Introduction to Programming II

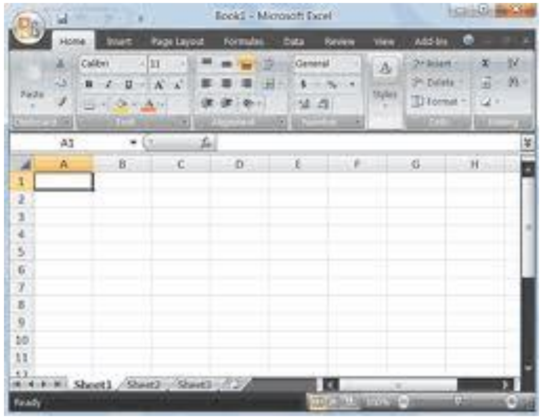
Lecture 8 – Data Representation, Files And Directories

Semester II

Information

- Computers store and manipulate many types of information:

Hello, World!



Data

- But underneath, the computer is an electric device that understands one language only:

Power Off

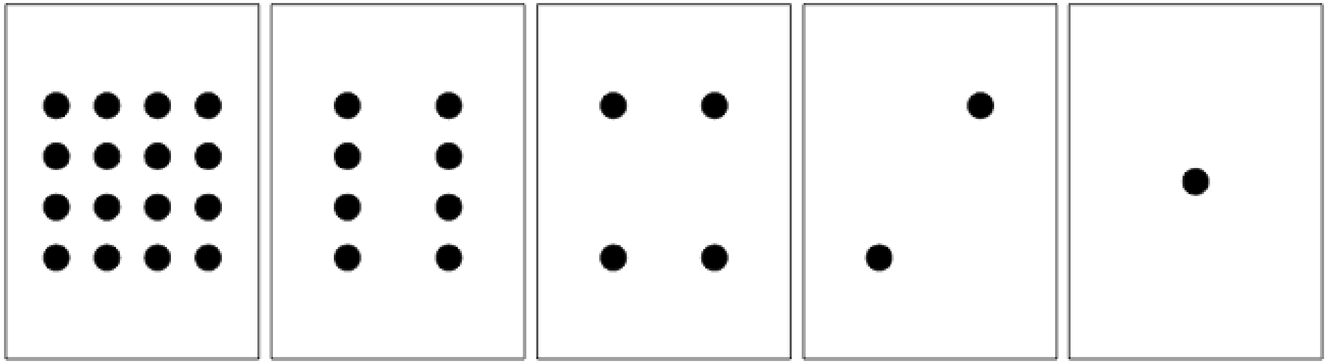
Power On

- We denote the ***power off*** state as 0 and the ***power on*** state as 1

Data

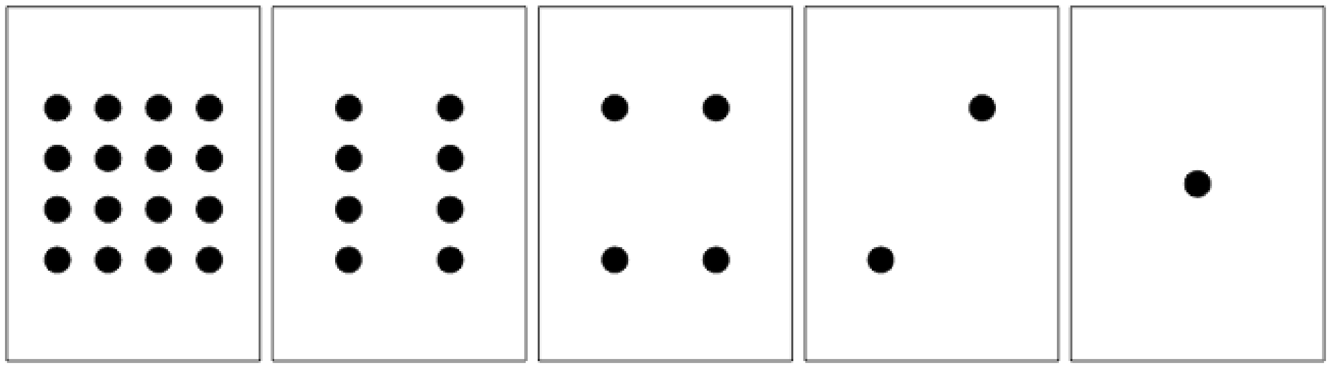
- So computers only understand 0's and 1's
- We need to "translate" our human-oriented information into this (ridiculously simple …) bi-state language
- Let's start with the simplest data: numbers
- How can we represent a positive integer using 0's and 1's?

Binary Numbers



- How can we use these cards to represent, say, 21?
  - You can take up to one card of each card

Binary Numbers

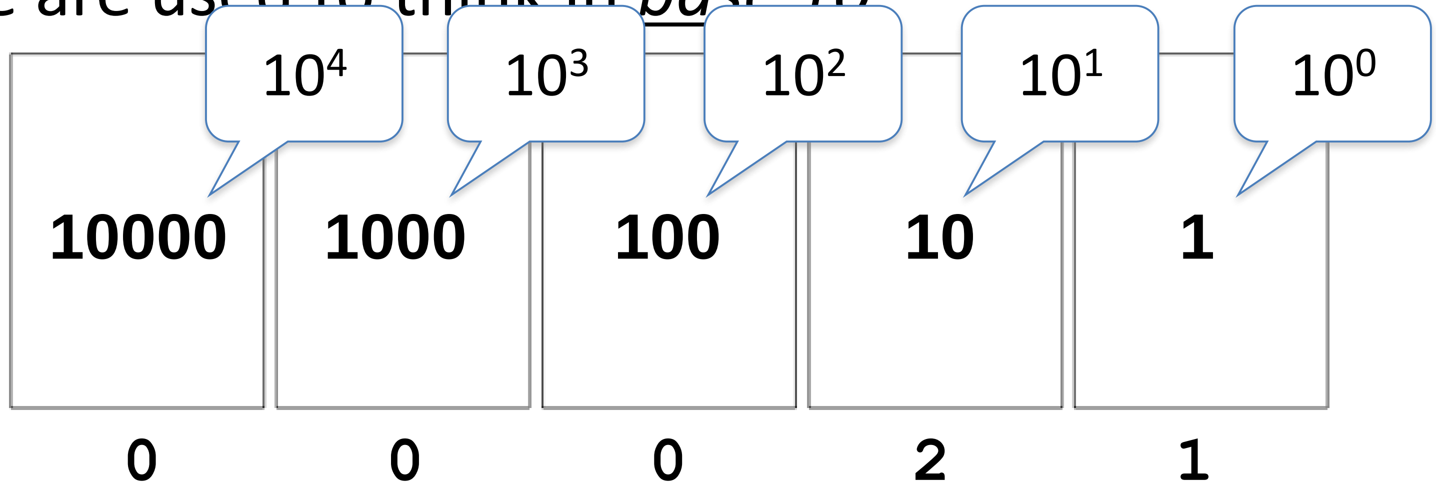


1 0 1 0 1

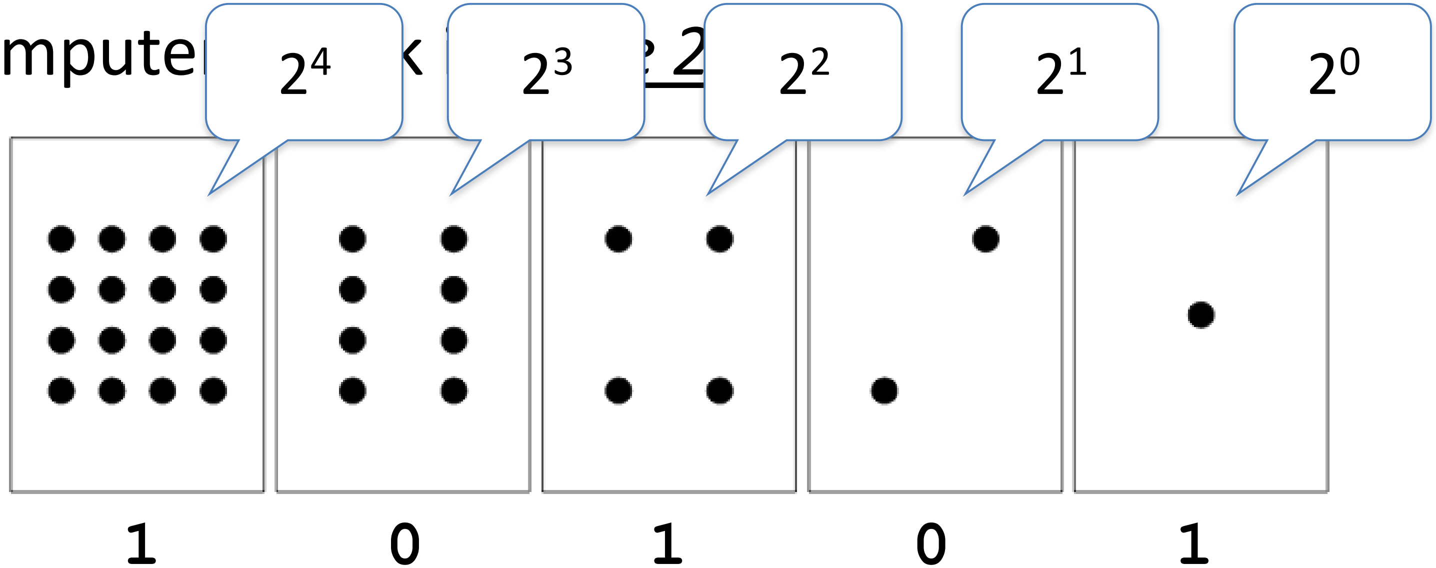
- 21 = 1*16 + 0*8 + 1*4 + 0*2 + 1*1
- We can represent the number 21 with only 0's and 1's, as the number 10101
- Can we represent the number 30? How?

Binary Numbers

- We are used to think in *base 10*:



- Compute[illegible]k [illegible] 2



Binary Numbers

- In base 10, which is also called the ***decimal base***: Each number is a sum of 1's, 10's, 100's, ...
  - 21 is 2*10 + 1*1
  - 308 is 3*100 + 0*10 + 8*1

- When using the ***binary base*** (base 2), we sum powers of 2 (instead of powers of 10): 1, 2, 4, 8, 16, ...
- That's why the number 21 in base 10 becomes 10101 in base 2!

Binary Numbers - Exercise

An algorithm to convert a `number` from base 10 to base 2:

1. `result` = 0
2. `digit` = 0
3. while `number` is positive:
    1. `shift` = 10^{digit}
    2. Add (`number` % 2) * `shift` to `result`
    3. Divide `number` by 2 (integer division)
    4. Increment `digit`
4. return `result`

Write the function decimal2binary(number).

```
public int Decimal2binary(int number)
{
    int result = 0, digit = 0;
    while (number > 0)
    {
        int shift = (int)Math.Pow(10, digit);
        result += (number % 2) * shift;
        number /= 2;
        digit ++;
    }
    return result;
}
```

Representing Data

- Conclusions so far:
  - Computers only understand 0's and 1's
  - We can represent positive integer numbers using 0's and 1's only

- In fact, we can represent any numeric information on a computer – using binary numbers!

1567995227

1011101011101011011010101011011

Representing Text

- Recall: a character (תו) represents a letter, digit, or symbol:

A	B	C	a	b	c	%	?	&
65	66	67	97	98	99	37	63	38

- Each character has a numeric ID. This ID is agreed by all computers in the world!
- When we type characters, they are actually represented as some numbers instead
  - And we already know how to represent numbers…

Bits and Bytes

- A binary digit (ספרה בינארית) is called a ***bit***
- A sequence of *8 bits* is called a *byte*
- 1024 bytes are called a Kilo-Byte (KB), about 1,000 bytes
- 1024 KB are called Mega-Byte (MB), about 1,000,000 bytes
- 1024 MB are called Giga-Byte (GB), about 1,000,000,000 bytes
- And so on…

Bits and Bytes

Marketing example: Internet Service Providers

What does "100 Mega transfer rate“ mean?

What transfer rate will your browser actually show?

Why is there a difference?

Text Representation

ASCII:

- Initially, each text character was represented using *one byte*
  - How many different characters can be represented this way?
  - Do you see any problem with this approach?

Unicode:

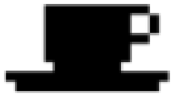
- We want computers around the world to be able to display text in many different languages
- 256 characters are not enough for all languages in the world!
- Unicode uses *two bytes* per character:
  - 65536 different characters!

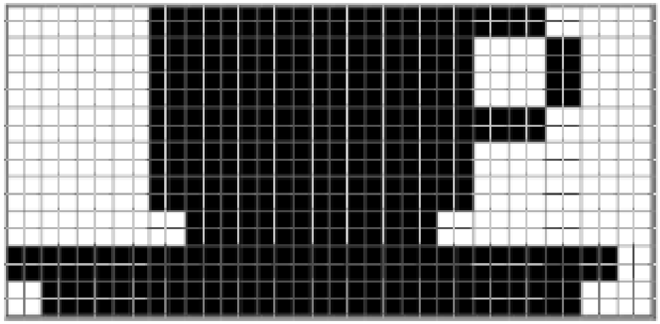
Representing Images

- What about images?
- How would you convert such complex information into 0's and 1's?

- We can divide the image to a grid of many small points, each point having its own color
- In computing terms, each point is called a *pixel*
- New challenge: we should decide how to represent colors.

Representing Images



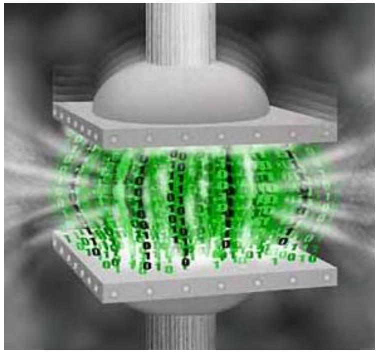


- Let's start with black-white images
- This picture size is 36x18 pixels
- A simple representation of it can be to represent white pixels as 0's and black pixels as 1's
- This will require 36*18=648 bits which are 81 bytes
- We can, instead, save the lengths of same-color sequences:
  - For example, the first row contains 8 white pixels, then 22 black pixels, then 6 white pixels
  - We can represent this row as 8,22,6 (each number is a byte)
  - The whole image will take only 60 bytes – 26% less!

Representing Images

Some more ideas to reduce the data size:

- omitting the last sequence length and calculating it knowing the row length
- counting it as one long row and then arrange it as a table. This will reduce the ending and starting white (or black) pixels to be one number instead of two numbers…
- reducing the size of a number to hold 5 bits instead of a byte because the line is < 63
- Or…



Data Compression

- We have reduced the size of the image file by 26%
- But why would we do that?
  - Digital cameras: smaller images imply less storage space and cheaper memory cards
  - Internet servers: (think of Facebook or Picasa) – smaller images imply less servers, less disks
  - Internet traffic: billions of images are being sent and downloaded over the internet, each day. Every bit we save means a faster internet to all!

facebook Picasa flickr

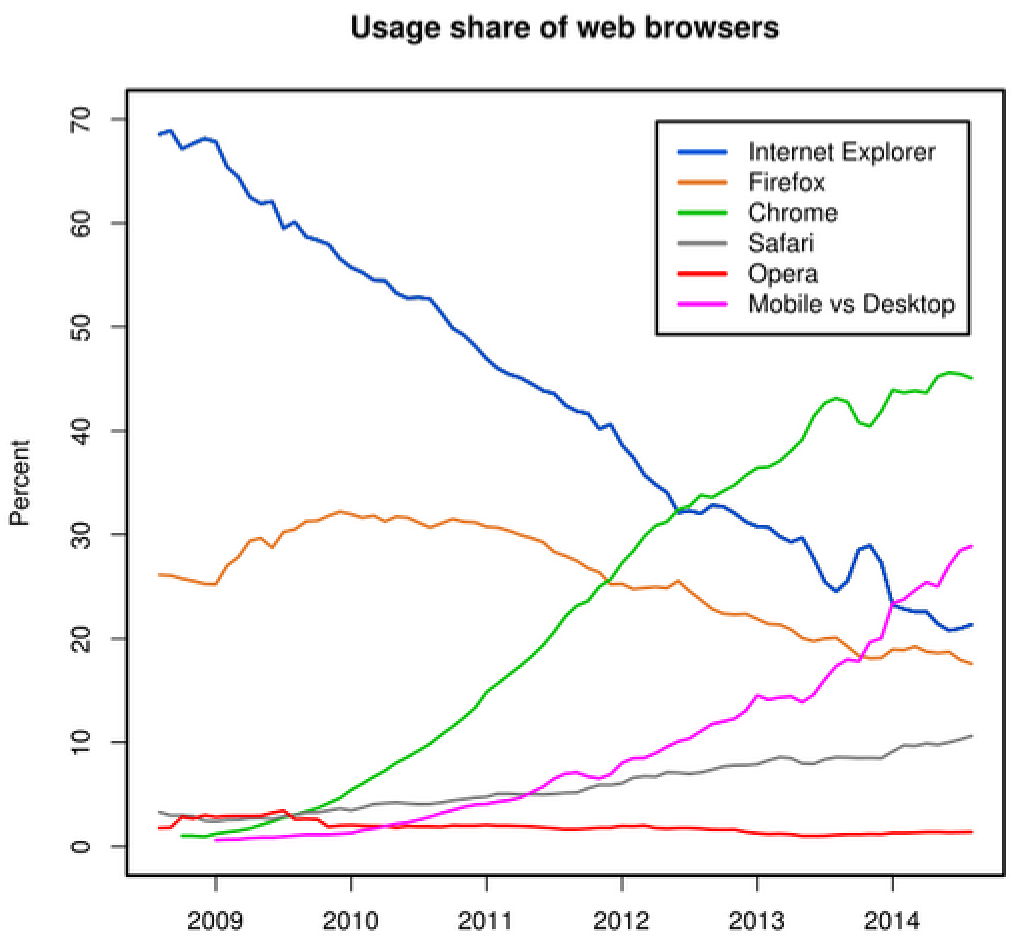
Data Compression

- Today, data is routinely compressed :
  - Images (.jpg, .gif, …)
  - Movies (DVD, downloads, you-tube, …)
  - Sound files (MP3)
  - Documents (Zip)
  - Web pages (Zip, Google Chrome)

Google Chrome

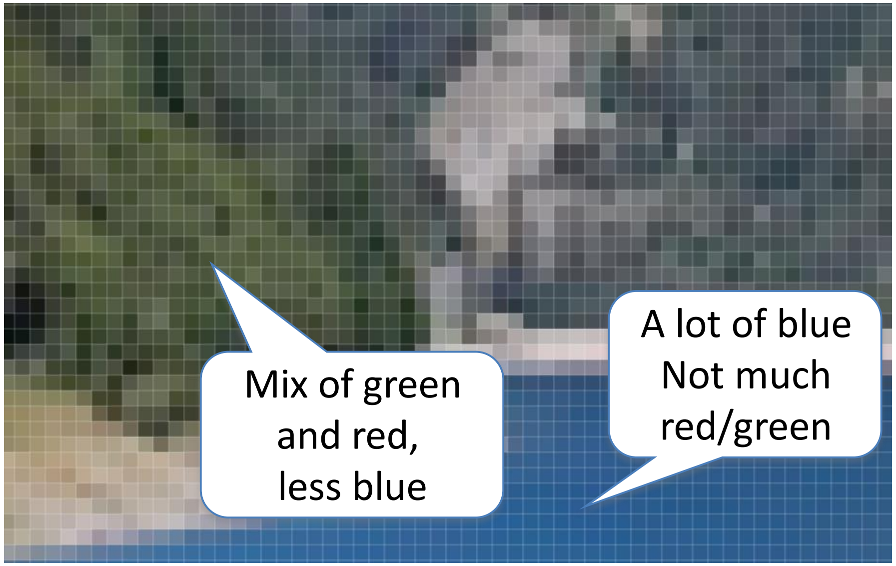
- Data compression is a very practical research area – less time / less space imply great savings.

Data Compression



Representing Color Images

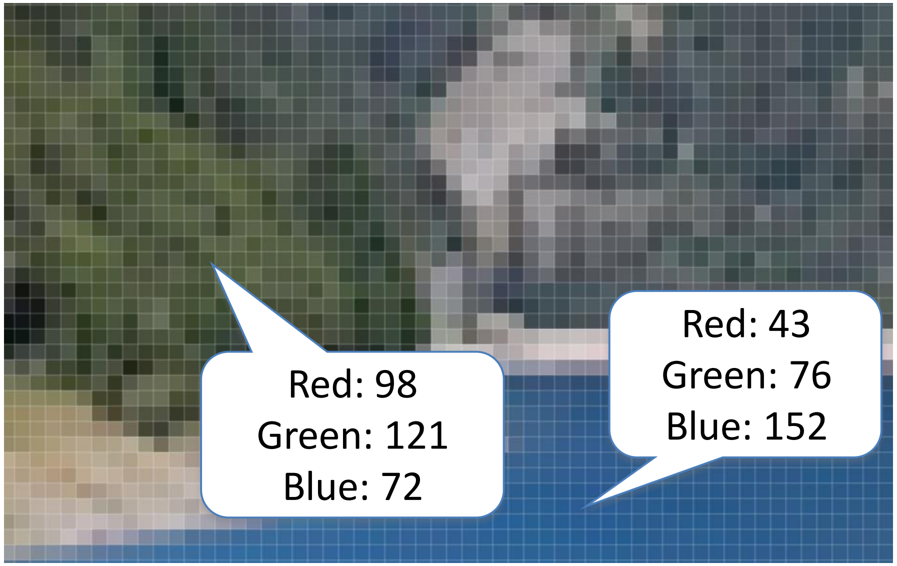
- We can represent each color using the three basic eye-visible colors: Red, Green and Blue
- If we mix some red, some green and some blue – we get a new color



- For each pixel, we need to decide in which *proportions* we wish to mix these colors
  - *Proportions* are numbers. We know how to represent numbers.
  - We use three different numbers to represent the color of each pixel.
- All we need to do is represent all pixels in an image…

Representing Color Images

- We use numbers in the range 0 – 255 to represent each color channel (red/green/blue) (why?)
- Each pixel is represented using 3 bytes (=24 bits)
  - Color image is 24 times larger than a black/white image!



- There are compression techniques for color images as well
  - BMP
  - JPEG
  - GIF
  - PNG
  - And more…

Representing Movies

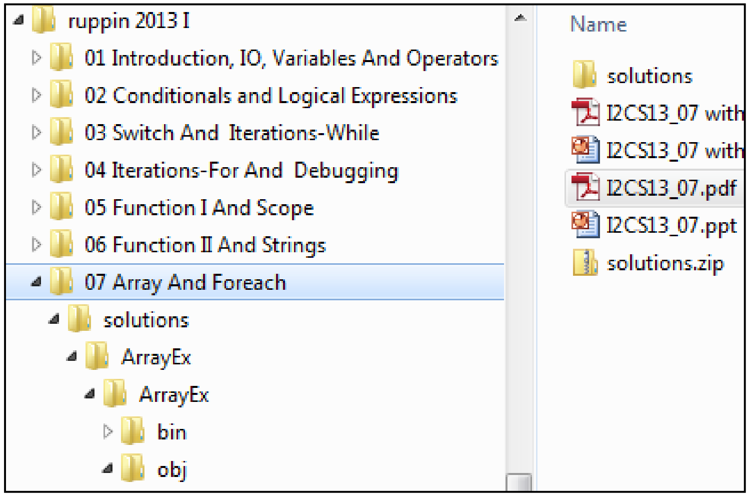
- Movies contain:
  - ~30 color images per second
  - Sound channels
- We can represent a movie as a series of (many) images, and additional sound channels
- However, the amount of images is enormous!
  - 10 seconds movie is about 300 images
  - 1 hour movie is 108,000 images!
- Can you think of a way to compress image data in a movie?

Files

- A file is a sequence of bytes that represent some data
- A file can represent a text document, an image, a song, a piece of C# code, etc.
- A file can be stored on the computer's hard drive
  - Then we can load it again later
  - Or send it by e-mail
  - Or save it to a CD / USB memory
  - Or upload it to the course website.

File System

- Files are usually organized in an hierarchical structure of directories (folders)
  - We call this structure the "directory tree"
- To access a file we need to know:
  - Its name
  - Its location
- The location of a file is described as a *path* in the directory tree:
  - In Windows: C:\Users\me\My Documents\myfile.txt
  - In Unix: /Users/me/Documents/myfile.txt



Opening a File

- In order to access a file, we should first open it
  - This actually means asking the Operating System for a permission to access the file
- Then, we have a ***file object*** with which we can work
- We can open a file for *reading* or for *writing*.
- In C# we have two classes that will already deal with the file opening for us: StreamReader, StreamWriter
- Need: using System.IO;

Reading a File

- Suppose we have a file named `file.txt`, stored at `C:\temp`
- The content of the file is:

```
This is a file
with some data in it
```

- We would like to read this content:

```
StreamReader sr = new
StreamReader(@"c:\temp\file.txt");
string firstline = sr.ReadLine();
string strInput = sr.ReadToEnd();
sr.Close();
```

Create a stream to 'C:\temp\file.txt' for reading. @ - ignore special characters

Reads the next line as a single string

Reads the rest of the file as a single string

Closes and releases the file **(Very important!!)**

Reading a File

- `file.read()`
  Reads one char as an int
- `file.Peek()`
  Reads and returns the next char but does not move the cursor. -1 if there is nothing left to read
- `StremReader.EndOfStream`
  Returns a boolean if we reached the end of the file

Reading a File

- We can also use the for loop to iterate on the lines of the file:

```
StreamReader sr = new StreamReader(@"c:\temp\file.txt");
while (!sr.EndOfStream)
{
    label3.Text += sr.ReadLine().TrimEnd() + "\n";
}

sr.Close(); //if we comment this, after running this function
            //we could not change the file.txt
            //from windows because it is still used by this
process
```

Removes whitespaces and line break characters from the end of line

- When reading large files, it is sometimes better to read the file line-by-line, and not read the whole file at once. (pros and cons, memory vs time)

Writing a File

- Suppose we have a file named `mydata.txt`, stored at `C:\temp`
- The content we want to write into it:

```
This is my data
that i need to store
```

- We would like to save the data:

```
string[] arr = new string[] {"This is my data",
                             "that i need to store"};

StreamWriter sw = new StreamWriter(@"c:\temp\mydata.txt");
//will create the file if not existed

for (int i = 0; i < arr.Length; i++)
{
    sw.Write(arr[i] + "\r\n");
    //sw.WriteLine(arr[i]); //the same as above
}
sw.Close();
```

Crate a stream to 'C:\temp\file.txt' for writing.
@ - ignore special characters

Writes the string into the file. \r\n instead of only \n

Closes and releases the file **(Very important!!)**

File

- File – this class has some function to handle files:
  - File.Exists(filePath) – boolean, if exists
  - File.Delete(filePath)
  - StreamWriter sw = File.CreateText("myFile.txt")

    sw.Close(); //the File.CreateText returns sw so if not stored and then closed we will get an error when trying to delete the file or dir!!!
  - File.AppendText(filePath) - append
  - File.OpenText("myFile.txt") – for reading
  - File.Move(srcfilePath, destfilePath) – change place
- filePath – relative to the exe file
- “\r\n” – new line in files

Directory

- Directory
  - Directory.Exists(*directoryPath*)
  - Directory.CreateDirectory(*path*)
  - Directory.Delete(*path*,recursive=t*rue/false) – recursive=subdir and files*
  - Directory.Move*(source path, destination path)*
  - Directory.GetDirectories(*path*) – returns a string[], the names of the subdirs
  - Directory.GetFiles(*path* [, *search pattern*]) – returns a string[], the names of the files
  - Directory.GetFileSystemEntries(*path*) – files and dirs
  - Directory.GetCurrentDirectory();

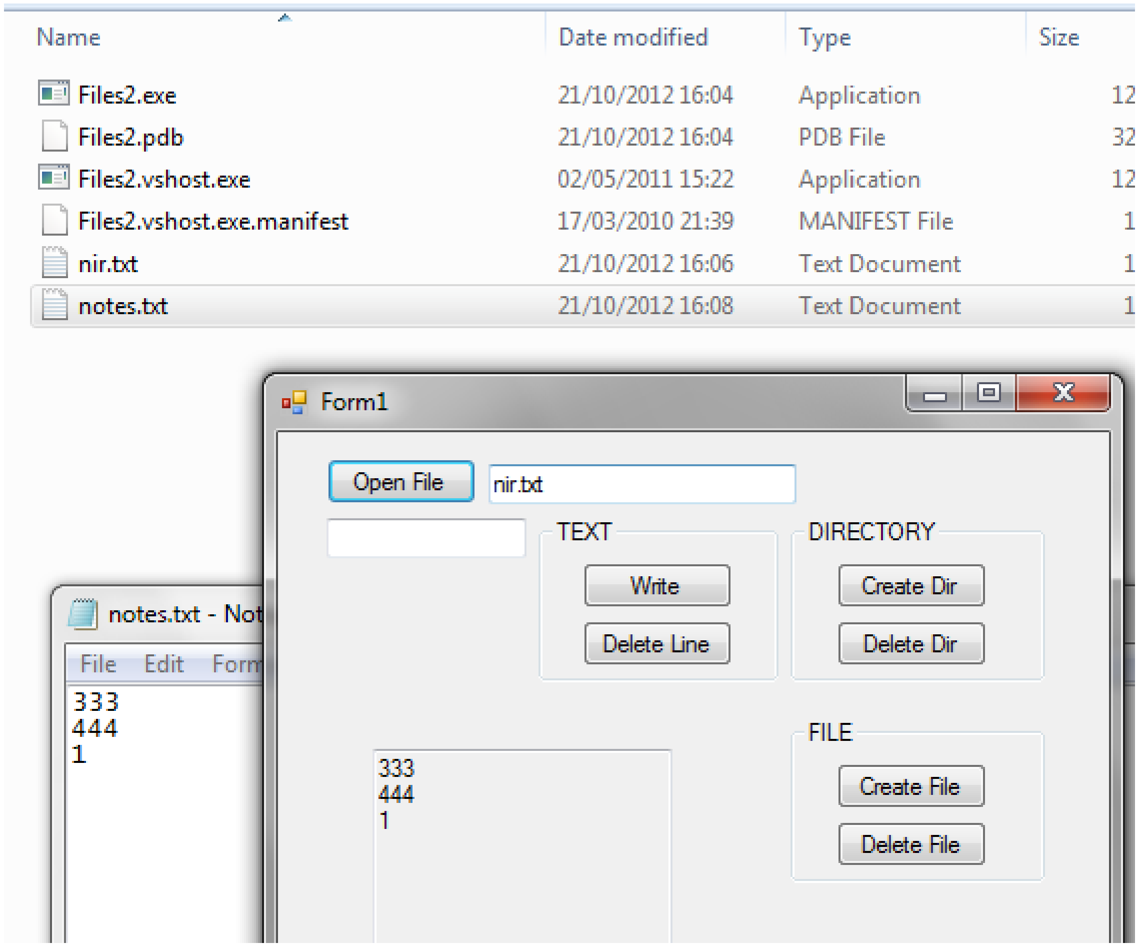
** directoryPath - relative to the exe file*

Extras

- “\r\n” – new line in window application

  Remember it takes 2 chars\places!!!
- try to work with one open stream, to avoid the time consuming opening\ closing commands
- (Each class will load\save itself)
- openFileDialog – to open a local file
  - openFileDialog1.ShowDialog();
- FileInfo – get information about a file
  - FileInfo fi = new FileInfo(fullName);
  - Name
  - Extension
  - CreationTime
  - DirectoryName

Exercise


Name
Date modified
Type
Size
Files2.exe
21/10/2012 16:04
Application
12
Files2.pdb
21/10/2012 16:04
PDB File
32
Files2.vshost.exe
02/05/2011 15:22
Application
12
Files2.vshost.exe.manifest
17/03/2010 21:39
MANIFEST File
1
nir.txt
21/10/2012 16:06
Text Document
1
notes.txt
21/10/2012 16:08
Text Document
1
notes.txt - Not
File
Edit
Form
333
444
1
Form1
Open File
nir.txt
TEXT
Write
Delete Line
DIRECTORY
Create Dir
Delete Dir
FILE
Create File
Delete File
333
444
1