



---

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ**

**Брянский государственный технический университет**

---

Утверждаю

Ректор университета

\_\_\_\_\_ О.Н. Федонин

«\_\_\_\_\_» \_\_\_\_\_ 2013 г.

## **ЯЗЫКИ ПРОГРАММИРОВАНИЯ**

### **РАБОТА С ФАЙЛАМИ**

Методические указания  
к выполнению лабораторной работы  
для студентов очной формы обучения  
специальностей 090303 – «Информационная безопасность  
автоматизированных систем»,  
090900 – «Информационная безопасность»

**Брянск 2013**

УДК 004.43

Языки программирования. Работа с файлами: методические указания к выполнению лабораторной работы для студентов очной формы обучения специальностей 090303 – «Информационная безопасность автоматизированных систем», 090900 – «Информационная безопасность». – Брянск: БГТУ, 2013. – 28 с.

*Разработали:*

Ю.А. Леонов, к.т.н., доц.,

Е.А. Леонов, к.т.н., доц.

Рекомендовано кафедрой «Компьютерные технологии и системы»  
БГТУ (протокол № 2 от 19.09.2013)

## 1. ЦЕЛЬ РАБОТЫ

Целью работы является изучение основных классов предназначенных для работы с файлами, а также овладение практическими навыками составления алгоритмов с их использованием.

Продолжительность работы – 4 ч.

## 2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

При создании приложений часто требуется сохранять данные в постоянной памяти. Для этого можно использовать классы, предназначенные для работы с файлами и каталогами, которые располагаются в пространстве имен *System.IO*.

Приведем основные классы для работы с файлами и их назначение (табл. 1).

**Таблица 1**

**Основные классы для работы с файлами и каталогами**

Название	Описание
BinaryReader BinaryWriter	Позволяют сохранять и извлекать информацию встроенных типов данных (целочисленных, логических, строковых и т.п.) как двоичные значения.
BufferedStream	Осуществляет буферизацию в операциях чтения и записи в другие потоки.
Directory	Предоставляет статические методы для создания, перемещения и перечисления в каталогах и вложенных каталогах.
DirectoryInfo	Предоставляет методы экземпляра класса для создания, перемещения и перечисления в каталогах и подкаталогах.
File	Предоставляет статические методы для создания, копирования, удаления, перемещения и открытия файлов, а также помогает при создании объектов <i>FileStream</i> .
FileInfo	Предоставляет свойства и методы экземпляра для создания, копирования, удаления, перемещения и открытия файлов, а также позволяет создавать объекты <i>FileStream</i> .

Окончание табл. 1

Название	Описание
FileStream	Обеспечивает произвольный доступ к файлу, представляемому как поток байтов. Поддерживает синхронные и асинхронные операции чтения и записи.
MemoryStream	Обеспечивает произвольный доступ к потоку байтов в оперативной памяти.
StreamReader StreamWriter	Используются для считывания и записи текстовой информации в файл. Данные классы не поддерживают произвольный доступ к файлам.
StringReader StringWriter	Предназначены для работы с текстовой информацией, однако они используются для работы с буфером в оперативной памяти, а не файлом на диске.

Рассмотрим подробно особенности работы со следующими классами: *DirectoryInfo*, *FileInfo*, *FileStream*, *StreamReader*, *StreamWriter*.

### 2.1. Работа с каталогами (*DirectoryInfo*)

Класс *DirectoryInfo* содержит набор элементов класса унаследованных от классов *FileSystemInfo*, также он дополнен своими собственными элементами.

Для создания объекта данного класса необходимо воспользоваться конструктором:

```
public DirectoryInfo(string path),
```

например:

```
DirectoryInfo dir = new DirectoryInfo(@"C:\MyDir");
```

Рассмотрим основные свойства и методы для работы с классом *DirectoryInfo* (табл. 2).

Таблица 2

#### Основные свойства и методы класса *DirrectoryInfo*

Название	Описание
<b>Свойства</b>	
Attributes	Получает или задает атрибуты для текущего файла или каталога. Значение этого свойства представляет собой сочетание следующих флагов атрибутов файла: "архивный", "сжатый", "каталог", "скрытый", "автономный", "только для чтения", "системный" и "временный".

Окончание табл. 2

Название	Описание
CreationTime	Получает или задает время создания текущего файла или каталога.
Exists	Получает значение, определяющее наличие каталога.
Extension	Получает строку, содержащую расширение файла.
FullName	Получает полный путь к каталогу или файлу.
Name	Получает имя каталога.
Parent	Получает родительский каталог заданного подкаталога.
<b>Методы</b>	
Create	Создает каталог.
Delete	Удаляет каталог, если он пуст.
GetDirectories	Возвращает подкаталоги текущего каталога.
GetFiles	Возвращает список файлов текущего каталога.
MoveTo	Перемещает каталог и все его содержимое по новому адресу в файловой системе.

### Примеры использования свойств и методов

Почти каждый из рассматриваемых методов имеет множество вариантов реализации (перегрузок). Рассмотрим некоторые из реализаций представленных ранее методов.

#### *Attributes*

```
public FileAttributes Attributes { get; set; }
DirectoryInfo dir = new DirectoryInfo(@"C:\temp");
if ((dir.Attributes & FileAttributes.Directory) == FileAttributes.Directory)
    Console.WriteLine("Атрибут имеет значение: 'каталог'");
```

#### *CreationTime*

```
public DateTime CreationTime { get; set; }
DirectoryInfo dir = new DirectoryInfo(@"C:\temp");
Console.WriteLine(dir.CreationTime.ToString()); // На экране будет
дата и время создания каталога, например «10.12.2012 17:05:01»
```

#### *Exists*

```
public override bool Exists { get; }
DirectoryInfo dir = new DirectoryInfo(@"C:\temp");
if (dir.Exists)
```

```
Console.WriteLine("Такой каталог существует");
```

### ***Extension***

```
public string Extension { get; }
```

```
DirectoryInfo dir = new DirectoryInfo(@"C:\Temp\1.txt");
```

```
string s = dir.Extension; // s = «.txt»
```

### ***FullName***

```
public virtual string FullName { get; }
```

```
DirectoryInfo dir = new DirectoryInfo(@"C:\Temp\1.txt");
```

```
string s = dir.FullName; // s = «Y:\Temp\1.txt»
```

### ***Name***

```
public override string Name { get; }
```

```
DirectoryInfo dir = new DirectoryInfo(@"C:\Temp\1.txt");
```

```
string s = dir.Name; // s = «1.txt»
```

### ***Parent***

```
public DirectoryInfo Parent { get; }
```

```
DirectoryInfo dir = new DirectoryInfo(@"C:\Temp\1.txt");
```

```
string s = dir.Parent.ToString(); // s = «Temp»
```

### ***Create***

```
public void Create()
```

```
DirectoryInfo dir = new DirectoryInfo(@"C:\Temp\Новая папка");
```

```
dir.Create(); // Создает новый каталог
```

### ***Delete***

```
public override void Delete()
```

```
DirectoryInfo dir = new DirectoryInfo(@"C:\Temp\Новая папка");
```

```
dir.Delete(); // Удаляет каталог «Новая папка»
```

### ***GetDirectories***

```
public DirectoryInfo[] GetDirectories()
```

```
DirectoryInfo dir = new DirectoryInfo(@"C:\Temp");
```

```
DirectoryInfo [] arrayDir = dir.GetDirectories();
```

```
foreach (DirectoryInfo elem in arrayDir) {
```

```
    Console.WriteLine(elem);
```

```
} // На экран выводятся все каталоги, находящиеся с папке
```

```
«C:\Temp»
```

### ***GetFiles***

```
public FileInfo[] GetFiles()
```

```
DirectoryInfo dir = new DirectoryInfo(@"C:\Temp");
```

```
FileInfo [] arrayFiles = dir.GetFiles();
foreach (FileInfo elem in arrayFiles) {
    Console.WriteLine(elem);
} // На экран выводятся все файлы, находящиеся в папке
«C:\Temp»
```

### ***MoveTo***

```
public void MoveTo(string destDirName)
DirectoryInfo dir = new DirectoryInfo(@"C:\Temp");
dir.MoveTo(@"D:\Temp"); // Все содержимое папки "C:\Temp" пе-
реместится в папку "D:\Temp" включая вложенные папки и файлы
```

## **2.2. Работа с файлами (*FileInfo*)**

Класс *FileInfo* содержит набор элементов класса унаследованных от классов *FileSystemInfo*, также он дополнен своими собственными элементами. Методы класса *FileInfo* позволяют создавать, копировать, удалять, перемещать, открывать файлы, а также совершать другие операции над файлами.

Для создания объекта данного класса необходимо воспользоваться конструктором:

```
public FileInfo(string fileName),
```

например:

```
FileInfo dir = new FileInfo(@"C:\temp\1.txt");
```

Рассмотрим основные свойства и методы класса *FileInfo* (табл. 3).

**Таблица 3**

### **Основные свойства и методы класса *FileInfo***

Название	Описание
<i>Свойства</i>	
Attributes	Получает или задает атрибуты для текущего файла. Значение этого свойства представляет собой сочетание следующих флагов атрибутов файла: "архивный", "сжатый", "каталог", "скрытый", "автономный", "только для чтения", "системный" и "временный".
CreationTime	Получает или задает время создания текущего файла.
Exists	Получает значение, определяющее наличие файла.
Extension	Получает строку, содержащую расширение файла.
FullName	Получает полный путь к файлу.
Name	Получает имя файла.

Окончание табл. 3

Название	Описание
<i>Методы</i>	
AppendText	Создает объект <i>StreamWriter</i> , который добавляет текст в файл.
CopyTo	Копирует существующий файл в новый файл и запрещает перезапись существующего файла.
Create	Создает файл.
CreateText	Создает <i>StreamWriter</i> , который записывает новый текстовый файл.
MoveTo	Перемещает заданный файл в новое местоположение и разрешает переименование файла.
Open	Открывает файл в заданном режиме.
OpenRead	Создает <i>FileStream</i> , который доступен только для чтения.
OpenText	Создает <i>StreamReader</i> с кодировкой <i>UTF-8</i> , который считывает данные из существующего текстового файла.
OpenWrite	Создает <i>FileStream</i> , доступный только для записи.
Replace	Заменяет содержимое заданного файла на содержимое файла, которое описано в текущем объекте <i>FileInfo</i> , удаляет исходный файл и создает резервную копию замененного файла.

Назначение и приемы работы со свойствами класса *FileInfo* аналогичны назначению и приемам работы со свойствами класса *DirectoryInfo*, поэтому рассматриваться они не будут.

### Примеры использования методов

Почти каждый из рассматриваемых методов имеет множество вариантов реализации (перегрузок). Рассмотрим некоторые из реализаций представленных ранее методов.

#### *AppendText*

```
public StreamWriter AppendText()
FileInfo file = new FileInfo(@"C:\Temp\test.txt");
StreamWriter sw = file.AppendText(); // Создает объект класса
StreamWriter
```



**CopyTo**

```
public FileInfo CopyTo(string destFileName)  

FileInfo file = new FileInfo(@"C:\Temp\test.txt");  

file.CopyTo(@"C:\Temp\Other Folder\test.txt"); // Копирует файл  

C:\Temp\test.txt в новое место C:\Temp\Other Folder\test.txt
```

**Create**

```
public FileStream Create()  

FileInfo file = new FileInfo(@"C:\Temp\NewFile.txt");  

FileStream fs = file.Create(); // Создает новый файл  

C:\Temp\NewFile.txt
```

**CreateText**

```
public StreamWriter CreateText()  

FileInfo file = new FileInfo(@"C:\Temp\test.txt");  

StreamWriter sw = file.CreateText(); // Создается объект типа  

StreamWriter для добавления текстовой информации, если файл не  

существует, то он создается, если существует, то он перезаписы-  

вается
```

**MoveTo**

```
public void MoveTo(string destFileName)  

FileInfo file = new FileInfo(@"C:\Temp\test.txt");  

file.MoveTo(@"C:\Destination Folder\New Name.txt"); // Переме-  

щается файл C:\Temp\test.txt в новое место C:\Destination Folder\New  

Name.txt
```

**Open**

```
public FileStream Open(FileMode mode)  

FileInfo file = new FileInfo(@"C:\Temp\test.txt");  

FileStream fs = file.Open(FileMode.Open); // Открывает файл
```

**OpenRead**

```
public FileStream OpenRead()  

FileInfo file = new FileInfo(@"C:\Temp\test.txt");  

FileStream fs = file.OpenRead(); // Создает объект класса  

FileStream
```

**OpenText**

```
public StreamReader OpenText()  

FileInfo file = new FileInfo(@"C:\Temp\test.txt");  

FileStream fs = file.OpenText(); // Создает объект класса  

StreamReader
```

**OpenWrite**

```
public FileStream OpenWrite()
FileInfo file = new FileInfo(@"C:\Temp\test.txt");
FileStream fs = file.OpenWrite (); // Создает объект класса
FileStream
```

**Replace**

```
public FileInfo Replace(string destinationFileName, string destina-
tionBackupFileName)
FileInfo file = new FileInfo(@"C:\Temp\test.txt");
file.Replace(@"C:\Temp\test1.txt", @"C:\Temp\test.bak"); // Заменя-
ет файл C:\Temp\test.txt файлом C:\Temp\test1.txt, при этом создает
копию файла C:\Temp\test.txt с именем C:\Temp\test.bak
```

**2.3. Работа с классом *FileStream***

Класс *FileStream* поддерживает синхронные и асинхронные операции чтения и записи. Для создания объекта данного класса необходимо воспользоваться одним из множества доступных конструкторов, например:

```
public FileStream(string path, FileMode mode, FileAccess access)
Создадим объект класса FileStream для описанного конструктора:
FileStream fs = new FileStream(@"C:\Temp\test.txt", File-
Mode.Open, FileAccess.Read);
```

Рассмотрим основные методы для работы с классом *FileStream* (табл. 4).

**Таблица 4****Основные методы для работы с классом *FileStream***

Название	Описание
Close	Закрывает текущий поток и отключает все ресурсы (например, сокеты и файловые дескрипторы), связанные с текущим потоком.
Flush	Очищает буферы для этого потока и вызывает запись всех буферизованных данных в файл.
Read	Выполняет чтение блока байтов из потока и запись данных в заданный буфер.
Seek	Устанавливает текущее положение этого потока на заданное значение.
Write	Записывает блок байтов в файловый поток.

## Примеры использования методов

Рассмотрим некоторые из реализаций представленных ранее методов.

### ***Close***

```
public virtual void Close()
FileStream fs = new FileStream(@"C:\Temp\test.txt", File-
Mode.Open, FileAccess.Read);
... // Операции с файлом
fs.Close(); // Закрывает текущий поток
```

### ***Flush***

```
public override void Flush()
FileStream fs = new FileStream(@"C:\Temp\test.txt", File-
Mode.OpenOrCreate, FileAccess.ReadWrite);
... // Операции с файлом
fs.Flush(); // Записывает данные файлы и очищает буферы те-
кущего потока
```

### ***Read***

```
public override int Read(byte[] array, int offset, int count)
FileStream fs = new FileStream(@"C:\Temp\test.txt", File-
Mode.Open, FileAccess.Read);
byte[] array = new byte[fs.Length];
int n = fs.Read(array, 0, 5); // Записывает в массив array 5 байт,
считанных с файла C:\Temp\test.txt начиная с нулевой позиции в мас-
сиве array. В переменную n будет записано количество считанных
байт
```

### ***Seek***

```
public override long Seek(long offset, SeekOrigin origin)
FileStream fs = new FileStream(@"C:\Temp\test.txt", File-
Mode.Open, FileAccess.Read);
fs.Seek(0, SeekOrigin.Begin); // Устанавливаем текущую позицию
в файле в начало файла
```

### ***Write***

```
public override void Write(byte[] array, int offset, int count)
FileStream fs = new FileStream(@"C:\Temp\test.txt", File-
Mode.OpenOrCreate, FileAccess.ReadWrite);
byte[] array = new byte[3] { 1, 2, 3};
```

`fs.Write(array, 0, 3);` // Записывает содержание массива *array* в буфер. Информация, содержащаяся в буфере, запишется в файл *C:\Temp\test.txt* в случае вызова одного из методов: *Flush*, *Close*.

## 2.4. Работа с классом *StreamReader*

Класс *StreamReader* применяется для чтения текстовых файлов. Для создания объекта данного класса необходимо воспользоваться одним из множества доступных конструкторов, например:

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

Создадим объект класса *StreamReader* для описанного конструктора:

```
StreamReader sr = new StreamReader(@"C:\Temp\test.txt");
```

Рассмотрим основные методы для работы с классом *StreamReader* (табл. 5).

**Таблица 5**

**Основные методы для работы с классом *StreamReader***

Название	Описание
Close	Закрывает объект <i>StreamReader</i> и основной поток и освобождает все системные ресурсы, связанные с устройством чтения.
Peek	Возвращает следующий доступный символ, но не использует его. Если доступных для чтения символов нет или поток не поддерживает поиск, то возвращает значение -1.
Read	Выполняет чтение следующего символа из входного потока и перемещает положение символа на одну позицию вперед.
ReadBlock	Считывает указанное максимальное количество символов из текущего потока и записывает данные в буфер, начиная с заданного индекса.
ReadLine	Выполняет чтение строки символов из текущего потока и возвращает данные в виде строки.
ReadToEnd	Считывает все символы, начиная с текущей позиции до конца потока.

## Примеры использования методов

Рассмотрим некоторые из реализаций представленных ранее методов. Для представленных ниже примеров проведена предварительная инициализация объекта *sr* класса *StreamReader*:

```
StreamReader sr = new StreamReader(@"C:\Temp\test.txt");
```

### ***Close***

```
public override void Close()
```

```
... // Операции с файлом
```

```
sr.Close(); // Закрывает объект и основной поток
```

### ***Peek***

```
public override int Peek()
```

```
int code = sr.Peek(); // Читает код текущего символа, при этом  
указатель в потоке не перемещается
```

### ***Read***

```
public override int Read()
```

```
char ch = (char) sr.Read(); // Считываем из потока текущий код  
символа и преобразовываем его в символьный тип
```

### ***ReadBlock***

```
public override int ReadBlock(char[] buffer, int index, int count)
```

```
char [] array = new char[3];
```

```
int n = sr.ReadBlock(array, 0, 3); // Считываем 3 символа в массив  
array
```

### ***ReadLine***

```
public override string ReadLine()
```

```
string s = sr.ReadLine(); // Считываем строку с текущей позиции  
и записываем в переменную s
```

### ***ReadToEnd***

```
public override string ReadToEnd()
```

```
string s = sr.ReadToEnd(); // Считываем строку с текущей пози-  
ции до последней в файле C:\Temp\test.txt и записываем в переменную  
s
```

## 2.5. Работа с классом *StreamWriter*

Класс *StreamWriter* применяется для записи информации в текстовый файл. Для создания объекта данного класса необходимо вос-

пользоваться одним из множества доступных конструкторов, например:

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

Создадим объект класса *StreamWriter* для описанного конструктора:

```
StreamWriter sw = new StreamWriter(@"C:\Temp\test.txt");
```

Рассмотрим основные методы для работы с классом *StreamWriter* (табл. 6).

**Таблица 6**

**Основные методы для работы с классом *StreamWriter***

Название	Описание
Close	Закрывает текущий объект <i>StreamWriter</i> и базовый поток.
Flush	Очищает все буферы для текущего средства записи и вызывает запись всех данных буфера в основной поток.
Write	Записывает в текстовую строку или поток текстовое представление переданного значения
WriteLine	Записывает в текстовую строку или поток текстовое представление переданного значения, за которой следует признак конца строки.

**Синтаксис методов класса *StreamWriter***

Примеры использования представленных методов класса *StreamWriter* рассматриваться не будут, т.к. они схожи с методами класса *StreamReader*. Рассмотрим только синтаксис описания некоторых реализаций методов.

```
public override void Close()
```

```
public override void Flush()
```

```
public override void Write(char value)
```

```
public virtual void Write(double value)
```

```
public virtual void Write(int value)
```

```
public override void Write(string value)
```

```
public virtual void WriteLine(char value)
```

```
public virtual void WriteLine(double value)
```

```
public virtual void WriteLine(int value)
```

```
public virtual void WriteLine(string value)
```

## 2.6. Основные этапы работы с файлами

1. Подключение пространства имен System.IO.

```
using System.IO;
```

2. Инициализация необходимых объектов для работы с файлом, например:

```
FileStream fs = new FileStream(@"C:\Temp\test.txt", FileMode.OpenOrCreate, FileAccess.ReadWrite);
```

```
StreamReader sr = new StreamReader(@"C:\Temp\test.txt");
```

```
StreamWriter sw = new StreamWriter(@"C:\Temp\test.txt");
```

```
...
```

3. Операции над файлом, например что-то считываем или что-то записываем в файл.

4. Закрытие открытых потоков по работе с файлом, например:

```
fs.Close();
```

```
sr.Close();
```

```
sw.Close();
```

```
...
```

## 2.7. Примеры на работу с файлами

**Задача 1.** Необходимо в текстовый файл записать строку «Пример работы с текстовым файлом».

```
using System;
using System.IO;
namespace ExampleWorkWithTextFile
{
    class Program
    {
        static void Main()
        {
            StreamWriter sw = new StreamWriter(@"C:\Temp\test.txt");
            sw.WriteLine("Пример работы с текстовым файлом");
            sw.Close();
        }
    }
}
```

**Задача 2.** Необходимо в текстовом файле подсчитать количество символов, относящихся к цифрам.

```

using System;
using System.Collections.Generic;
using System.IO;
namespace CountDigits
{
    class Program
    {
        static void Main()
        {
            StreamReader sr = new StreamReader(
er(@"C:\Temp\test.txt");
            char ch;
            List<char> digits = new List<char>() {'0',
'1', '2', '3', '4', '5', '6', '7', '8', '9'};
            int count = 0;
            while (sr.Peek() >= 0) {
                ch = (char)sr.Read();
                if (digits.Contains(ch)) count++;
            }
            sr.Close();
            Console.WriteLine("{0}", count);
            Console.Read();
        }
    }
}

```

**Задача 3.** Требуется перевернуть содержимое текстового файла.

```

using System;
using System.Collections.Generic;
using System.IO;
namespace ReverseFile
{
    class Program
    {
        static void Main()
        {
            StreamReader sr = new StreamReader(
er(@"C:\Temp\test.txt");
            char ch;
            List<char> digits = new List<char>() {'0',
'1', '2', '3', '4', '5', '6', '7', '8', '9'};

```



```

        int count = 0;
        while (sr.Peek() >= 0) {
            ch = (char)sr.Read();
            if (digits.Contains(ch)) count++;
        }
        sr.Close();
        Console.WriteLine("Количество цифр = {0}",
count);
        Console.Read();
    }
}

```

**Задача 4.** Необходимо из файла удалить все символы, совпадающие с символом, введенным с клавиатуры.

```

using System;
using System.IO;
namespace DeleteChars
{
    class Program
    {
        static void Main()
        {
            StreamReader sr = new StreamReader(
@"C:\Temp\test.txt");
            StreamWriter sw = new StreamWriter(
@"C:\Temp\test.tmp");
            FileInfo fi = new FileInfo(
@"C:\Temp\test.txt");
            FileInfo fiTemp = new FileInfo(
@"C:\Temp\test.tmp");
            char ch, c;
            c = (char)Console.Read();
            while (sr.Peek() >= 0) {
                ch = (char)sr.Read();
                if (c != ch) sw.Write(ch);
            }
            sr.Close();
            sw.Close();
            fi.Delete();
            fiTemp.MoveTo(@"C:\Temp\test.txt");
            Console.Read();
        }
    }
}

```

```

    }
}

```

**Задача 5.** Требуется выполнить запись и чтение структурированной информации из файла. Структура одного элемента должна описывать основные характеристики студента.

**Решение.**

Для удобства записи характеристик студента представим их в виде структуры (struct), при этом для записи в файл объект структуры необходимо сериализовать.

*Сериализация* представляет собой процесс преобразования объекта в поток байтов с целью сохранения его в памяти, в базе данных или в файле. Ее основное назначение – сохранить состояние объекта для того, чтобы иметь возможность воссоздать его при необходимости. Обратный процесс называется *десериализацией*.

```

using System;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
namespace StructFile
{
    class Program
    {
        // Установим атрибут сообщаящий компилятору воз-
        можность сериализовать объекты структуры Student
        [SerializableAttribute]

        // Структура, описывающая характеристики студента
        struct Student {
            public string name;
            public string surname;
            public string groupName;
            public DateTime birthday;
            public bool male;
        }

        //////////////////////////////////////
        static void Main()
        {
            BinaryFormatter bf = new BinaryFormatter();

```

```

        FileStream fs = new
FileStream(@"C:\Temp\test.txt", FileMode.OpenOrCreate, File-
Access.ReadWrite);
        Student student;

        // Заполняем данными объект student
        student.name = "Петя";
        student.surname = "Петров";
        student.groupName = "12-ИБАС";
        DateTime date = new DateTime(2012, 12, 18);
        student.birthday = date;
        student.male = true;

        // Преобразуем объект student в поток байтов и
запишем его в поток fs (сериализация)
        bf.Serialize(fs, student);

        // Запишем состояние потока в файл
        fs.Flush();

        // Установим текущую позицию для чтения на
начало файла
        fs.Seek(0, SeekOrigin.Begin);

        // Преобразуем поток байтов считанных из файла
в объект student (десериализация)
        student = (Student)bf.Deserialize(fs);

        // Выведем на экран характеристики студента
        Console.WriteLine("Характеристики студента:");
        Console.WriteLine("\tИмя: {0}", student.name);
        Console.WriteLine("\tФамилия: {0}", stu-
dent.surname);
        Console.WriteLine("\tГруппа: {0}", stu-
dent.groupName);
        Console.WriteLine("\tДата рождения: {0}", stu-
dent.birthday.ToString("dd.ММ.yyyy"));
        if (student.male) Console.WriteLine("\tПол:
Мужской");
        else Console.WriteLine("\tПол: Женский");

        // Закроем поток
        fs.Close();
        Console.Read();

```

```

    }
}

```

### 3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Для выполнения лабораторной работы необходимо предварительно ознакомиться с теоретической частью, в случае необходимости можно использовать дополнительный теоретический материал.

После ознакомления с теорией необходимо выполнить два задания:

- 1) работа с текстовым файлом;
- 2) работа с типизированным файлом.

При выполнении заданий необходимо соблюдать требования сформулированные ниже.

#### Общие требования к программе

1. Текст программы представляется в электронном виде и должен включать постановку задачи, сведения об авторе и подробные комментарии.

2. Названия переменных и констант должны быть логически обоснованы и давать понятие о том, какая информация в них представлена, при создании метода его имя должно отражать его функциональное назначение.

3. Программа должна запрашивать входные данные и выводить итоговый результат с пояснениями.

### 4. СПИСОК ЗАДАНИЙ

Ниже представлен список заданий на работу с текстовым и типизированным файлами.

#### 4.1. Работа с текстовыми файлами

1. Каждая строка текстового файла имеет следующую структуру: фамилия студента; факультет; группа; средний балл. Поля разделены одним или несколькими пробелами. Внутри поля пробелы не допускаются. Требуется организовать текстовый файл с полями: факультет; группа; средний балл по группе. Определить группы с наибольшим и наименьшим средним баллом.
2. Имеется файл с текстом на русском языке. Дать варианты переноса всех слов. Перенос возможен по следующим правилам:

- в конце слова должно оставаться не менее двух символов;
  - невозможен перенос перед буквами 'ь' и 'Ъ';
  - слово должно иметь не менее двух слогов;
  - в оставшейся и переносимой частях слова должны быть гласные буквы.
3. Разработать программу, обеспечивающую замену в текстовом файле строчных букв прописными, а прописных строчными. Имя файла должно вводиться с клавиатуры.
  4. Текстовый файл содержит последовательность целых чисел, разделенных произвольным количеством других символов. Числа могут переходить со строки на строку. Требуется найти сумму чисел.
  5. Имеется текстовый файл с некоторым описанием. Переносы слов разрешены. Выдать информацию о том, сколько раз встречается в тексте каждое слово.
  6. Строки текстового файла содержат фамилии студентов в произвольном порядке. Определить порядковый номер символа, с которого располагается первая по алфавиту фамилия. Заменить данную фамилию в исходном файле символами '\*', не переписывая этого файла.
  7. В текстовом файле записан отдельный абзац. Переноса слов нет. Необходимо выравнить правые границы строк по заданной позиции за счет вставки дополнительных пробелов между словами. Число строк в исходном и конечном файлах может отличаться.
  8. Написать программу шифрации-дешифрации текстового файла.
  9. Дан текстовый файл и ключевое слово для поиска. Выдать на экран три строки файла, начиная с первой, включающей заданный образец. Выделить цветом ключевое слово поиска.
  10. Имеется текст документа, состоящий из нескольких параграфов. Каждый параграф начинается с заголовка и отделен пустой строкой от предыдущего параграфа. Текст разделен на страницы. Номер страницы проставлен в ее начале и выделен с обеих сторон знаком '-'. Сформировать файл с оглавлением документа.
  11. Дан текстовый файл. Выполнить его шифрацию по следующим правилам:

- шифровать текст блоками по 32 байта;
- коды символов 32-242 зашифровать по формулам:

$$a[n] = ((243 - n + m) \bmod 211 + 32;$$

$$m = (3 \cdot k - 1) \bmod 99,$$

где  $n$  – исходный код символа;  $a[n]$  – зашифрованный код;  $k$  – порядковый номер блока.

12. Дан текстовый файл и два слова. Переносов слов нет. Получить новый файл, в котором все вхождения первого слова заменены на второе слово.
13. В текстовом файле записан отдельный абзац. Некоторые слова перенесены со строки на следующую строку. Знак переноса “-”. Создать новый файл с заданным текстом, в котором правые границы строк выровнены по заданной позиции и нет переноса слов.
14. В некоторых строках текстового файла имеются выражения, состоящие из двух целых чисел, разделенных знаком арифметической операции ( '+', '-', '\*', '/' ). В строке перед выражением и после него могут находиться произвольные символы. Требуется выделить строку, в которой значение выражения максимально.
15. Имеется два текстовых файла. В первом из них содержится некоторое описание. Переносы слов допускаются. Второй файл содержит список слов, не подлежащих разглашению. Требуется переписать первый файл, заменив каждое из подобных слов точками.
16. В файле записан текст стихотворения. Форматировать текст так, чтобы каждое четверостишие следовало с одной и той же позиции, начиналось с прописной буквы и было сдвинуто относительно предыдущего на пять позиций вправо или влево поочередно.
17. Некоторый текст состоит из нескольких частей, записанных в отдельных файлах. Имена этих файлов и общий заголовок текста указаны в отдельном файле. Создать файл с полным текстом. Заголовок должен содержаться в центре первой строки.
18. Имеется текстовый файл с некоторым описанием. Все предложения заканчиваются точкой. Требуется проверить, является ли первая буква каждого предложения прописной. Исправить обнаруженные ошибки.

19. Задан текстовый файл. Создать новый файл, в котором строки будут следовать в обратном порядке.
20. Алфавит некоторого языка программирования включает латинские буквы, цифры, знаки пунктуации (',', '!', '?') и арифметические операции ( '+', '-', '\*', '/' ). Программа, написанная на этом языке, содержится в текстовом файле. Проверить допустимость текста программы. Выдать сообщение с указанием места ошибочных символов.
21. Имеется текстовый файл, состоящий из нескольких разделов и подразделов. Разделы нумеруются одной цифрой (например, 4), подразделы несколькими цифрами с разделителями в виде точки. Сформировать файл с постраничной печатью по  $n$  строк на странице. Разделы начинаются с новой страницы. Заглавия подразделов не печатаются на странице отдельно от текста подраздела.
22. Имеется некоторый файл с текстом и файл, содержащий отдельные слова (словарь). Разработать программу, проверяющую правильность написания отдельных слов, а в случае необходимости добавляющие отдельные слова в словарь.
23. Каждая строка текстового файла имеет структуру: фамилия; год рождения; специальность. Поля разделены одним или несколькими пробелами. Внутри поля пробелы не допускаются. Требуется организовать файл с отсортированными по алфавиту фамилиями и определить количество людей старше 50 лет.
24. Дан текстовый файл и одно слово. Найти в текстовом файле слова, отличающиеся от заданного на одну букву и заменить на заданное. Выделить их подсветкой.
25. Дан текстовый файл, в котором возможны переносы слов со строки на строку. Подсчитать общее число слов. Имя файла задать в командной строке.
26. Дан текстовый файл. Заменить все гласные буквы в тексте на соответствующий порядковый номер в алфавите.
27. Дан текстовый файл. Удалить из файла все цифры.
28. Заполнить файл случайными числами, русскими буквами и английскими. Взять случайный элемент файла. И подсчитать теоретическую и практическую вероятность того, что этим элементом окажется цифра.

29. Разработать программу перекодировки текстового файла из основной системы кодировки в альтернативную и обратно.
30. Преобразовать файл, используя транслитерацию, информация которого задана на русском языке.

#### **4.1. Работа с типизированными файлами**

1. Выполнить структурированную запись и чтение информации о точках из файла.
2. Выполнить структурированную запись и чтение информации о линиях из файла.
3. Выполнить структурированную запись и чтение информации о прямоугольниках из файла.
4. Выполнить структурированную запись и чтение информации об эллипсах из файла.
5. Выполнить структурированную запись и чтение информации об окружностях из файла.
6. Создать телефонный справочник, в котором содержится информация об абонентах: имя, отчество, фамилия, адрес, телефон.
7. Реализовать вывод \*.log файла, в котором содержится информация о файле: имя файла с расширением, размер файла, дата, атрибут.
8. Обеспечить вывод информации о библиотечной литературе: название книги, имя автора, год издания, количество страниц.
9. Реализовать вывод о товарах, содержащихся на складе: наименование товара, количество его на складе, стоимость единицы товара, единицы измерения товара.
10. Выполнить структурированную запись и чтение информации о многоугольниках из файла.
11. Обеспечить структурированную запись и чтение информации о студентах: имя, отчество, фамилия, специальность, возраст.
12. Реализовать список продукции видеопроката: характеристика, название фильма, присутствие в прокате (логич. тип), стоимость проката за одни сутки.



13. Реализовать вывод информации, содержащейся в файле, характеризующей сбои в работе приложения: название ошибки (или номер ошибки), дата и время возникновения ошибки.
14. Обеспечить выполнение поиска автора по произведению и наоборот.
15. Реализовать возможность редактирования информации библиотечной литературы. Смотри 8-е задание.
16. Создать каталог машин, в котором имеется информация о марке машины, годе выпуска, цвете, цене машины.
17. Реализовать каталог мобильных телефонов, в котором имеется информация о фирме изготовителя, модели, цене, серийном номере.
18. Имеется файл, где сохранены результаты игры: имя игрока, количество очков. Необходимо вывести результаты в виде таблицы. Данные должны быть отсортированы по убыванию.
19. Реализовать сортировку данных о работниках фирмы по возрасту. О работниках известна следующая информация: Ф.И.О., дата рождения, дата принятия на работу, специальность/должность.
20. Организовать поиск по таблице с заданными параметрами. Смотри 19-е задание.
21. Отобразить на экране список людей, достигших пенсионного возраста, и при желании удалить их из файла. Смотри 19-е задание.
22. Имеется два файла с информацией о работниках фирмы. Смотри 19-е задание. Необходимо объединить два файла в один с учетом того, что поля в файлах не идентичны.
23. Вывести список сотрудников, работающих не по специальности. Смотри 19-е задание.
24. Реализовать поиск мобильного телефона по заданному серийному номеру. Смотри 17-е задание.
25. Реализовать поиск по ценовому пределу мобильного телефона. Запрос выглядит следующим образом: подыскать телефон не выше 3000руб. Выводится список всех телефонов, соответствующих заданному условию. Смотри 17-е задание.

26. Реализовать расписание вылетов самолетов, в котором имеется информация о направлении, дате и времени вылета, продолжительности полета, количестве свободных мест.
27. Выполнить поиск самолетов, на которых можно попасть в заданный населенный пункт в определенную дату. Смотри 26-е задание.
28. Определить количество свободных мест в самолетах, вылетающих в определенный день. Смотри 26-е задание.
29. Создать файл, компонентами которого являются записи, содержащие сведения о путевках в туристическом бюро: место и продолжительность отдыха, дата отъезда, стоимость.
30. Определить места возможного отдыха в июле, продолжительностью не более 20 дней. Смотри 29-е задание.

## 5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назовите основные этапы работы с файлами.
2. Какие вы знаете классы, предназначенные для работы с файлами и каталогами?
3. Назовите основные методы следующих классов: *DirectoryInfo*, *FileInfo*, *FileStream*, *StreamReader*, *StreamWriter*, приведите примеры их использования.
4. Каким образом создать объекты классов: *DirectoryInfo*, *FileInfo*, *FileStream*, *StreamReader*, *StreamWriter*?
5. Каким образом организовать чтение текстового файла?
6. Как выполнить структурированную запись и чтение из файла?
7. Для чего используется сериализация и десериализация?
8. Для чего предназначены классы: *StreamReader*, *StreamWriter*?
9. Как узнать имеется ли требуемый файл или каталог?

## 6. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

### Основная

1. Павловская Т. А. С#. Программирование на языке высокого уровня. – Изд.: Питер, 2009. – 432с.
2. Эндрю Троелсен. Язык программирования С# 2010 и платформа .NET 4. – Изд.: Вильямс, 2011. – 1392с.

3. Кристиан Нейгел, Билл Ивсен, Джей Глинн, Карли Уотсон, Морган Скиннер. С# 4.0 и платформа .NET 4 для профессионалов. – Изд.: Питер, 2011. – 1440с.

***Дополнительная***

4. Тыртышников Е.Е. Методы численного анализа. Учебное пособие. – Изд.: МГУ, 2006. – 281с.
5. Джесс Либерти. Программирование на С#. – Изд.: КноРус, 2003. – 688с.
6. Харви Дейтел. С# в подлиннике. Наиболее полное руководство. – Изд.: БХВ-Петербург, 2006. – 1056с.

Языки программирования. Работа с файлами: методические указания к выполнению лабораторной работы для студентов очной формы обучения специальностей 090303 – «Информационная безопасность автоматизированных систем», 090900 – «Информационная безопасность». – Брянск: БГТУ, 2013. – 28 с.

ЮРИЙ АЛЕКСЕЕВИЧ ЛЕОНОВ  
ЕВГЕНИЙ АЛЕКСЕЕВИЧ ЛЕОНОВ

Научный редактор: Ю.М. Казаков  
Редактор издательства: Л.И. Афонина  
Компьютерный набор: Ю.А. Леонов

Темплан 2013 г., п.

---

Подписано в печать    Формат 60х84 1/16.    Бумага офсетная.  
Офсетная печать.

Усл. печ. л. 1,63    Уч. – изд. л. 1,63    Тираж 20 экз.    Заказ    Бесплатно

---

Издательство брянского государственного технического университета,  
241035, Брянск, бульвар 50-летия Октября, 7, БГТУ. 58-82-49  
Лаборатория оперативной полиграфии БГТУ, ул. Харьковская, 9