Практикум 3. Составные типы данных.

1. Класс Array

1.1. Теоретические сведения

Во многих приложениях требуется создавать группы связанных объектов и управлять этими группами. Существует два способа группировки объектов: создать массив объектов и создать коллекцию.

Массив — это структура данных, содержащая несколько переменных одного типа. Массивы объявляются со следующим типом.

Массив – набор *однотипных* компонентов (элементов), расположенных в памяти непосредственно друг за другом, доступ к которым осуществляется по *индексу* (индексам). Массив является структурой с *произвольным доступом*.

Размерность массива – *количество индексов*, необходимое для однозначного доступа к элементу массива, *задается в момент объявления массива*.

```
Тип_данных [] имя_массива = new Тип_данных [Размерность_массива];
```

Ключевое слово new отвечает за вычисление числа байтов, необходимых для заданного объекта, и выделение достаточного объема управляемой динамической памяти. В данном случае требуется разместить объект типа массив. Следует понимать, что объектные переменные C# на самом деле являются *ссылками* на объект в памяти, а не фактическими объектами и ссылаются на уникальный объект, размещенный в динамической памяти.

Примеры:

Можно объявить массив из пяти целых чисел:

```
int[] array = new int[5];
```

Массив содержит элементы с `array[0]` по `array[4]`. Оператор `new` служит для создания массива и инициализации элементов массива со значениями по умолчанию. В данном примере элементы массива инициализируются значением 0.

Массив, в котором хранятся строковые элементы:

```
string[] stringArray = new string[6];
```

Массив можно инициализировать при объявлении. В этом случае указывать размерность не обязательно, поскольку она уже предоставлена по числу элементов в списке инициализации:

```
int[] array1 = new int[] { 1, 3, 5, 7, 9 };
```

Строковый массив можно инициализировать таким же образом:

```
string[] weekDays = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
```

При инициализации массива при объявлении можно использовать следующие сочетания клавиш:

```
int[] array2 = { 1, 3, 5, 7, 9 };
string[] weekDays2 = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
```

Можно объявить переменную массива без инициализации, но при присвоении массива этой переменной нужно использовать оператор new. Пример.

C#

```
int[] array3;
array3 = new int[] { 1, 3, 5, 7, 9 };    // OK
//array3 = {1, 3, 5, 7, 9};    // Error
```

Класс System.Array

В платформе .NET Framework массивы реализуются как экземпляры класса Array.

Благодаря наследованию от класса *System.Array*, для массивов определены встроенные методы копирования, поиска, обращения, сортировки. Массивы можно рассматривать также как коллекции и использовать цикл *foreach* для перебора его элементов.

Ниже приведены основные свойства и методы класса *System.Array:*

Таблица 1. Основные свойства и методы класса *System.Array*

Свойство	Описание
IsFixedSize	Возвращает *true*, если массив является статическим
IsReadOnly	Для всех массивов возвращает значение *false*
IsSynchronized	Возвращает *true* или *false*, в зависимости от того, установлена ли синхронизация доступа для массива
Length	Число элементов в массиве
Rank	Размерность массива

Метод	Описание
BinarySearch	Бинарный поиск элементов
Clear	Выполняет инициализацию элементов. Числовые элементы становятся нулевыми, а строковые принимают значение *Null*
Copy(Array, Array, Int32)	Копирует диапазон элементов из массива Array, начиная с первого элемента, и вставляет его в другой массив Array, также начиная с первого элемента. Длина задается как 32-разрядное целое число.
Copy	Копирование части или всего массива в другой массив
CopyTo	Копируются все элементы одномерного массива в другой одномерный массив, начиная с заданного индекса
IndexOf	Индекс первого вхождения образца в массив
LastIndexOf	Индекс последнего вхождения образца в массив
Reverse	«Переворачивание» одномерного массива в обратном порядке
Sort	Сортировка массива

Более подробно о методах класса Array можно посмотреть здесь:
http://msdn.microsoft.com/ru-ru/library/system.array%28v=vs.110%29.aspx

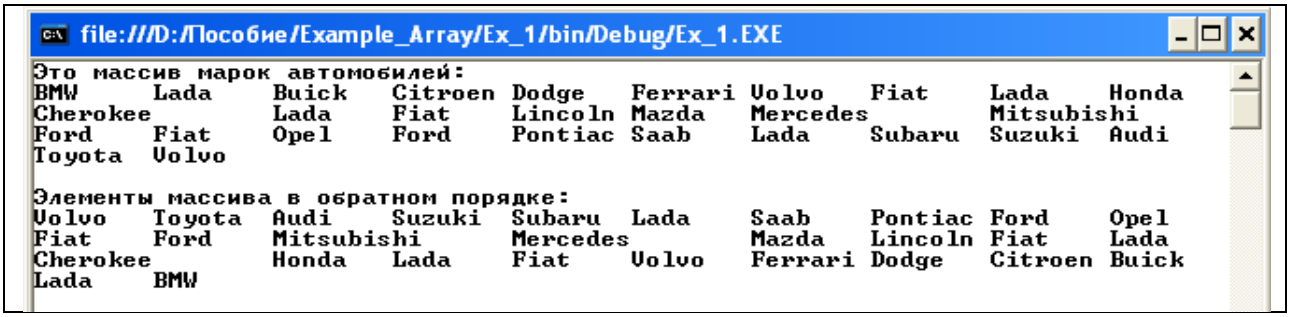
1.2. Пример. Свойства и методы класса Array.

Следующий пример демонстрирует использование некоторых из свойств и методов класса *Array*:

Листинг кода 3.1.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ArrayDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            // Массив символьных строк
            string[] Brands = new string[]
               {"BMW",
                "Lada",
                "Buick",
                "Citroen",
                "Dodge",
                "Ferrari",
                "Volvo",
                "Fiat",
                "Lada",
                "Honda",
                "Cherokee",
                "Lada",
                "Fiat",
                "Lincoln",
                "Mazda",
                "Mercedes",
                "Mitsubishi",
                "Ford",
                "Fiat",
                "Opel",
                "Ford",
                "Pontiac",
                "Saab",
                "Lada",
                "Subaru",
                "Suzuki",
                "Audi",
                "Toyota",
                "Volvo"};
            // Вывод марок автомобилей в соответствии с порядком элементов в массиве
            Console.WriteLine("Это массив марок автомобилей:");

            for (int i = 0; i < Brands.Length; i++)
                Console.Write(Brands[i] + "\t");

            Console.Write("\n\n");

            // Сортировка элементов в обратном порядке
            Array.Reverse(Brands);

            // Вывод автомарок
            Console.WriteLine("Элементы массива в обратном порядке:");
            for (int i = 0; i < Brands.Length; i++)
                Console.Write(Brands[i] + "\t");

            Console.Write("\n\n");

            Console.ReadKey();
        }
    }
}
```

Результат работы программы:



Задание 3.1.

1. Реализуйте код примера 1 и убедитесь в его работоспособности.
2. Добавьте строки кода, определяющие количество элементов в массиве
3. Добавьте строки кода, позволяющие отсортировать (упорядочить элементы массива а) по алфавиту от A до Z б) от Z до A

1.3. Пример. Визуальные компоненты для работы с массивами в формах

В приложениях Windows Forms для отображения элементов удобно использовать элемент управления ListBox.

Для добавления элементов массива в коллекцию Items для ListBox можно использовать вследующий код:

```
for (int i=0; i<B.Length;i++)
            listBox1.Items.Add(B [i]);
```

Задание 3.2.

1. Добавьте к решению новый проект WindowsForms. Разместите на нем элементы управления ListBox и Menu Strip.
2. Пунктами меню должны быть следующие действия: Вывести (исходный массив), Обратить (перевернуть в обратном порядке), Отсортировать.

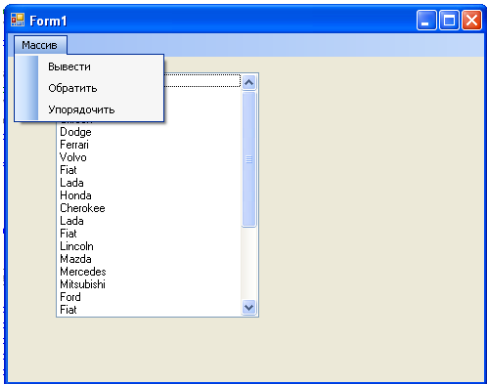


Рисунок 3.1. Элементы управления для работы с массивами

Задание 3.3

1. Добавьте строки в код проекта Задание 3.1., чтобы продемонстрировать работу методов LastIndexOf, IndexOf.

Подсказка: например:

```
Console.WriteLine("В массиве Лада стоит на {0}  месте ", Array.IndexOf(Brands,"Lada")+1);
```

2. Доработайте проект таким образом, чтобы можно было вывести сообщение вида «Указанная марка есть в списке» или «Указанной марки нет в списке» для любого введенного с клавиатуры значения.

1.4. Сравнение быстродействия работы встроенных методов класса и методов, созданных программистом.

Алгоритм упорядочивания массива (сортировки) может быть написан программистом самостоятельно. Существует несколько легко доступных для понимания способов сортировки: сортировка вставками, сортировка «пузырьковая», сортировка выбором и т.д. Они подробно описаны практически во всех учебниках программирования и их алгоритмы не зависят от языка реализации.

У класса Array есть метод Sort, алгоритм которого оптимизирован и выверен математически. С этой точки зрения интересно посмотреть на разницу в быстродействии.

Листинг примера сравнения приведен ниже:

Задание 3.4.

Проверьте на практическом примере разницу в быстродействии самописных методов сортировки и встроенного метода класса Array. Для получения результата следует использовать массивы достаточно большого размера.

Листинг кода 3.2.

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

namespace Сортировка_пузырьком
{
    class Program
    {
        static void Main(string[] args)
        {
            Random rnd = new Random();
            int[] mas = new int[20000];
            //for (int i = 0; i < 20; i++)
            //{
            //    mas[i] = rnd.Next(1, 100);
            //    Console.Write("{0} \t", mas[i]);
            //}
            // Сортировка пузырьком
            DateTime start, finish;
            double res = 0;
            start = DateTime.Now;
            for (int i = 0; i < mas.Length; i++)
            {
                for (int j = 0; j < mas.Length-1; j++)
```

```
                {
                    if (mas[j] <= mas[j + 1])
                    {
                        int d = mas[j];
                        mas[j] = mas[j + 1];
                        mas[j + 1] = d;
                    }
                 }
            }
            finish = DateTime.Now;

            res = (finish - start).Milliseconds;
            Console.WriteLine("Сортировка пузырьком");
                //foreach(int elem in mas) Console.Write("{0} \t", elem);
                //Console.WriteLine();

                Console.WriteLine("t1={0} ", res);

                DateTime start1, finish1;
                double res1 = 0;
                start1 = DateTime.Now;
                Array.Sort(mas);
                finish1 = DateTime.Now;
                res1 = (finish1 - start1).Milliseconds;
                Console.WriteLine("t2={0} ", res1);
            Console.ReadKey();


        }
    }
}
```

2. Структуры

2.1 Понятие

Кроме базовых элементарных типов данных и перечислений в C# имеется и составной тип данных, который называется **структурой**. Структуры могут содержать в себе обычные переменные и методы.

Для примера создадим структуру Book, в которой будут храниться переменные для названия, автора и года издания книги. Кроме того, структура будет содержать метод для вывода информации о книге на консоль:

```
struct Book
{
    public string name;
    public string author;
    public int year;

    public void Info()
    {
        Console.WriteLine("Книга '{0}' (автор {1}) была издана в {2} году", name,
author, year);
    }
}
```

Чтобы можно было использовать переменные и методы структуры из любого места программы мы ставим перед переменными и методом модификатор доступа `public`

Пример

```
using System;
```

```
namespace Structures
{
    class Program
    {
        static void Main(string[] args)
        {
            Book book;
            book.name = "Война и мир";
            book.author = "Л. Н. Толстой";
            book.year = 1869;

            //Выведем информацию о книге book на экран
            book.Info();

            Console.ReadLine();
        }
    }

    struct Book
    {
        public string name;
        public string author;
        public int year;

        public void Info()
        {
            Console.WriteLine("Книга '{0}' (автор {1}) была издана в {2} году",
name, author, year);
        }
    }
}
```

2.2 Массив структур

По сути структура Book представляет новый тип данных. Мы также можем использовать массив структур:

```
Book[] books=new Book[3];
books[0].name = "Война и мир";
books[0].author = "Л. Н. Толстой";
books[0].year = 1869;

books[1].name = "Преступление и наказание";
books[1].author = "Ф. М. Достоевский";
books[1].year = 1866;

books[2].name = "Отцы и дети";
books[2].author = "И. С. Тургенев";
books[2].year = 1862;

foreach (Book b in books)
{
    b.Info();
}
```

К массивам структур нельзя применить стандартный метод SORTиз класса Array, т.к. нельзя отсортировать массив, состоящий из структур, без указания критерия сортировки. Например, нельзя упорядочить данные о студентах, если не указать, по какому критерию идет упорядочивание: по алфавитному принципу по фамилиям, или по среднему баллу оценок, или по группам. Массивы структур сортируют по какому либо из полей. Язык C# имеет очень мощное и элегантное средство для решения подобных задач – язык запросов LINQ.

2.3. Примеры использования языка запросов LINQ

Язык запросов очень обширен, но достаточно понятен, при этом он избавляет вас от придумывания многоциклических алгоритмов.

Рассмотрим несколько простейших примеров.

1 способ: Использование запросов Linq

Пример 1. Из строкового массива выбрать только те элементы, которые начинаются с определенной буквы.

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

namespace Начальная_буква
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] names = { "Ann", "Peter", "Bob", "Joile", "Ljuse", "Olga",
                               "Boris", "Helga", "Bill","Zarra" };

            var queryResults =
                from n in names
                where n.StartsWith("B")
                select n;
            Console.WriteLine("С буквы B начинаются имена: ");
            foreach (var k in queryResults)
            {
                Console.WriteLine(k);
            }
```

100 %

Обратите внимание:

1. Подключено пространство имен ***using System.Linq;***
2. Сам запрос выглядит так:

```
var queryResults =
        from n in names
        where n.StartsWith("B")
        select n;
```

Результат запроса – переменная queryResults имет тип данных ***var.*** Этот тип предназначен для обобщенного типа переменных. Тип данных будет самостоятельно выбран компилятором по результатам запроса (число, строка, массив строк и т.д.).

```
        from n in names
```

-- эта конструкция указывает на источник данных – массив (в нашем случае) или коллекцию,но не единичный объект. А переменная **n** просто переменная, принимающая поочередно значения кадого из элементов этого перечисления.

```
        where n.StartsWith("B")
```

-- это указание условия, здесь может быть описано любое возможное булевское выражение, применительно к переменной **n.** Например – длина больше 5 символов (where n.Length>5) или содержит букву «а» where n.Contains("a").

```
        select n;
```

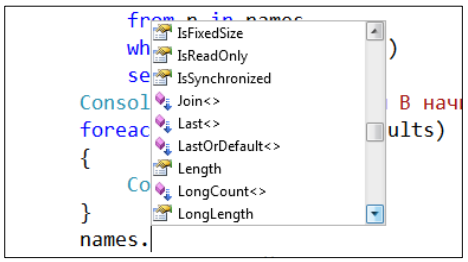
-- формируем отобранные элементы в выборку. Эта выборка будет являться результатом нашего запроса, т.е. значением переменной queryResults.

Задание 3.5. Использование запроса для отбора данных

По приведенному выше примеру напишите программу, которая будет выводить имена, в которых содержится буква, вводимая с клавиатуры пользователем.

Как узнать, какие запросы можно создать к определенному набору данных?

Вернитесь к своему проекту. Наберите names и поставьте точку. И вы увидите все возможные варианты в выпадающем списке интеллектуальных подсказок.



2 способ: Использование методов Linq

Выполним ту же задачу, но используем другой механизм, лямбда-выражение.

Пример.

```
{
    static void Main(string[] args)
    {
        string[] names = { "Ann", "Peter", "Bob", "Joile", "Ljuse", "Olga",
                           "Boris", "Helga", "Bill","Zarra" };

        var queryResults = names.Where(n => n.StartsWith("B"));

        Console.WriteLine("С буквы В начинаются имена: ");
        foreach (var k in queryResults)
        {
            Console.WriteLine(k);
        }

        Console.ReadKey();
    }
```

Комменарии:

Лямбда-выражение: `var queryResults = names.Where(n => n.StartsWith("B"));`

Метод Where `(n => n.StartsWith("B"))` применяется к каждому элементу массива names. Компилятор определяет, что метод Where должен вернуть выражение типа string, а так как таких значений несколько, то они будут собраны в коллекцию (массив)

Задание 3.6. Добавьте к решению новый проект и испытайте использование лямбда-выражений на практике.

2.4. Упорядочивание (сортировки)

И в этом случае можно использовать два варианта: запросы и методы. Для того, чтобы результаты работы были сразу видны, проследите, чтобы в наборе имен было несколько значений, начинающихся на одну букву и они бы шли не в алфавитном порядке.

1 вариант: LINQ-запрос

```
{
    static void Main(string[] args)
    {
        string[] names = { "Ann", "Peter", "Bob", "Joile", "Ljuse", "Olga",
                           "Boris", "Helga", "Bill","Barbara","Zarra" };

        var queryResults =
            from n in names
            where n.StartsWith("B")
            orderby n
            select n;
        Console.WriteLine("С буквы B начинаются имена: ");
        foreach (var k in queryResults)
        {
            Console.WriteLine(k);
        }

        Console.ReadKey();
```

2 вариант: метод Linq

```
    static void Main(string[] args)
    {
        string[] names = { "Ann", "Peter", "Bob", "Joile", "Ljuse", "Olga",
                           "Boris", "Helga", "Bill","Barbara","Zarra" };

        var queryResults = names.OrderBy(n => n).Where(n => n.StartsWith("B"));

        Console.WriteLine("С буквы B начинаются имена: ");
        foreach (var k in queryResults)
        {
            Console.WriteLine(k);
        }
```

Во втором примере выражение стало более сложно восприниматься для понимания. Из=за этой сложности чтения отдают предпочтение построению запросов.

Задание 3.7. Добавьте новые проекты и испытайте оба примера на практике.

С помощью Linq и специальных агрегатных операций (см. табл.1) можно легко анализировать результаты запроса не используя циклы.

Таблица 1. Агрегатные операции.

Операция	Описание
Count()	Количество результатов
Min()	Минимальное значение среди результатов
Max()	Максимальное значение среди результатов
Averege()	Среднее значение результатов
Sum()	Сумма числовых значение среди результатов

Пример: Имеется массив из большого количества (100 000) случайных чисел, выбираемых из интервала [0;1000]. Применим агрегатные состояния для выборки чисел, больших 500.

Фрагмент программы:

```
static void Main(string[] args)
{
    int[] numbers = new int[100000];
    Random rand = new Random(0);
    for (Int32 i = 0; i < 100000; i++)
    {
        numbers[i] = rand.Next(0, 1000);
    }
    var queryResults =
        from n in numbers
        where n <= 500
        select n;
    Console.WriteLine("Кол-во чисел <=500 {0}", queryResults.Count());
    Console.ReadKey();
```

Комментарии: Обращение к конструктору класса Random с параметро 0

Random rand = new Random(0);

гарантирует, что всегда будет вызываться одна и та же последовательность чисел.

Задание 3.8. Доработайте код примера 2 так, чтобы продемонстрировать выполнение всех агрегатных операций..

Использование Linq облегчает работу с коллекциями структур.

Задание 3.9. Использование Linq для работы с коллекциями структур

1. Опишите структуруCustomer (Заказчик) с полями(int ID, string Country, string City, String Name, int age)
2. Опишите для структуры открытый метод Print, выводящий на экран значения полей.
3. Считайте из файла данные в список (List) Заказчики (Customers) (20элементов)
4. Создайте запрос для вывода Заказчиков из Белоруссии. Запрос будетвыглядеть примерно, так:

```
Var queryResult=
      from c in Customers
    where c.Country==”Белоруссия”
    select c;
……….
Console.Writeline(“Заказчики из Белоруссии”);
Foreach (Customers c in queryResult)
{
c.Print;

}
```

Отсортируйте их перед выводом на печать по именам.

Использование многоуровневой сортировки.

Если структура достаточно сложная, а данных много, то может потребоваться упорядочивать по нескольким параметрам, например, сначала по странам, внутри каждой страны по алфавиту городов, а для каждого города по алфавиту названий.

Посмотрите на этот фрагмент:

```
Var queryResult= customers.OrderBy(c=>c.Country)
      .ThenBy((c=>c.City)
       ThenBy((c=>c.Name)
    select c;
```

Задание 3.10. Доведите проект с описанным примером до рабочего состояния.

3. Файлы.

3.1 Пространство имен System.IO

Файл – поименованная область совокупности данных, расположенных во внешней памяти. При больших объемах данных повторное введение их при каждом новом запуске не целесообразно и утомительно. Кроме того, вывод результата в большинстве программ тоже желательно осуществлять во внешний файл.

Рассмотрим возможность записи и чтения потока символов (строк), представленных в кодировке Unicode. Для этого будут использоваться классы StreamWriter и StreamReader. Для работы с потоками в первую очередь необходимо подключить библиотеку **System.IO;**

Using System.IO;

Пространство имен System.IO содержит все необходимые классы, методы и свойства для манипуляций с каталогами и файлами (В Таблице 1 приведены основные классы).

Таблица 2.

Класс	Применение
Binary Reader и Writer	Чтение и запись простых типов данных
Directory, File, DirectoryInfo и FileInfo	Создание, удаление и перемещение файлов и директорий. Получение подробной информации о файлах, при помощи свойств, определенных в этих классах.
FileStream	Доступ к файлам потоковым способом
MemoryStream	Доступ к данным храняшимся в памяти
StreamWriter и StreamReader	Чтение и запись текстовой информации
StringReader и StringWriter	Чтение и запись текстовой информации из строкового буфера

3.2 Запись в текстовый файл

Для записи в файл нужно сначала создать файл и открыть поток для работы с ним **StreamWriter** sw = **File.CreateText**("test.txt");

Здесь был создан поток **sw**, а при помощи метода **File.CreateText("test.txt")** создан файл «test.txt»(по умолчанию сохраняется в папку Debug сохраненного проекта).

После того как поток был открыт, можно записывать в него текстовые строки используя методы **Write** и **WriteLine**

sw.WriteLine("Мой первый файл созданый в C#");

Но если сейчас, после запуска программы, проверить файл, то мы увидим, что он пуст. Это нормально. Данные не появятся в файле до тех пор, пока не будет закрыт поток sw. Делается это с помощью метода **Close**: **sw.Close();**

Теперь после запуска можно увидеть, что в файле появилась строка введенного текста.

Выше была рассмотрена ситуация, когда файл не существовал ранее. Но возможна и другая ситуация, программе требуется дописать какой-либо текст в уже имеющийся файл. Для этого нам понадобится метод **AppendText**. Использовать его нужно вместо **CreateText:**

StreamWriter sw = **File.AppendText**("test.txt");

sw.WriteLine("Добавили вторую строчку в файл");

sw.Close();

3.3 Чтение из текстового файла

Для чтения в первую очередь нужно открыть поток класса **StreamReader**, привязав его к файлу. Делается это при помощи метода **File.OpenText**(‘имя файла’);

StreamReader sr = File.OpenText("test.txt");

Если необходимо считать только первую строку из файла, то все просто. Для этого воспользуется следующая конструкция: **sr.ReadLine().**

Допустим, что требуется вывести первую строчку из файла на консоль. Делается следующим образом:

System.Console.WriteLine(sr.ReadLine());

Немного сложнее вариант, когда количество строк в файле заранее не известно, а нужно считать содержимое всего файла. Чтение из файла проводится построчно. Для вывода всего файла придется построчно его считывать до тех пор пока в нем есть строчки. Фрагмент кода приведен ниже.

```
while (true){
string st = sr.ReadLine();
if (st==null)
break;
System.Console.WriteLine(st);
}
```

В этом примере файл считывается построчно и выводится на консоль.

Как и при записи в файл, так и при чтении из него не стоит забывать о закрытии потока методом Close.

```
Sr.Close();
```

Задание 3.11. Создадать файл, записать в него 2 строчки текста и после этого вывести их на консоль

```
StreamWriter sw = File.CreateText("test.txt");
sw.WriteLine("Первая строка");
sw.WriteLine("Вторая строка");
sw.Close();
StreamReader sr = File.OpenText("test.txt");
while (true)
{
string st = sr.ReadLine();
if (st==null)
break;
System.Console.WriteLine(st);
}
sw.Close();
System.Console.ReadLine();
```

3.4 Файлы с разделителями.

Файлы с разделителями – это распространенная форма хранения данных, используемая во многих системах. Например, файл хранит данные о студентах. В каждой строке файла хранится информация об одном студенте, в качестве разделителей могут служить, например, запятые: «Иванов, Иван, 1992, Прикладная информатика».

Данные из такого файла будут читаться построчно, а затем они должны быть разделены на отдельные части, соответствующие разделителям. В нашем примере из каждой строки следует выделить *«Фамилию», «Имя», «Год рождения», «Специальность»*.

Для этого понадобится строковая команда *Split*.

<строка>.Split () – преобразует строку в массив строк, разбивая ее на части по указанным символам-разделителям. Эти символы указываются в массиве char, который может содержать, например, запятую, пробел и др. символы. Символы-разделители указываются в апострофах.

```
String [] MyWords; //описание массива для хранения слов строки
MyWords=MyString.Split(‘,’);//формирование массива слов
```

Далее все слова в этом массиве выводятся на консоль с помощью цикла foreach:

```
Foreach(string word in MyWords)
{
Console.WriteLine(“{0}”,word);
}
```

При работе команды Split, все лишние пробелы удаляются, остаются только сами слова.

3.5 Работа с классами DirectoryInfo и FileInfo

Классы DirectoryInfo и FileInfo унаследованы от FileSystemInfo, который является абстрактным. Это значит, что Вы не можете унаследовать от него свой класс, но можете использовать свойства, определённые в нём. В таблице 3 перечислены его свойства и методы.

Таблица 3. Свойства и методы класса FileSystemInfo

Свойства	Назначение
Attributes	Возвращает атрибуты файла в виде значений перечисления FileAttributes
CreationTime	Возвращает время создания файла
Exists	Проверяет является ли файл директорией или нет
Extension	Возвращает расширение файла
LastAccessTime	Возвращает время последнего доступа к файлу
FullName	Возвращает полный путь к файлу
LastWriteTime	Возвращает время последнего изменения файла
Name	Возвращает имя данного файла
Delete()	Удаляет файл. Будьте осторожны при использовании этого метода.

Класс DirectoryInfo содержит методы для создания, перемещение и удаление каталогов. Чтобы использовать вышеприведённые свойства, необходимо создать объект класса DirectoryInfo как показано в примере:

```
DirectoryInfo dir1 = new DirectoryInfo(@"F:\WINNT");
```

После этого уже можно просмотреть свойства директории при помощи объекта dir1, как показано на фрагмент кода:

```
Console.WriteLine("Full Name is : {0}", dir1.FullName);
Console.WriteLine("Attributes are : {0}",
                  dir1.Attributes.ToString());
```

3.6 Работа с файлами в директории

Предположим, вы хотите получить список всех файлов с расширением BMP в папке F:\Pictures. Для этого можно использовать следующий код:

```
DirectoryInfo dir = new DirectoryInfo(@"F:\WINNT");
FileInfo[] bmpfiles = dir.GetFiles("*.bmp);
Console.WriteLine("Total number of bmp files", bmpfiles.Length);
Foreach( FileInfo f in bmpfiles)
{
  Console.WriteLine("Name is : {0}", f.Name);
  Console.WriteLine("Length of the file is : {0}", f.Length);
  Console.WriteLine("Creation time is : {0}", f.CreationTime);
  Console.WriteLine("Attributes of the file are : {0}",
                    f.Attributes.ToString());
}
```

3.7 Создание подкаталогов

Следующий фрагмент кода описывает как можно создать поддиректорию MySub в директории Sub:

```
DirectoryInfo dir = new DirectoryInfo(@"F:\WINNT");
```

```
try
{
  dir.CreateSubdirectory("Sub");
  dir.CreateSubdirectory(@"Sub\MySub");
}
catch(IOException e)
{
  Console.WriteLine(e.Message);
}
```

3.8 Создание файлов при помощи класса FileInfo

Класс FileInfo позволяет создавать новые файлы, получать информацию, удалять и перемещать их. В этом классе также есть методы для открытия, чтения и записи в файл. В следующем примере показано, как можно создать текстовый файл и получить доступ к его информации (времени его создания, полное имя, и так далее):

```
FileInfo fi = new FileInfo(@"F:\Myprogram.txt");
FileStream fstr = fi.Create();
Console.WriteLine("Creation Time: {0}",f.CreationTime);
Console.WriteLine("Full Name: {0}",f.FullName);
Console.WriteLine("FileAttributes: {0}",f.Attributes.ToString());

//Удаление файла Myprogram.txt.

Console.WriteLine("Press any key to delete the file");
Console.Read();
fstr.Close();
fi.Delete();
```