдельном проекте реализовать класс. Этот проект следует скомпилировать в виде dll-файла. Затем, в основной программе просто необходимо использовать данную библиотеку.

Класс библиотеки назовем Matrix. В нем определим следующие функции:

```
class Matrix
{
    private float[,] matrix;
    int m, n;

    // Конструктор
    public Matrix()...

    // Генерация матрицы заданного размера
    public void GenerateMatrix(int M, int N)...

    // Сохранение сгенерированной матрицы в файл
    public void SaveMatrix(string pFileName)...

    // Загрузка сохраненной матрицы из файла
    public Boolean LoadMatrix(string pFileName)...

    // Вывод матрицы в консоль
    public void PrintMatrix()...
}
```

Рисунок 19.1 – Основные методы класса Matrix.

Листинг метода GenegateMatrix:

```
// Генерация матрицы заданного размера
public void GenerateMatrix(int M, int N)
{
    m = M; n = N;

    Random r = new Random(DateTime.Now.Millisecond);

    matrix = new float[M, N];

    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            matrix[i, j] = (float)r.Next(1000) / 973f;
}
```

Рисунок 19.2 – Метод для создания матрицы заданного размера и заполнения ее случайными числами.

Листинг метода SaveMatrix:

```
// Сохранение сгенерированной матрицы в файл
public void SaveMatrix(string pFileName)
{
    if (matrix.Length > 0)
    {
        if (File.Exists(pFileName))
            File.Delete(pFileName);

        FileInfo f = new FileInfo(pFileName);
        TextWriter tw = f.CreateText();

        tw.WriteLine(m.ToString());
        tw.WriteLine(n.ToString());

        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                tw.WriteLine(i.ToString() + " " + j.ToString() + " " + matrix[i, j].ToString("E10"));

        tw.Close();
    }
}
```

Рисунок 19.3 – Метод для сохранения матрицы в файл.

```
// Загрузка сохраненной матрицы из файла
public Boolean LoadMatrix(string pFileName)
{
    if (File.Exists(pFileName))
    {
        try
        {
            TextReader tr = File.OpenText(pFileName);
            m = Convert.ToInt32(tr.ReadLine());
            n = Convert.ToInt32(tr.ReadLine());

            matrix = new float[m, n];
            string line;
            string[] substring;

            for (int i = 0; i < m; i++)
                for (int j = 0; j < n; j++)
                {
                    line = tr.ReadLine();
                    substring = line.Split(new char[] {' '}, 3);
                    matrix[i, j] = Convert.ToSingle(substring[2]);
                }

            tr.Close();

            return true;

        }
        catch
        {
            return false;
        }
    }

    return false;
}
```

Рисунок 19.4 – Метод загрузки матрицы из файла.

На рис. 19.4 представлен листинг метода LoadMatrix. Листинг метода PrintMatrix:

```
// Вывод матрицы в консоль
public void PrintMatrix()
{
    if (matrix.Length > 0)
    {
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
                Console.Write(matrix[i, j].ToString("E3") + "  ");

            Console.WriteLine();
        }

    }
}
```

Рисунок 19.5 – Метод для вывода матрицы.

Основная программа для использования класса-матрицы представлена на рис. 19.6

```
class Program
{
    static void Main(string[] args)
    {
        Console.BackgroundColor = ConsoleColor.White;
        Console.ForegroundColor = ConsoleColor.Black;
        Console.Clear();

        Matrix m = new Matrix();

        m.GenerateMatrix(10, 5);
        m.SaveMatrix("FileForMatrix.txt");

        if (m.LoadMatrix("FileForMatrix.txt"))
            m.PrintMatrix();

        Console.ReadKey();
    }
}
```

Рисунок 19.6 – Работа с библиотечным классом.

В результате работы данного кода будет создан файл FileForMatrix.txt следующего содержания:

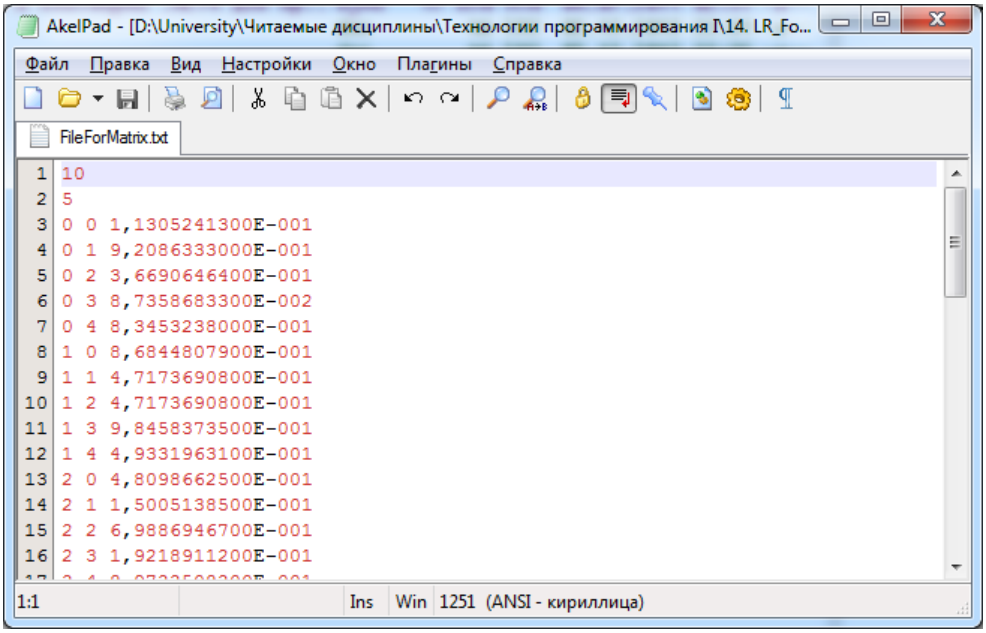


Рисунок 19.7 – Результирующий файл.

На экран будет выведена следующая информация:

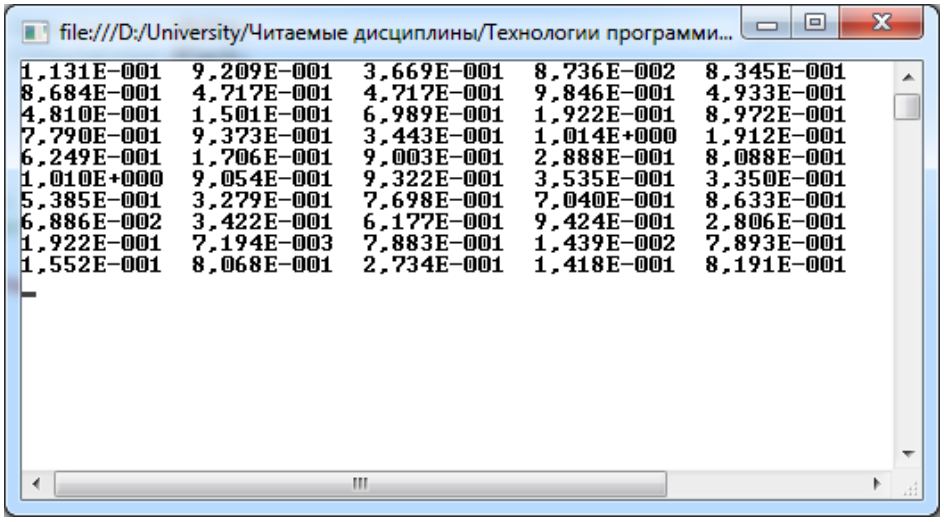


Рисунок 19.8 – Вывод программы в консоль.

Выполните индивидуальное задание, согласно предложенному варианту. В каждом варианте необходимо спроектировать многомодульное приложение (оптимально 2 модуля). Исходные данные в каждом варианте программа получает из входного файла.

**Индивидуальное задание.**

| Вариант | Структура данных |
|---|---|
| 1 | Программа рассчитывает произведение двух матриц, которые хранятся в разных файлах. |
| 2 | Программа рассчитывает сумму двух матриц, которые хранятся в разных файлах. |
| 3 | Программа находит максимальный элемент в двух матрицах, которые хранятся в разных файлах. |
| 4 | Программа рассчитывает сумму диагональных элементов двух матриц, которые хранятся в разных файлах. |
| 5 | Программа рассчитывает сумму элементов с четной суммой индексов в двух матрицах, которые хранятся в разных файлах. |
| 6 | Программа рассчитывает разность сумм элементов матриц, которые хранятся в разных файлах. |
| 7 | Программа рассчитывает сумму элементов главной диагонали первой матрицы и сумму элементов второстепенной диагонали второй матрицы. Матрицы хранятся в разных файлах. |
| 8 | Программа рассчитывает сумму элементов четных столбцов в двух матрицах, которые хранятся в разных файлах. |
| 9 | Программа рассчитывает сумму диагональных элементов четных столбцов в двух матрицах, которые хранятся в разных файлах. |
| 10 | Программа рассчитывает сумму элементов четных строк в двух матрицах, которые хранятся в разных файлах. |
| 11 | Программа рассчитывает сумму элементов , находящихся в четном столбце и нечетной строке, в двух матрицах, которые хранятся в разных файлах. |
| 12 | Программа рассчитывает сумму элементов четных строк и расположенных на второстепенной диагонали в обеих матрицах, которые хранятся в разных файлах. |
| 13 | Программа рассчитывает произведение элементов в двух матрицах, которые хранятся в разных файлах. |
| 14 | Программа рассчитывает сумму первой матрицы и матрицы транспонированной относительно второй. Обе матрицы хранятся в отдельных файлах. |
| 15 | Программа рассчитывает произведение элементов диагонали первой матрицы и сумму всех элементов второй матрицы. Обе матрицы хранятся в отдельных файлах. |
| 16 | Программа рассчитывает сумму элементов четных строк в двух матрицах, которые хранятся в разных файлах. |
| 17 | Программа находит максимальный элемент в двух матрицах, которые хранятся в разных файлах. |

| Вариант | Структура данных |
|---|---|
| 18 | Программа рассчитывает сумму элементов четных строк и расположенных на второстепенной диагонали в обеих матрицах, которые хранятся в разных файлах. |
| 19 | Программа рассчитывает сумму первой матрицы и матрицы транспонированной относительно второй. Обе матрицы хранятся в отдельных файлах. |
| 20 | Программа рассчитывает произведение двух матриц, которые хранятся в разных файлах. |
| 21 | Программа рассчитывает сумму элементов четных столбцов в двух матрицах, которые хранятся в разных файлах. |
| 22 | Программа рассчитывает сумму диагональных элементов четных столбцов в двух матрицах, которые хранятся в разных файлах. |
| 23 | Программа рассчитывает произведение элементов диагонали первой матрицы и сумму всех элементов второй матрицы. Обе матрицы хранятся в отдельных файлах. |
| 24 | Программа рассчитывает сумму элементов с четной суммой индексов в двух матрицах, которые хранятся в разных файлах. |
| 25 | Программа рассчитывает разность сумм элементов матриц, которые хранятся в разных файлах. |

## 7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

## 8. Контрольные вопросы

1. Какие классы для работы с файловой системой вы знаете?

2. Что такое сборка?

3. Как определить проект по умолчанию в многомодульном решении?

4. Какие классы отвечают за представление файлов в программе?

5. Что такое поток? Какие типы классов потоков используются при работе с файлами?

6. Опишите последовательность действий при необходимости записать одну строку в файл. Приведите примеры использования различных классов.

## 9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [5], [6-8].

# ЛАБОРАТОРНАЯ РАБОТА 15. РЕШЕНИЕ ВЫЧИСЛИТЕЛЬНОЙ ЗАДАЧИ С ПРИМЕНЕНИЕМ ФАЙЛОВОГО ВВОДА-ВЫВОДА.

## 1. Цель и содержание

Цель лабораторной работы: научиться использовать возможности файлового ввода-вывода для решения практических задач.

Задачи лабораторной работы:

– научиться проектировать приложение для реализации хранения данных программы;

– научиться производить выбор оптимальных инструментов для обеспечения сериализации.

## 2. Формируемые компетенции

Лабораторная работа направлена на формирование следующих компетенций:

– способность к проектированию базовых и прикладных информационных технологий (ПК-11);

– способность разрабатывать средства реализации информационных технологий (методические, информационные, математические, алгоритмические, технические и программные) (ПК-12).

## 3. Теоретическая часть

Перед выполнением лабораторной работы необходимо повторить теоретический материал лабораторной работы №19.

## 4. Оборудование и материалы