Занятие 5, ч1. Строки, регулярные выражения, ввод-вывод и работа с файловой системой

Тренер: Алексей Дышлевой

Основные вопросы

- Робота з рядками. Клас String. Рядкові літерали. Специфікатори формату й глобалізація. Object.ToString, IFormattable, ICustomFormatter та CultureInfo. Порівняння рядків.
- Регулярні вирази. Пошук та групування. Заміна тексту за допомогою Regex.
- Клас Console. Метод ReadLine. Метод ReadKey. Метод Read. Метод WriteLine. Метод Write
- Файловий ввід-вивід. Класи StreamReader та StreamWriter. Класи FileAccess та FileAttribute. Класи Directory та DirectoryInfo, Path, FileInfo и DriveInfo. Класи BinaryReader та BinaryWriter. Класи StringReader та StringWriter. Класи TextReader та TextWriter. Класи XmlReader та XmlWriter. Клас ToBase64Transform. Класи FileStream, BufferedStream та MemoryStream
- Серіалізація та десеріалізація

Класс Console

- Предоставляет стандартные потоки для консольных приложений: входной, выходной и поток сообщений об ошибках. Этот класс не наследуется
- ReadLine - Считывает следующую строку символов из стандартного входного потока.
- ReadKey - Получает следующий нажатый пользователем символ или функциональную клавишу. Нажатая клавиша отображается в окне консоли.
- Read - Читает следующий символ из стандартного входного потока
- WriteLine - Записывает текстовое представление заданного значения типа, за которым следует текущий признак конца строки, в стандартный выходной поток.
- Write - Записывает текстовое представление заданного значения типа в стандартный выходной поток.

Строки – класс String

- Строка является упорядоченной коллекцией символов Юникода, используемой для представления текста. Объект String является упорядоченной коллекцией объектов System.Char, представляющей строку. Значением объекта String является содержимое упорядоченной коллекции, и это значение является неизменяемым (т. е. доступным только для чтения). Максимальный размер объекта String в 2 ГБ памяти, или около 1 миллиард символов.

String - конструкторы

Имя	Описание
String(Char*)	Инициализирует новый экземпляр класса String значением, на которое указывает заданный указатель на массив знаков Юникода.
String(Char[])	Инициализирует новый экземпляр класса String значением, заданным в виде массива знаков Юникода.
String(SByte*)	Инициализирует новый экземпляр класса String значением, определенным указателем на массив 8-разрядных целых чисел со знаком.
String(Char, Int32)	Инициализирует новый экземпляр класса String значением, полученным путем повторения заданного знака Юникода заданное число раз.
String(Char*, Int32, Int32)	Инициализирует новый экземпляр класса String значением, которое определяется заданным указателем на массив знаков Юникода, начальной позицией знака в этом массиве и длиной.
String(Char[], Int32, Int32)	Инициализирует новый экземпляр класса String значением, заданным массивом знаков Юникода, начальной позицией знака в пределах данного массива и длиной.
String(SByte*, Int32, Int32)	Инициализирует новый экземпляр класса String значением, определяемым заданным указателем на массив 8-битовых целых чисел со знаком, позицией начальной в пределах данного массива и длиной.
String(SByte*, Int32, Int32, Encoding)	Инициализирует новый экземпляр класса String значением, определенным заданным указателем на массив 8-разрядных целых чисел со знаком, начальной позицией в пределах данного массива, длиной и объектом Encoding.

Основные методы класса String

- Contains - Возвращает значение, указывающее, содержит ли указанная строка значение подстроки переданной в качестве параметра.
- Format – форматирование строки, как в консольных приложениях
- IndexOf -Возвращает индекс с отсчетом от нуля первого вхождения значения указанной строки в данном экземпляре
- Insert - Возвращает новую строку, в которой указанная строка вставляется на указанной позиции индекса в данном экземпляре
- Remove - Возвращает новую строку, в которой были удалены все символы, начиная с указанной позиции и до конца в текущем экземпляре
- Replace - Возвращает новую строку, в которой все вхождения заданной строки в текущем экземпляре заменены другой заданной строкой.
- ToLower() и ToUpper() - Возвращает копию этой строки, переведенную в верхний (ToUpper) или нижний (ToLower) регистр
- Substring - Извлекает подстроку из данного экземпляра. Подстрока начинается с указанной позиции знака и имеет указанную длину
- ToCharArray() - Копирует знаки данного экземпляра в массив знаков Юникода.

Управляющие символы (Escape-последовательности)

Escape-последовательность	Представляет
\a	Звонок (предупреждение)
\b	Удаление предыдущего символа
\f	Перевод страницы
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\'	Одиночная кавычка, используется для литералов char
\ "	Двойная кавычка, используется для объявления строковых литералов
****	Обратная косая черта
\?	Литерал вопросительного знака
\ooo	Символ ASCII в восьмеричной нотации
\xhh	Символ ASCII в шестнадцатеричной нотации
\xhhhh	Символ юникода в шестнадцатеричном формате, если эта escape-последовательность используется в многобайтовой знаковой константе или строковом литерале юникода. Например, `WCHAR f = L'\x4e00'` или `WCHAR b[] = L"The Chinese character for one is \x4e00"`.

Задача

- Задать некоторую текстовую строку не менее 300 символов. Строка должна содержать путь к файлу и текст.
- Удалить из строки все символы «n».
- После каждого символа «,» добавить символ «-».
- Вставить в строку символы переноса на следующую строку после последнего слова, близкого к символу, кратному 50-и. Т.е., сделать переносы текста, не разрывая слова. Обновлять переност текста после каждого изменения строки.
- Добавить к текущей еще одну строку.
- Определить, включает ли строка следующую подстроку «hello».
- Получить ссылку на текущую строку.

Сравнение и сортировка строк

- Правила для сортировки и сравнения строк, отличаются в зависимости от языка и региональных параметров. Например, порядок сортировки может определяться на фонетике или на визуальном представлении символов. В языках Восточной Азии символы сортируются по штрихам и корням идеограмм. Сортировка также зависит от того, какой порядок используют региональные параметры для алфавита. Например, датский языке есть символ «Æ», он действует в алфавите после буквы «Z».Кроме того, сравнения могут быть зависеть от регистра символов, и в некоторых случаях, в которых перечисляются правила также различается. Порядковое сравнение, с другой стороны, использует кодовых точек юникода отдельных символов в строке сравнение и сортировка строк.
- Правила сортировки определяют алфавитный порядок знаков Юникода и принципы сравнения двух строк.

Пример сравнения

если текущий язык и региональные параметры (США), то первый вызов метода
String.Compare(String, String, StringComparison)
(с помощью сравнения с учетом языка и региональных параметров) проверяет «a» меньше «a»,
но второй вызов того же метод а
(с помощью порядкового сравнения) проверяет «a» больше «a».

```
using System;
using System.Globalization;
using System.Threading;
public class Example
 {
    public static void Main()
    {
       Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US");
       Console.WriteLine(String.Compare("A", "a", StringComparison.CurrentCulture));
       Console.WriteLine(String.Compare("A", "a", StringComparison.Ordinal));
    }
 }
 // The example displays the following output:
// 1
// -32
```

Сортировка

- Платформа .NET Framework поддерживает правила сортировки по словам, строкам и порядковым номерам:
- При сортировке по словам выполняется сравнение строк с учетом языка и региональных параметров, при котором некоторые символы Юникода, отличные от букв и цифр, могут иметь специально присвоенные им весовые коэффициенты. Например, дефису (-) можно присвоить очень низкий весовой коэффициент, и тогда в отсортированном списке слова "coop" и "co-op" окажутся рядом.
- При сортировке строк выполняется сравнение с учетом языка и региональных параметров. Она аналогична сортировке по словам, за исключением того, что особых случаев нет и все буквы и цифры следуют после всех остальных символов Юникода. Две строки можно сравнить с использованием правил сортировки строк, вызывая перегрузки метода CompareInfo.Compare с параметром *options*, в который передается значение CompareOptions.StringSort. Это единственный метод который.NET Framework предоставляет для сравнения двух строк, используя правила сортировки по строке.
- При сортировке по порядковому номеру строки сравниваются на основе числовых значений каждого объекта Char в строке. Порядковое сравнение автоматически выполняется независимо от регистра, поскольку один и тот же символ в нижнем и верхнем регистре имеет разные кодовые точки. Однако если регистр не важен, можно задать порядковое сравнение, при котором игнорируется регистр.

Сортировка

- Если нет дополнительных условий сортировки, лучше использовать Sort() класса Array или коллекций

Задача

- Создать список студентов со следующей информацией: имя, фамилия, дата рождения (DateTime), курс, номер студенческого, средний балл каждого курса, на котором учился студент, итоговый средний балл по всем годам обучения.
- Отсортировать список студентов по имени/ фамилии, по курсу, по среднему баллу с использованием Compare и Sort

Форматы строк

- String.Format - Заменяет каждый элемент формата в указанной строке текстовым эквивалентом значения соответствующего объекта.

Имя	Описание
Format(String, Object)	Заменяет один или более элементов формата в указанной строке строковым представлением указанного объекта.
Format(String, Object[])	Заменяет элемент формата в указанной строке строковым представлением соответствующего объекта в указанном массиве.
Format(IFormatProvider, String, Object[])	Заменяет элементы формата в указанной строке строковым представлениями соответствующих объектов в указанном массиве. Параметр предоставляет сведения об особенностях форматирования, связанных с языком и региональными параметрами.
Format(String, Object, Object)	Заменяет элементы формата в указанной строке строковым представлением двух указанных объектов.
Format(String, Object, Object, Object)	Заменяет элементы формата в указанной строке строковым представлением трех указанных объектов.

Синтаксис форматирования

- **{ *index*[,*alignment*][:*formatString*] }**
- ***Индекс*** - Индекс с отсчетом от нуля аргумента строковое представление которого требуется включить в этой позиции в строке. Если этот аргумент **null**, то пустая строка будет включена в этой позиции в строке.
- ***Alignment*** - Необязательно. Знаковое целое число, указывающее общую длину поля, в которое вставляется аргумент и является ли он выровнен по правому краю (положительное целое число) или по центру (отрицательное целое число). Если опустить *выравнивание*, строковое представление соответствующего аргумента вставляется в поле без пробелов.
- ***formatString*** - Необязательно. Строка, которая определяет формат результирующей строки соответствующего аргумента. Если опустить *formatString*, метод **ToString** соответствующего аргумента не вызывается для создания его строковое представление. Если задано *formatString*, аргумент указанный элемент форматирования должен реализовать интерфейс IFormattable.

Пример

```
DateTime dat = new DateTime(2012, 1, 17, 9, 30, 0);
string city = "Chicago";
int temp = -16;
string output = String.Format("At {0} in {1}, the temperature was {2} degrees.", dat, city, temp);
Console.WriteLine(output);

//форматирование валюты
String.Format("{0,-10:C}", 126347.89m);

short[] values= { Int16.MinValue, -27, 0, 1042, Int16.MaxValue };
 Console.WriteLine("{0,10} {1,10}\n", "Decimal", "Hex");
 foreach (short value in values)
{
 string formatString = String.Format("{0,10:G}: {0,10:X}", value);
 Console.WriteLine(formatString);
}
// The example displays the following output:
// Decimal   Hex
//
// -32768:  8000
//    -27:  FFE5
//      0:     0
//   1042:   412
//  32767:  7FFF
```

Задача

- Создать форматированную строку для вывода информации о студенте в виде таблицы:
- Имя, фамилия | Дата роджения | курс | номер студенческого | Средний балл
- | 1 | 2 | 3 | 4 | 5 | Итог

- Вывести студентов в отформатированном виде

Пример – передача массива аргументов для форматирования

```
public class Example
 {
    public static void Main()
    {
      CityInfo nyc2010 = new CityInfo("New York", 8175133, 302.64m, 2010);
      ShowPopulationData(nyc2010);
      CityInfo sea2010 = new CityInfo("Seattle", 608660, 83.94m, 2010);
      ShowPopulationData(sea2010);
    }
    private static void ShowPopulationData(CityInfo city)
    {
      object[] args = { city.Name, city.Year, city.Population, city.Area };
      String result = String.Format("{0} in {1}: Population {2:N0}, Area {3:N1} sq. feet", args);
      Console.WriteLine(result);
    }
 }
```

Интерфейс IFormattable

- Интерфейс IFormattable преобразует объект в строковое представление на основании строки формата и поставщика формата.
- Строка формата обычно определяет общий внешний вид объекта. Платформа .NET Framework поддерживает следующие:
  - Стандартный формат строк для форматирования значений перечисления
  - Стандартный и настраиваемый формат строк для форматирования числовых значений
  - Стандартный и настраиваемый формат строк для форматирования значений даты и времени
  - Стандартный и настраиваемый формат строк для форматирования временных интервалов
- Более подробно о форматах можно узнать по ссылках в пункте «Заметки» для IFormattable (http://msdn.microsoft.com/ru-ru/library/system.iformattable(v=vs.110).aspx)

Примеры

Строки стандартных числовых форматов

"C" или "c"	Валюта 123.456 ("C", en-US) -> $123.46
"E" или "e"	Экспоненциальный (научный) 1052.0329112756 ("E", en-US) -> 1.052033E+003
"P" или "p"	Процент 1 ("P", en-US) -> 100.00 %

Интерфейс IFormattable

- Классы, которым необходимы дополнительные возможности форматирования строк, по сравнению с возможностями метода Object.ToString должны реализовать интерфейс IFormattable.
- Класс, который реализует IFormattable должен поддерживать код форматирования "G" (общий). Кроме описателя "G", в классе можно определить перечень других описателей форматирования, которые он поддерживает. Кроме того, класс должен быть подготовлен для обработки спецификатора формата, значение которого **null**.

Пример

- Форматирование температур из документа (пример 1)

Форматирование и язык

- Как правило, объекты в списке аргументов преобразованы в их строковым представлении с помощью соглашений текущего языка и региональных параметров, которые возвращается свойством CultureInfo.CurrentCulture. Используется это расширение функциональности с помощью вызова Format(IFormatProvider, String, Object[]). Параметр *provider* данной перегруженной реализации IFormatProvider, и сведения о форматировании для конкретных языков и региональных параметров, используются в качестве переопределения в следующих классах:
- CultureInfo . Метод GetFormat возвращает объект NumberFormatInfo языка и региональных параметров для форматирования числовых значений и объектDateTimeFormatInfo языка и региональных параметров для форматирования значения даты и времени.
- DateTimeFormatInfo , который используется для форматирования для конкретных языков и региональных параметров значений даты и времени.
- NumberFormatInfo , который используется для языка и региональных параметров форматирования числовых значений.

Пример

```
using System.Globalization;
class Program
{
 static void Main(string[] args)
 {
   DateTime date = new DateTime(2013, 5, 30, 19, 00, 0);
   CultureInfo[] cultures = { new CultureInfo("en-US"),
   new CultureInfo("fr-FR"),
   new CultureInfo("it-IT"),
   new CultureInfo("de-DE"),
   new CultureInfo("ru-RU")};
   foreach (CultureInfo culture in cultures)
       Console.WriteLine("{0}: {1}",
    culture.Name, date.ToString(culture));
    Console.ReadKey();
 }
}
```

```
//результат
en-US: 5/30/2013 7:00:00 PM
fr-FR: 30/05/2013 19:00:00
it-IT: 30/05/2013 19:00:00
de-DE: 30.05.2013 19:00:00
ru-RU: 30.05.2013 19:00:00
```

Задача

- Реализовать вывод студентов из предыдущего примера с учетом форматирования даты рождения

IFormatProvider

- Предоставляет механизм извлечения объекта для управления форматированием.
- GetFormat - Возвращает объекты, предоставляющие службы форматирования для заданного типа.
- Интерфейс IFormatProvider предоставляет объект, предоставляющий сведения о форматировании для форматирования и анализа операций. Операции форматирования преобразуют значение типа в строковое представление этого значения.
- Реализации IFormatProvider часто используемые неявно для форматирования и анализа методов.

Операции пользовательского форматирования

- Можно также вызвать перегруженный Format(IFormatProvider, String, Object[]) для выполнения операций пользовательского форматирования. Например, можно указать целое число как идентификационный номер или как номер телефона. Для запуска пользовательского форматирования этот аргумент *provider* должен реализовывать интерфейсы и IFormatProvider и ICustomFormatter. Когда метод Format(IFormatProvider, String, Object[]) передается реализацией ICustomFormatter в качестве аргумента *provider*, этот метод вызывает Format реализации метода IFormatProvider.GetFormat и запросы объект типа ICustomFormatter. Затем вызывается метод Format возвращаемого объекта ICustomFormatter для форматирования каждого элемента форматирования в составной строке, переданной в него.

ICustomFormatter

- Определяет метод, поддерживающий нестандартное форматирование значения объекта.
- Интерфейс ICustomFormatter содержит единственный метод ICustomFormatter.Format. При реализации этого интерфейса с помощью ссылочного типа или типа значения, метод Format возвращает строковое представление значения объекта с пользовательским форматированием.
- Как правило, интерфейс ICustomFormatter реализуется с помощью интерфейса IFormatProvider настраивать расширение функциональности 2 методов формата строки .NET Framework

ICustomFormatter – последовательность реализации

- Создайте производный класс, реализующий интерфейс ICustomFormatter и единственный элемент, метод Format.
- Создайте производный класс, реализующий интерфейс IFormatProvider и единственный элемент, метод GetFormat. Метод GetFormat возвращает экземпляр реализации ICustomFormatter. Часто один класс реализует как ICustomFormatter, так и IFormatProvider. В этом случае реализация класса **GetFormat** возвращает экземпляр самого себя.
- Передайте реализацию IFormatProvider в качестве аргумента *provider* метода String.Format(IFormatProvider, String, Object[]) или сравнимого метода

ICustomFormatter

- Существует два типа реализаций интерфейса ICustomFormatter: встроенная и расширенная.
- Встроенные реализации являются реализациями, которые предоставляют настраиваемые параметры форматирования для определенного приложением объекта.
- Реализации расширения являются реализациями, которые предоставляют пользовательское форматирование для типа, который уже имеет поддержку форматирования.

Пример

- Пример 2 из документа

Задача

- Реализовать собственное форматирование времени в формате HH:MM:SS.MS для целого числа

Нормализация

- Некоторые символы Юникода имеют несколько представлений.
- Наличие нескольких представлений одного символа затрудняет выполнение поиска, сортировки, сопоставления и других строковых операций.
- В стандарте Юникода определен процесс, называемый нормализацией, который возвращает одно двоичное представление символа в кодировке Юникод для любого из нескольких эквивалентных двоичных представлений. При нормализации могут использоваться несколько алгоритмов, которые называются формами нормализации и следуют различным правилам. Платформа .NET Framework поддерживает формы нормализации Юникода C, D, KC и KD. После нормализации строк до одной и той же формы нормализации их можно сравнить с использованием порядкового сравнения.
- Определить, нормализована ли строка до формы нормализации C, можно путем вызова метода String.IsNormalized(); также можно вызватьString.IsNormalized(NormalizationForm) метод, чтобы определить, нормализована ли строка до указанной формы нормализации. Также можно вызвать методString.Normalize() для преобразования строки в форму нормализации C или вызвать метод String.Normalize(NormalizationForm) для преобразования строки в указанную форму нормализации. Пошаговые инструкции нормализации и сравнения строк см. в разделе Normalize() и методы Normalize(NormalizationForm).

Класс StringBuilder

- Класс StringBuilder можно использовать вместо класса String для операций, предполагающих внесение множественных изменений в значение строки. В отличие от экземпляров класса String, объекты StringBuilder являются изменяемыми; при сцеплении, добавлении или удалении подстрок из строки операции выполняются над одной строкой. Закончив изменение значения объекта StringBuilder, можно вызвать его метод StringBuilder.ToString, чтобы преобразовать его в строку.

Когда использовать String

- Если количество изменений в строке, которые выполняет приложение, мало.
- При выполнении фиксированного числа операций объединения, особенно со строковыми литералами. В этом случае компилятор может объединить операции объединения в единственную операцию.
- При необходимости выполнения масштабных операций поиска при построении строки. Класс StringBuilder - без методов поиска, такие как **IndexOf** или **StartsWith**.

Когда использовать StringBuilder

- Если предполагается, что приложение будет делать неизвестное число изменений в строке во время выполнения (например, при использовании цикла для сцепления случайного число строк, содержащих данные от пользователя).
- Если предполагается, что приложение будет вносить значительное число изменений в строку.

Пример

```
public static void Main()
{
 StringBuilder sb = new StringBuilder();
 ShowSBInfo(sb);
 sb.Append("This is a sentence.");
 ShowSBInfo(sb);
 for (int ctr = 0; ctr <= 10; ctr++)
 {
   sb.Append("This is an additional sentence.");
   ShowSBInfo(sb);
  }
}
 private static void ShowSBInfo(StringBuilder sb)
{
   foreach (var prop in sb.GetType().GetProperties())
   {
     if (prop.GetIndexParameters().Length == 0)
         Console.Write("{0}: {1:N0} ", prop.Name, prop.GetValue(sb));
   }
   Console.WriteLine();
}
```

```
// The example displays the following output:
// Capacity: 16 MaxCapacity: 2,147,483,647 Length: 0
// Capacity: 32 MaxCapacity: 2,147,483,647 Length: 19
// Capacity: 64 MaxCapacity: 2,147,483,647 Length: 50
// Capacity: 128 MaxCapacity: 2,147,483,647 Length: 81
// Capacity: 128 MaxCapacity: 2,147,483,647 Length: 112
// Capacity: 256 MaxCapacity: 2,147,483,647 Length: 143
// Capacity: 256 MaxCapacity: 2,147,483,647 Length: 174
// Capacity: 256 MaxCapacity: 2,147,483,647 Length: 205
// Capacity: 256 MaxCapacity: 2,147,483,647 Length: 236
// Capacity: 512 MaxCapacity: 2,147,483,647 Length: 267
// Capacity: 512 MaxCapacity: 2,147,483,647 Length: 298
// Capacity: 512 MaxCapacity: 2,147,483,647 Length: 329
// Capacity: 512 MaxCapacity: 2,147,483,647 Length: 360
```

Регулярные выражения

- System.Text.RegularExpressions предоставляет классы для работы с регулярными выражениями
- Регулярное выражение – это шаблон, согласно которому выполняется поиск соответствующего фрагмента текста. Использование регулярных выражений обеспечивает: проверку строки на соответствие шаблону, поиск в тексте по заданному шаблону, а также разбиение текста на фрагменты.
- Язык описания регулярных выражений состоит из символов двух видов: обычных символов и метасимволов. Обычный символ представляет в выражении сам себя, а метасимвол – некоторая совокупность символов.
- Подробно: http://msdn.microsoft.com/ru-ru/library/az24scfc.aspx

Наиболее употребляемые метасимволы

Набор символов	Описание	Пример
.	Любой символ, кроме \n.	Выражение c.t соответствует фрагментам: cat, cut, c#t, c{t и т.д.
[ ]	Любой одиночный символ из последовательности, записанной внутри скобок. Допускается использование диапазонов символов,для задания которого используется символ «-».	Выражение c[aui]t соответствует фрагментам: cat, cut, cit. Выражение c[a-c]t соответствует фрагментам: cat, cbt, cct.
[^]	Любой одиночный символ, не входящий в последовательность, записанную внутри скобок. Допускается использование диапазонов символов.	Выражение c[^aui]t соответствует фрагментам: cbt, cct, c2t и т.д. Выражение c[^a-c]t соответствует фрагментам: cdt, cet, c%t и т.д.
\w	Любой алфавитно-цифровой символ, а также символ подчеркивания.	Выражение c\wt соответствует фрагментам: cbt, cct, c2t, c_t и т. д., но не соответствует фрагментам c%t, c{t и т. д.
\W	Любой символ не удовлетворяющий \w.	Выражение c\Wt соответствует фрагментам: c%t, c{t, c.t и т.д., но не соответствует фрагментам cbt, cct, c2t и т.д.
\s	Любой пробельный символ (пробел, табуляция и переход на новую строку).	Выражение \s\w\w\w\s соответствует любому слову из трех букв, окруженному пробельными символами. Следует отметить, что слова стоящие на границе текста не удовлетворяют данному регулярному выражению, т.к. начало/конец строки не являются пробельными символами.
\S	Любой не пробельный символ.	Выражение \s\S\S\S\s соответствует любым трем непробельным символам, окруженным пробельными.

Наиболее употребляемые метасимволы

Набор символов	Описание	Пример
\b	Любой символ, кроме удовлетворяющих \b.	Выражение \B\d\d\d\B соответствует любым трем цифрам, входящим в состав слова так, что ни справа ни слева от них нет конца слова.
\d	Любая десятичная цифра.	Выражение c\dt соответствует фрагментам: c1t, c2t, c3t и т.д.
\D	Любой символ, не являющийся десятичной цифрой.	Выражение c\Dt не соответствует фрагментам: c1t, c2t, c3t и т.д.
|	Задает альтернативу, другими словами, символ | соответствует операции или.	Выражение c[a-c]|[0-2] соответствует фрагментам: ca, cb, cc, c0, c1, c2.
^	Если стоит в начале выражения, то фрагмент, совпадающий с регулярным выражением, следует искать только в начале текста. Иначе трактуется просто как символ.	Выражение ^cat определяет последовательность символов cat, расположенную в начале строки.
$	Если стоит в конце выражения, то фрагмент, совпадающий с регулярным выражением, следует искать только в конце текста. Иначе трактуется просто как символ.	Выражение cat$ определяет последовательность символов cat, расположенную в конце строки.
\A	Аналог ^ для многострочной строки.	
\Z	Аналог $ для многострочной строки.	

Кванторы (повторители) в регулярных выражениях

Квантификатор	Описание	Шаблон	Соответствия
*	Соответствует предыдущему элементу ноль или более раз.	`\d*\.\d`	".0", "19.9", "219.9"
+	Соответствует предыдущему элементу один или более раз.	`"be+"`	"bee" в "been", "be" в "bent"
?	Соответствует предыдущему элементу ноль или один раз.	`"rai?n"`	"ran", "rain"
{ *n* }	Предыдущий элемент повторяется ровно *n* раз.	`",\d{3}"`	",043" в "1,043.6", ",876", ",543" и ",210" в "9,876,543,210"
{ *n* ,}	Предыдущий элемент повторяется как минимум *n* раз.	`"\d{2,}"`	"166", "29", "1930"
{ *n* , *m* }	Предыдущий элемент повторяется как минимум *n* раз, но не более чем *m* раз.	`"\d{3,5}"`	"166", "17668" "19302" в "193024"
*?	Предыдущий элемент не повторяется вообще или повторяется, но как можно меньшее число раз.	`\d*?\.\d`	".0", "19.9", "219.9"
+?	Предыдущий элемент повторяется один или несколько раз, но как можно меньшее число раз.	`"be+?"`	"be" в "been", "be" в "bent"
??	Предыдущий элемент не повторяется или повторяется один раз, но как можно меньшее число раз.	`"rai??n"`	"ran", "rain"
{ *n* }?	Предыдущий элемент повторяется ровно *n* раз.	`",\d{3}?"`	",043" в "1,043.6", ",876", ",543" и ",210" в "9,876,543,210"
{ *n* ,}?	Предыдущий элемент повторяется как минимум *n* раз (как можно меньше).	`"\d{2,}?"`	"166", "29", "1930"
{ *n* , *m* }?	Предыдущий элемент повторяется не менее *n* и не более *m* раз (как можно меньше).	`"\d{3,5}?"`	"166", "17668" "193", "024" в "193024"

Регулярные выражения

- Регулярное выражение записывается в виде строкового литерала, при этом, перед строкой желательно ставить символ @, который говорит о том, что строку нужно будет рассматривать и в том случае, если она будет занимать несколько строчек на экране. Ниже приведены примеры регулярных выражений вместе с их описаниями.
- Слово ua – @“ua"
- Целое число со знаком, или без знака – @"[+-]?\d+"
- Номер телефона в формате xxx-xx-xx – @"\d\d\d-\d\d-\d\d" или
- @"\d{3}(-\d\d){2}"
- Время в формате чч.мм, или чч:мм – @"([01]\d)|(2[0-4])[\.:][0-5]\d«
- E-mail - [.\\-_a-z0-9]+@([a-z0-9][\\-a-z0-9]+\\.)+[a-z]{2,6}

Класс Regex

- Класс Regex представляет обработчик регулярных выражений .NET Framework. Его можно использовать, чтобы быстро просмотреть большие объёмы текста в поисках конкретных шаблонов символов; извлечь, редактировать, заменить или удалить подстроки; добавить извлечённые строки к коллекции для создания отчёта.
- Для использования регулярных выражений следует определить шаблон, который необходимо идентифицировать в текстовом потоке, используя специальный синтаксис регулярных выражений.
- Далее, при необходимости можно создать объект Regex.
- Наконец, следует вызвать метод, который выполняет некоторую операцию, например, замена текста, соответствующего шаблону регулярного выражения, или определение соответствия текста шаблону.
-

Функциональные возможности Regex

- Проверка совпадения – метод IsMatch
- Извлечение одного совпадения – вызов метода Match позволяет извлечь объект Match, который предоставляет первое совпадение, найденное в строке или части строки. Последующие совпадения находятся методом Match.NextMatch
- Извлечение всех совпадений. Вызов метода Matchesпозволяет извлечь объект System.Text.RegularExpressions.MatchCollection, который представляет все совпадения, найденные в строке или части строки
- Замена совпадающего текста - вызов метода Replace. Регулярным выражением также может быть определен текст замещения. Некоторые методы Replace включают параметр MatchEvaluator, который позволяет программно определить текст замены.
- Создание массива строк, сформированный на основе части входной строки. Вызов метода Split позволяет разбить входную строку в позициях, определенных регулярным выражением

Пример

- string strText =
- "Lorem ipsum dolor sit amet, vasya.pupkin@gmail.com consectetur adipiscing elit. "+
- "In posuere, elit ut tristique condimentum, lectus est sodales nibh, "+
- "sed adipiscing velit address@mail.ru lectus vel felis. Praesent id urna "+
- "ut quam dapibus sollicitudin sit amet et mi. Quisque in magna nisi, in "+
- "scelerisque mi admin@mysite.com.ua! Vestibulum suscipit lacinia tempor. Donec "+
- "euismod massa sit amet tellus consectetur dapibus. Donec nisl justo, egestas at "+
- "mattis ut, sagittis eu ipsum. Aliquam porttitor massa support@somedomain.info in.";
- string strFind = "([A-Z;a-z;0-9;\x2E;\x2D;_]+)?@([A-Z;a-z;0-9;\x2E;-;]+)?";
- string strReplace = "$1";
- Console.WriteLine(strText);
- Console.WriteLine();
- Regex regex = new Regex(strFind);
- MatchCollection matches = regex.Matches(strText);
- foreach(var match in matches)
- {
- Console.WriteLine(match);
- }
- Console.WriteLine();
- strText = regex.Replace(strText, strReplace, 10);
- Console.WriteLine(strText);
- Console.ReadKey();

Результат

- Lorem ipsum dolor sit amet, vasya.pupkin@gmail.com consectetur adipiscing elit.
- In posuere, elit ut tristique condimentum, lectus est sodales nibh, sed adipisci
- ng velit address@mail.ru lectus vel felis. Praesent id urna ut quam dapibus soll
- icitudin sit amet et mi. Quisque in magna nisi, in scelerisque mi admin@mysite.c
- om.ua! Vestibulum suscipit lacinia tempor. Donec euismod massa sit amet tellus c
- onsectetur dapibus. Donec nisl justo, egestas at mattis ut, sagittis eu ipsum. A
- liquam porttitor massa support@somedomain.info in.
- vasya.pupkin@gmail.com
- address@mail.ru
- admin@mysite.com.ua
- support@somedomain.info
- Lorem ipsum dolor sit amet, vasya.pupkin consectetur adipiscing elit. In posuere
- , elit ut tristique condimentum, lectus est sodales nibh, sed adipiscing velit a
- ddress lectus vel felis. Praesent id urna ut quam dapibus sollicitudin sit amet
- et mi. Quisque in magna nisi, in scelerisque mi admin! Vestibulum suscipit lacin
- ia tempor. Donec euismod massa sit amet tellus consectetur dapibus. Donec nisl j
- usto, egestas at mattis ut, sagittis eu ipsum. Aliquam porttitor massa support i
- n.

Особенности Regex

- Регулярные выражения – очень мощное средство обработки данных.
- При этом методы работаю примерно в 10 раз медленнее, чем аналоги string

Домашнее задание

- Создать класс для хранения информации о клиентах банка: имя, фамилию, дату рождения, номер телефона, e-mail, паспортные данные, номера и даты карточек.
- Реализовать возможности сортировки по имени/ фамилии, номере телефона, дате рождения; поиска по полях (в том числе, поиск по подстроке). (использовать регулярные выражения)
- Вывести информацию о клиентах в табличном виде (как в задаче со студентами). Вывод форматировать в зависимости от текущего языка и региональных параметров.
- Реализовать собственное форматирование номера карточки для числа в формате NNNN-NNNN-NNNN-NNNN
- Обеспечить контроль ввода данных с помощью регулярных выражений (тел - +380 11 111 11 11, серия паспорта - AA 111111, и т.д.)
- Заменить неправильные данные, если такие есть в списке клиентов банка, добавленных ранее