Лекция 12.
Работа со строками.
Работа с файлами

Строки в C#

- Строки являются объектами
- Строки являются **неизменяемыми объектами**, т.е. объект строки нельзя изменить после его создания
- Поэтому все функции, которые работают со строками, возвращают новые строки, а старые – остаются без изменения
- Строки можно сравнивать через == и !=, либо через Equals

Документация по строкам

- https://msdn.microsoft.com/ru-ru/library/system.string(v=vs.110).aspx

- Там перечислены все функции строк и их описание

Функции для работы со строками

- int CompareTo(string s)
- Сравнивает строки лексикографически:
  - возвращает 0, если строки равны,
  - положительное число, если данная строка больше переданной
  - отрицательное число – в противном случае
- ```
  string a = “123”;
  if (a.CompareTo(“344”) > 0)
  {
     // код
  }
  ```
  ```

Лексикографическое сравнение строк

- Сравниваем коды первых символов строк. Если один из кодов больше, то это строка больше
- Если коды равны, переходим к проверке следующих символов и т.д.
- Если при этом одна из строк кончилась, то она считается меньше

- Пример верных высказываний:
- “abc” меньше “b”, “ab” > “a”

Contains

- bool Contains(string s)
- Проверяет, входит ли переданная строка в данную строку

- Пример:
- string s = “Monday, 2014”;
- if (s.Contains(“2014”))
  {
  Console.WriteLine(“Есть 2014”);
  }

EndsWith, StartsWith

- bool EndsWith(string s)
- Проверяет, заканчивается ли текущая строка на переданную строку

- bool StartsWith(string s)
- Проверяет, начинается ли текущая строка на переданную строку

IndexOf

- int IndexOf(string s)
- Выдает первый индекс, начиная с которого в текущей строке находится переданная строка. Если переданной строки нет в строке, то выдается -1

- Пример:
- string s = “Of 2014 2014 2014”;
- int index = s.IndexOf(“2014”); // 3

IndexOf

- int IndexOf(string s, int startIndex)
- Аналогично, только ищет, начиная с переданного индекса startIndex

- Пример:

```
string s = “Of 2014 2014 2014”;
int index = s.IndexOf(“2014”, 4);   // 8
```

lastIndexOf

- int LastIndexOf(string s)
- Выдает последний индекс, начиная с которого в текущей строке находится переданная строка. Если переданной строки нет в строке, то выдается -1

- Пример:

```
string s = “Of 2014 2014 2014”;
int index = s.LastIndexOf(“2014”);   // 13
```

Replace

- string Replace(string toSearch, string replacement)

- Заменяет все вхождения первой переданной строки на вторую переданную строку

- “2014 2014 2014”.Replace(“2014”, “2015”)
  // 2015 2015 2015

Split

- string[] Split(string s, StringSplitOptions options)
- Разбивает строку на массив подстрок по указанной строке

- Пример:

```
string[] numbers = “1, 2, 3”.Split(new string[] { “, ” },
    StringSplitOptions.RemoveEmptyEntries);
// массив из элементов “1”, “2” и “3”
```

- Параметр StringSplitOptions имеет 2 возможных варианта – оставлять, либо убирать пустые строки из результата
- Обычно нужно их убирать

ToLower, ToUpper

- string ToLower()
- Переводит новый объект строки, который содержит текущую строку, но в нижнем регистре

- string ToUpper()
- Аналогично, только в верхнем регистре

trim

- string Trim()
- Возвращает новый объект строки, который содержит текущую строку, но в которой обрезаны пробельные символы в начале и конце строки

- Пример:
- “ 123\t ”.Trim()
  // “123”

Substring

- string Substring(int startIndex, int length)
- Возвращает подстроку, начиная с начального индекса startIndex, длина итоговой строки будет length символов
- Можно указать только начальный индекс, тогда подстрока возьмется до конца строки
- Пример: “123 456”.Substring(4, 7); // “456”
- То же самое: “123 456”.Substring(4); // “456”

Преобразование строк в числа

- Convert.ToInt32(string s)
- Convert.ToDouble(string s)

- Пример:
- int a = Convert.ToInt32 (“345”); // 345
- double b = Convert.ToDouble (“3.2”); // 3.2

StringBuilder

- Класс StringBuilder используется для формирования больших строк
- StringBuilder sb = new StringBuilder();
  sb.Append(“Номер квартиры = ”)
  .Append(flatNumber)
  .Append(“, номер подъезда = ”)
  .Append(entranceNumber);

  string result = sb.ToString();
  // получение результирующей строки

Вызовы Append можно составлять в цепочки

Это потому что Append делает в конце return this;

StringBuilder

- Важные методы:
  - Append() – вставка в конец
  - Remove(int startIndex, int length) – удаление заданного количества символов от начального индекса
  - ToString() – преобразование в строку
  - Length – получение длины строки

Задача «StringBuilder»

- Создать строку из чисел от 1 до 100 через запятую при помощи StringBuilder
- Распечатать строку в консоль

Задача «URL»

- Написать программу, которая вычленяет из URL адреса имя сервера. Имеется в виду следующее. Для строки вида http://SomeServerName/abcd/dfdf.htm?dfdf=dfdf вычленить SomeServerName
- Строка может начинаться не обязательно с http, но и с https или чего-то другого. Но :// есть всегда
- Учесть случай, когда после :// больше нет слэша:
- http://SomeServerName
- Использовать IndexOf и Substring

Задача «Число вхождений»

- Прочитать текст из файла, и написать функцию, которая считает количество вхождений некоторой строки в этот текст без учета регистра символов

- Использовать цикл и IndexOf, который принимает начальный индекс, с которого искать

Задача «Разбиение строки»

- Разбить строку “1, 2, 3, 4, 5” и получить массив из этих чисел и найти их сумму

- Использовать Split и Convert.ToInt32

Работа с файлами

Пример файла

- Первое число n – целое, означает количество чисел
- Далее идёт n вещественных чисел

- Пример:

  3 1,3 4,4 5,5

- Хотим прочитать его и положить числа в массив

Чтение файлов

```
using System.IO;
using System;

public class Main
{
   public static void Main()
   {
      // создаем StreamReader, указываем путь к файлу
      StreamReader reader = new StreamReader("input.txt");

      // читаем строку при помощи метода ReadLine
      string line = reader.ReadLine();

      /* дальше нужно при помощи строковых функций вытащить данные */

      // когда мы все прочитали, reader нужно закрыть методом Dispose
      reader.Dispose();
   }
}
```

Закрытие потока

```
public static void Main()
{
    using (StreamReader reader = new StreamReader( “input.txt"))
    {
        // работаем с ридэром
        string x = reader.ReadLine();
    }
}
```

- После того, как работа со reader’ом завершена, его обязательно нужно закрывать, вызвав метод Dispose()
- Лучше всего для потоков использовать конструкцию using, которая закрывает ресурсы при завершении блока

Правильное чтение файлов

```
using System.IO;
using System;

public class Main
{
    public static void Main()
    {
        // создаем StreamReader, указываем путь к файлу
        // файл должен лежать в папке bin/Debug
        using (StreamReader reader = new StreamReader("input.txt"))
        {
            // читаем строку при помощи метода ReadLine
            string line = reader.ReadLine();
            // дальше нужно при помощи строковых функций вытащить данные
        }
    }
}
```

Теперь не нужно вызывать Dispose, using вызовет его сам в конце блока

Разбор данных строки

```
string line = reader.ReadLine();
string[] splits = line.Split(new string[] { " " },
    StringSplitOptions.RemoveEmptyEntries);

int count = Convert.ToInt32(splits[0]);
double[] numbers = new double[count];

for (int i = 1; i <= count; ++i)
{
    numbers[i - 1] = Convert.ToDouble(splits[i]);
}
```

Путь к файлу

- **Абсолютный путь**
  (с полным указанием буквы диска и т.д.):

  F:\Users\Pavel\IdeaProjects\Test\folder\input.txt

- **Относительный путь**
  (относительно корневой папки проекта):

  \folder\input.txt

- Этот путь указывает туда же, что и абсолютный путь в предыдущем примере

Путь к файлу

- **Относительный путь**

  input.txt

- Такой путь означает что файл input.txt лежит в корневой папке проекта

Специальные символы . и ..

- В относительных путях могут использоваться специальные символы . и ..
- Одна точка означает текущую папку
  Т.е. Эквивалентно:
  - input.txt
  - ./input.txt
- Две точки означают родительскую папку
  - ../input.txt // находится в родительской папке

Задача

- Создать строковый файл
- Сохранить в массив строки файла. Массив создать заведомо большей длины
- Вывести содержимое массива на консоль отдельным циклом

Задача

- Возьмите какую-нибудь свою задачу, которая читала данные из сканнера
- Измените программу так, чтобы данные читались из файла

Запись в файл

- ```
  StreamWriter writer = new StreamWriter("output.txt");
  writer.WriteLine("OK!");
  writer.Dispose();
  ```

- Класс StreamWriter имеет те же методы, что Console, т.е. можно использовать Write, WriteLine
- Всё это будет записываться в файл
- Файл с указанным именем будет создан если его нет, либо перезаписан, если файл уже существует
- Как и при чтении, после окончания работы, writer нужно закрыть при помощи метода Dispose
  ```

Правильное закрытие потока

- ```
  public static void Main()
  {
      using (StreamWriter writer = new StreamWriter("output.txt"))
      {
          // что-то пишем во writer
          writer.WriteLine(“OK”);
      }
  }
  ```
- После того, как работа с writer’ом завершена, его обязательно нужно закрывать, вызвав метод Dispose()
- Лучше всего для потоков использовать конструкцию using, которая закрывает ресурсы при завершении блока, автоматически вызывая метод Dispose
  ```

Несколько блоков using

- Блоки using можно вкладывать друг в друга
-
```
public static void Main()
{
    using (StreamWriter writer = new StreamWriter("output.txt"))
    {
        using (StreamReader reader = new StreamReader("in.txt"))
        {
            string currentLine;
            while ((currentLine = reader.ReadLine()) != null)
            {
                writer.WriteLine(currentLine);
            }
        }
    }
}
```

Копирование текстового файла

Задача «Перевод файла в верх.регистр»

- Написать программу, которая читает строки файла, переводит их в верхний регистр и записывает результат во второй файл