

C#.Net Manual 22 March 2019

Table of Contents

| | |
|--|-----------|
| 1. Cheat Sheet for C#.net | 5 |
| Converting Variables – Automatic and Manual | 7 |
| 1. Software Installation | 10 |
| Server Management Studio | 10 |
| Installing Visual Studio 2017 /2019 | 11 |
| Set up the Text Editor for Word Wrap and Line Numbers | 12 |
| Visual Studio Extensions | 13 |
| Viasfora | 14 |
| Keyboard Shortcuts | 15 |
| 2. Things we need on the Course | 16 |
| Dropbox | 16 |
| Microsoft Virtual Academy | 16 |
| New Microsoft Learn website | 16 |
| Watch this introduction to C# | 16 |
| GitHub | 16 |
| Why Learning is Your Job | 18 |
| Soft Skills | 19 |
| About Time – The Pomodoro Method | 21 |
| Being Productive | 21 |
| How to be an Excellent Programmer | 22 |
| How to become a successful programmer - from Reddit | 22 |
| C# Programming Yellow Book – good starter | 23 |
| Clean Code – Planning for the ‘stupid you’ | 24 |
| Some coding ideas for new programmers | 25 |
| How programmers precode, or prepare for coding. | 27 |
| 3. What does all the code layout mean? | 28 |
| Operators | 31 |
| Variables | 32 |
| Data Types | 32 |
| The Ternary Operator ? : (IF shortcut) | 34 |
| How to declare a variable | 35 |
| Renaming your code automatically - Refactoring | 36 |
| 4. Introduction to C# | 37 |
| Hello World Lesson | 37 |
| Exercises | 46 |
| 5. Conditionals - If Statements | 47 |
| Grades Project using IF | 48 |
| Finding out the Average of Grades entered | 55 |
| C# - Logical Operators | 57 |
| IF with AND and OR – Student grades | 58 |
| Calculator Project- | 60 |
| 6. Types of Methods | 63 |
| Methods that don't return data | 64 |
| Methods that takes data in | 64 |
| Methods that return data – Pure Methods | 65 |
| New C#7 Methods within Methods (Local Functions). | 69 |
| Named Parameters in your methods | 70 |
| Returning more than one value from a method | 71 |
| Using a Class to return multiple data. | 71 |
| Using a Tuple to return multiple data. | 72 |

| | |
|--|------------------------------|
| 7. Unit Testing – Test Driven Development (TDD) | 74 |
| TradeMe tells us about their use of Unit Tests. | 76 |
| Unit Testing the Calculator | 80 |
| Temp Converter - Pure Methods and Unit Tests | 85 |
| Body Mass Index (BMI) exercise | 87 |
| Skanky Old Bus exercise | 88 |
| 8. Debugging your program | 89 |
| Visual Studio 2019 Tools | 92 |
| Enable and Disable Just My Code | 95 |
| 9. Adding images to a Resource file | 96 |
| 10. Conditionals - Loops | 99 |
| While Loop (Test at the beginning) | 99 |
| The Do and Do While Loops | 100 |
| Clear out multiple spaces between words challenge | 100 |
| Loops Exercises | 101 |
| Do While – Fibonacci Sequence | 102 |
| Find the even Fibonacci numbers only | 105 |
| What is the first Fibonacci number to contain 1000 digits? | 105 |
| Dice rolls using Do While loop | 106 |
| Exercise 1: Can you make this with 3 dice? | 107 |
| Exercise 2 How many times does the program run before it gets a triple. | 108 |
| Exercise 3 Tell the program to run until it gets a triple number to stop on. | 109 |
| Exercise 4 How to select different sided dice. | Error! Bookmark not defined. |
| Guess a number program | 111 |
| Calculate Pi | 112 |
| For Loop | 113 |
| Cycling through colours | 114 |
| For Each loop | 115 |
| Rock Paper Scissors | 117 |
| Can you sum the numbers from 1 to n? | 121 |
| Find the multiples of 3 and 5 | 122 |
| The FizzBuzz Project – Standard interview Question | 122 |
| 11. Source Control | 124 |
| Installing Git | 126 |
| Git on your Visual Studio – Review History | 126 |
| GitHub Installation | 127 |
| 12. Arrays | 128 |
| Common Mistakes in Arrays | 130 |
| Array with a For loop | 131 |
| Array with For Each Loop | 133 |
| Create a random number generator that holds the number of the people in the array. | 133 |
| Char Array (Character Array) Find Vowels | 135 |
| Char Array Palindrome | 138 |
| Unit Testing the Palindrome Code | 140 |
| Array Exercises and Problems – Include Job interview Q's | 142 |
| Change calculator using Arrays and Loops | 145 |
| Lotto number Generator and Checker – Shuffle function | 148 |
| C#7 Using Tuples to return data from Array | 151 |
| 13. Multidimensional Arrays | 152 |
| Two dimensional array - Times table program | 153 |
| Tic Tac Toe game – 2 Dimensional Array of Pictureboxes | 155 |
| 14. String Operations | 162 |
| The String.Format and String Interpolation Methods | 166 |
| String Exercises - Do this | 171 |
| 15. Generic Collections | 173 |

| | |
|---|------------|
| List<T> Exercise | 173 |
| Using a SortedList Exercise (Key, Value) | 175 |
| Generics - Dictionaries | 179 |
| Use a real Dictionary in a Dictionary | 180 |
| Oxford English Dictionary | 182 |
| Count how many letters are in a sentence | 183 |
| Enumerators – under the hood theory | 185 |
| 16. Multiple forms and passing data using a dialog box | 187 |
| Passing data with a Public Method | 190 |
| 17. Opening files and saving with File | 191 |
| Choosing a File Mode. | 191 |
| Address Book – 2D Array with file loading | 197 |
| Part 2: Save, Delete, Save changes exercise | 201 |
| StreamReader and StreamWriter | 202 |
| Using StreamReader to load files | 206 |
| The Using Statement – Make sure you are using ‘Using’ | 208 |
| StreamWriter to save a list to a file. | 209 |
| Open a text file with StreamReader | 210 |
| Using Async to load files. | 212 |
| StreamReader and Sender Exercise | 213 |
| Open File Dialog | 216 |
| Open, Select, Sort and Save a list exercise | 218 |
| File and Directory Information Program | 220 |
| Tip of the day program | 221 |
| 18. Introduction to Classes | 222 |
| The Object Orientated Toaster ... | 222 |
| Properties and their Generation | 224 |
| How to create Class Diagrams and Code Map | 227 |
| New C# Property features in C# 6 | 228 |
| Calculator with Class – Intro to classes | 229 |
| 19. ADO.Net and SQL Database Creation | 233 |
| Watch these video series | 233 |
| Learn SQL | 233 |
| TSQL Cheat Sheet | 234 |
| 20. Using SQL Server Management Studio | 236 |
| Setting up SQL Server Management Studio | 236 |
| How to Attach an existing Database | 239 |
| How to Remove a Database from SQLSMS | 240 |
| How to Run a Query in SQL Express Management Studio | 241 |
| Set the ID field to be Auto Incrementing. | 242 |
| How to Run an SQL Script to create a Database (Reference) | 244 |
| Why Can't SQL Server Management Studio Access The “My Documents” Folder | 245 |
| 21. The Golf Project using ADO.net | 246 |
| How to find your Connection String. | 246 |
| Display data in a ListView | 247 |
| Create the Headings for the Listview. | 248 |
| The Code for Golf | 249 |
| Using the ADO.NET DataReader | 249 |
| Returning a Scalar (single) value | 252 |
| Insert /New Command | 253 |
| Using Parameters | 253 |
| Update Command | 256 |
| To modify data within a database | 256 |
| Delete Command | 257 |
| Stored Procedure - The SelectCommand Property | 258 |
| Adding a DataGridView | 262 |
| Click on a DGV row and fill the text boxes | 264 |

| | |
|--|------------|
| Golf Exercise using SQL | 266 |
| 22. Presentation, Business, and Data Layers / Tiers | 267 |
| Create a Data Layer for your Golf Project | 268 |
| Three ways of moving Data to the Data layer from the Form | 268 |
| Returning data to the Form | 270 |
| Here is the code for the first part of the Data Class. | 270 |
| 23. Music Database - Relational Database | 275 |
| Primary Key / Foreign Key relationships | 275 |
| Table structure to date. | 276 |
| Table Data | 277 |
| Creating Database Views – Queries | 280 |
| 24. Music Database in Visual Studio | 284 |
| The Owners DataGridView and Class | 285 |
| Fill the ListBox from the Database | 287 |
| Using Views in the Form | 289 |
| Linking DataGridViews to each other | 290 |
| The Owner DataGridView Click Event | 291 |
| Fill the DataGridView With the Owner Click | 292 |
| DGV Owner Cell Content Click event | 293 |
| Create the CD and Tracks DataGridView Click events. | 295 |
| Update and Delete Queries | 296 |
| 25. Preventing SQL Injections | 299 |
| Little Bobby Tables | 300 |
| How can we stop SQL Injection it in our projects? | 301 |
| SQL Injection Cheat Sheet | 301 |
| We can even hack into Card controlled doors via the blinking LED with Injection. | 301 |
| Creating a Database Unit Test | 302 |

1.Cheat Sheet for C#.net

Working with Numbers

```
int i = 0; // convert from a string
int i = int.Parse("1"); // convert
from a string and don't throw
exceptions

if (int.TryParse("1", out i)) { }
i++; // increment by one
i--; // decrement by one
i += 10; // add 10
i -= 10; // subtract 10
i *= 10; // multiply by 10
i /= 10; // divide by 10
i = checked(i * 2); // check for
overflow
i = unchecked(i * 2); // ignore
overflow
```

Dates and Times

```
DateTime now = DateTime.Now; // date and
time
DateTime today = DateTime.Today; // just
the date
DateTime tomorrow = today.AddDays(1);
DateTime yesterday = today.AddDays(-1);
TimeSpan time = tomorrow - today;
int days = time.Days;
int hours = time.Hours;
int minutes = time.Minutes;
int seconds = time.Seconds;
int milliseconds = time.Milliseconds;
time += new TimeSpan(days, hours, minutes,
seconds, milliseconds);
```

Compare Operators

```
if (i == 0) // equal
if (i != 0) // not equal
if (i <= 0) // less than or equal
if (i >= 0) // greater than or equal
```

```
if (i > 0) // greater than
if (i < 0) // less than
if (o is MyClass) // check type of
object
if (o == null) // check if reference is
null
```

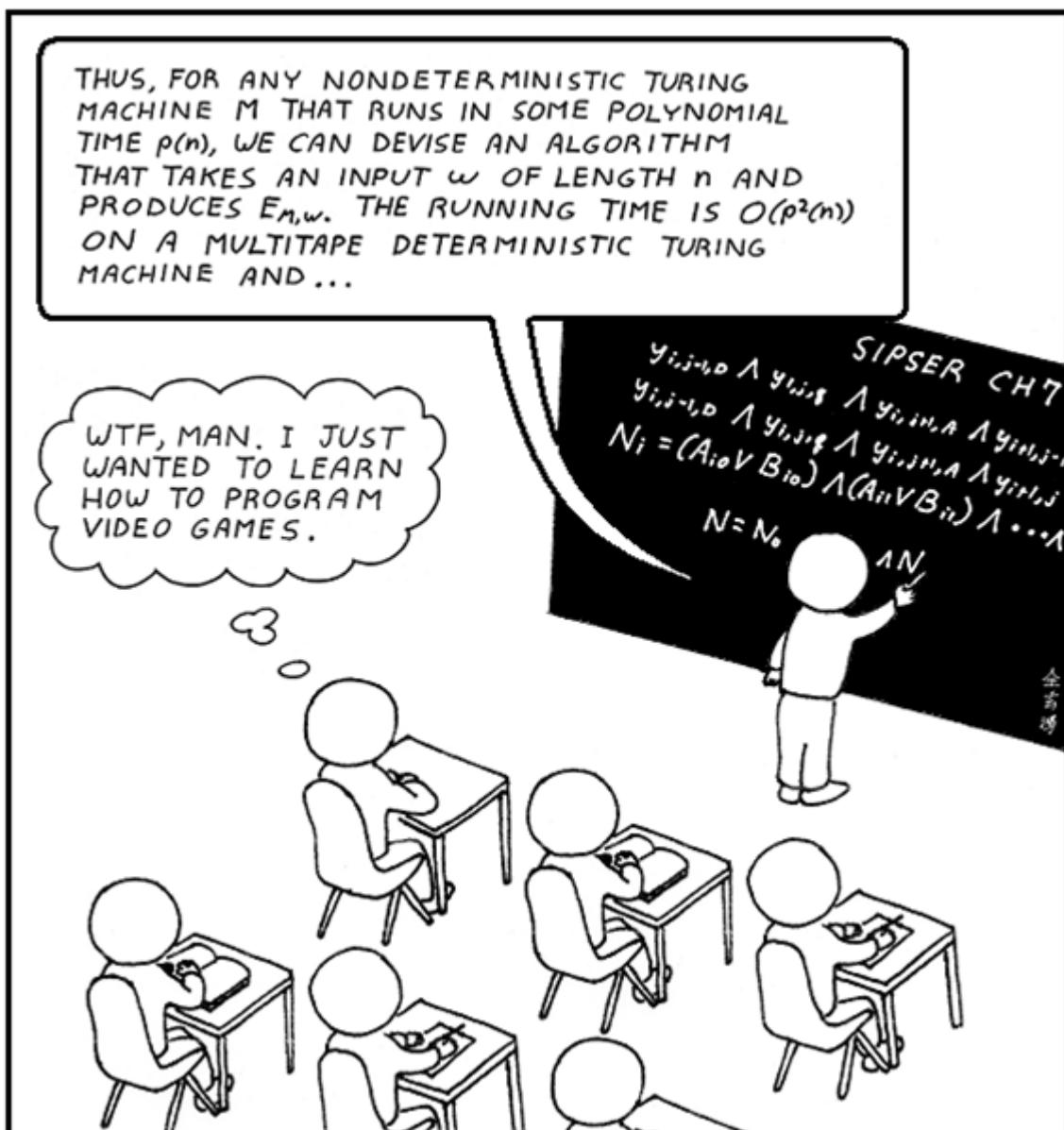
Working with Text

```
string name = "David";
string hello1 = "Hello " + name;
string hello2 = string.Format("Hello
{0}", name);
Console.WriteLine("Hello {0}", name);
StringBuilder sb = new
StringBuilder("Hello ");
sb.AppendLine(name);
sb.AppendFormat("Goodbye {0}", name);
Console.WriteLine(sb.ToString());
// Create a text file
using (StreamWriter w =
File.CreateText(@"c:\names.txt"))
{
w.WriteLine("Bob");
w.WriteLine("David");
}
// Read from a text file
using (StreamReader r =
File.OpenText(@"c:\names.txt"))
{
while (!r.EndOfStream)
{
Console.WriteLine(r.ReadLine());
}
foreach (string s in
File.ReadAllLines(@"c:\names.txt"))
{
Console.WriteLine(s);
}
```

THUS, FOR ANY NONDETERMINISTIC TURING MACHINE M THAT RUNS IN SOME POLYNOMIAL TIME $p(n)$, WE CAN DEVISE AN ALGORITHM THAT TAKES AN INPUT w OF LENGTH n AND PRODUCES $E_{n,w}$. THE RUNNING TIME IS $O(p^2(n))$ ON A MULTITAPE DETERMINISTIC TURING MACHINE AND ...

WTF, MAN. I JUST
WANTED TO LEARN
HOW TO PROGRAM
VIDEO GAMES.

SIPSER CH7
 $y_{i,j=0} \wedge y_{i,j=1} \wedge y_{i,j=2} \wedge y_{i,j=3} \wedge y_{i,j=4}$
 $y_{i,j=0} \wedge y_{i,j=1} \wedge y_{i,j=2} \wedge y_{i,j=3} \wedge y_{i,j=4}$
 $N_i = (A_{i,0} \vee B_{i,0}) \wedge (A_{i,1} \vee B_{i,1}) \wedge \dots \wedge$
 $N = N_0 \wedge N_1 \wedge \dots \wedge N_n$



Converting Variables – Automatic and Manual

C# is a strongly typed language. This means that when a variable is defined we have to specify what type of data the variable will hold. After a variable is declared, it cannot be declared again or used to store values of another type unless that type is convertible to the variable's type.

For example, there is no conversion from an integer to a string. Therefore, after you declare `i` as an integer, you cannot assign the string "Hello" to it.

Automatic Conversions

These variable types can be **automatically converted**, meaning you don't have to add any extra code.

From To

[sbyte](#) short, int, long, float, double, or decimal

[byte](#) short, ushort, int, uint, long, ulong, float, double, or decimal

[short](#) int, long, float, double, or decimal

[ushort](#) int, uint, long, ulong, float, double, or decimal

[int](#) long, float, double, or decimal

[uint](#) long, ulong, float, double, or decimal

[long](#) float, double, or decimal

[char](#) ushort, int, uint, long, ulong, float, double, or decimal

[float](#) Double

[ulong](#) float, double, or decimal

Manual conversions and solving the error messages

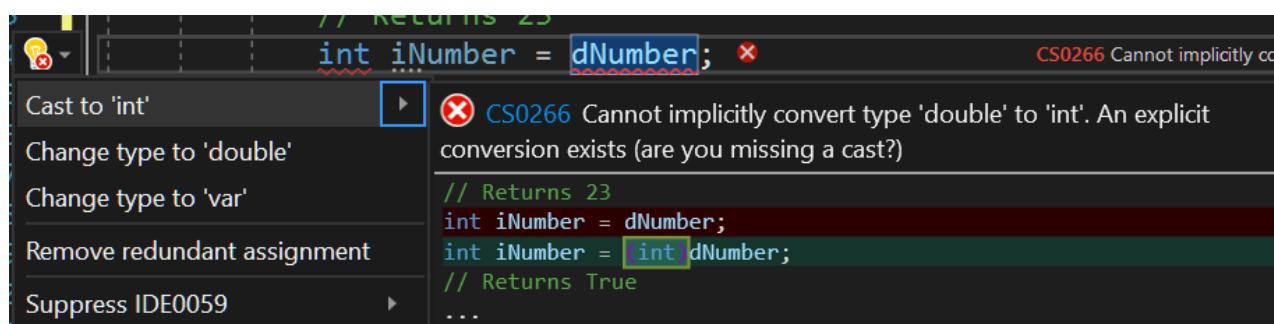
These must be manually converted – this becomes very important as you get an error that comes up if it can't convert automatically.

```
double dNumber = 23.15; ⓘ
// Returns 23
int iNumber = dNumber; ✘
// Returns T💡
bool bNumber = Sy
// Returns "23.15"
string strNumber
// Returns '2'
char chrNumber = Show potential fixes (Ctrl+.)
```

RCS1118 Mark

CS0266 Cannot implicitly convert type 'double' to 'int'. An explicit conversion exists (are you missing a cast?)
 ⓘ (local variable) double dNumber
 Cannot implicitly convert type 'double' to 'int'. An explicit conversion exists (are you missing a cast?)
 Cannot convert source type 'double' to target type 'int'

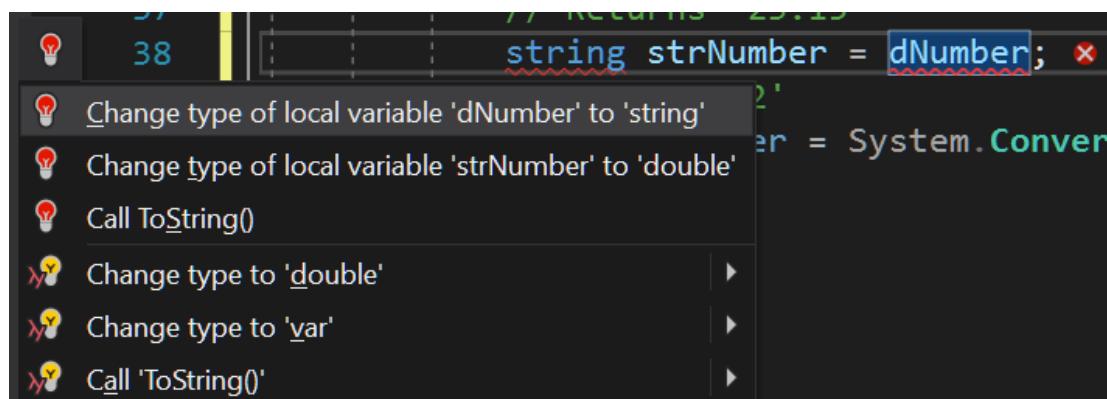
The solution is to click on the lightbulb and choose to convert it.



AND learn how to convert it manually.

```
double dNumber = 23.15;
// Returns 23
int iNumber = System.Convert.ToInt32(dNumber); //Or the new way is
int iNumber = (int)dNumber;
// Returns True
bool bNumber = System.Convert.ToBoolean(dNumber);
// Returns "23.15"
string strNumber = System.Convert.ToString(dNumber);
string strNumber = dNumber.ToString();
// Returns '2'
char chrNumber = System.Convert.ToChar(strNumber[0]);
```

Note with the String conversion all the options you get



Use NEW to set default values

You can create your variables with the NEW constructor and they will automatically set a default value, this is handy if you don't know what the default values are.

```
int i = new int(); //sets the value of i at 0
DateTime dt = new DateTime(); // sets dt to 1/1/0001 12:00:00 am
Boolean b = new bool(); //set to false
```

Using Var variables

Variables are normally explicitly typed, you tell the program if it's an int, bool, string etc. But using the var keyword you can implicitly type variables.

```
var i = 10; // implicitly typed
int i = 10; //explicitly typed
var myint = 0;
var mybool = true;
var mystring = "yet another string";
```

In these cases the compiler infers from the initially assigned value that myint is an integer, mystring is a string and mybool is a Boolean.

Var is used with LINQ queries; however, you often see it in programming and need to be able to recognise it.

1. Software Installation

Server Management Studio

We use this to be able to create, modify and mess around with databases.

There are 2 options, the older guaranteed to work version and the newest version.

Click the link [Here for the older version](#)

Scroll down and click on this file **ExpressAndTools 32BIT\SQLEXPRWT_x86_ENU.exe**

Choose the download that you want

- | <input type="checkbox"/> | File Name |
|-------------------------------------|---|
| <input type="checkbox"/> | ExpressAdv 64BIT\SQLEXPRADV_x64_ENU.exe |
| <input checked="" type="checkbox"/> | ExpressAndTools 32BIT\SQLEXPRWT_x86_ENU.exe |

However I have just downloaded V18 from [here](#) and it works great so do this if you want to live on the edge.

SSMS 18.0 (preview 6)

SSMS 18.0 Public Preview 6 is now available, and is the latest generation of *SQL Server Management Studio* that provides support for SQL Server 2019 preview!



[Download SQL Server Management Studio 18.0 \(preview 6\)](#)



Installing Visual Studio 2017 /2019

You can use either version 2017 or 2019. I like 2019 and it's the same for installation. Get it [here](#)

In the VS Installer click Modify and make sure that these are installed

 Visual Studio Enterprise 2019 Preview
16.0.0 Preview 3.0
Microsoft DevOps solution for productivity and coordination across teams of any size
[Release notes](#)

[Modify](#)
[Launch](#)
[More ▾](#)

Install these three modules.

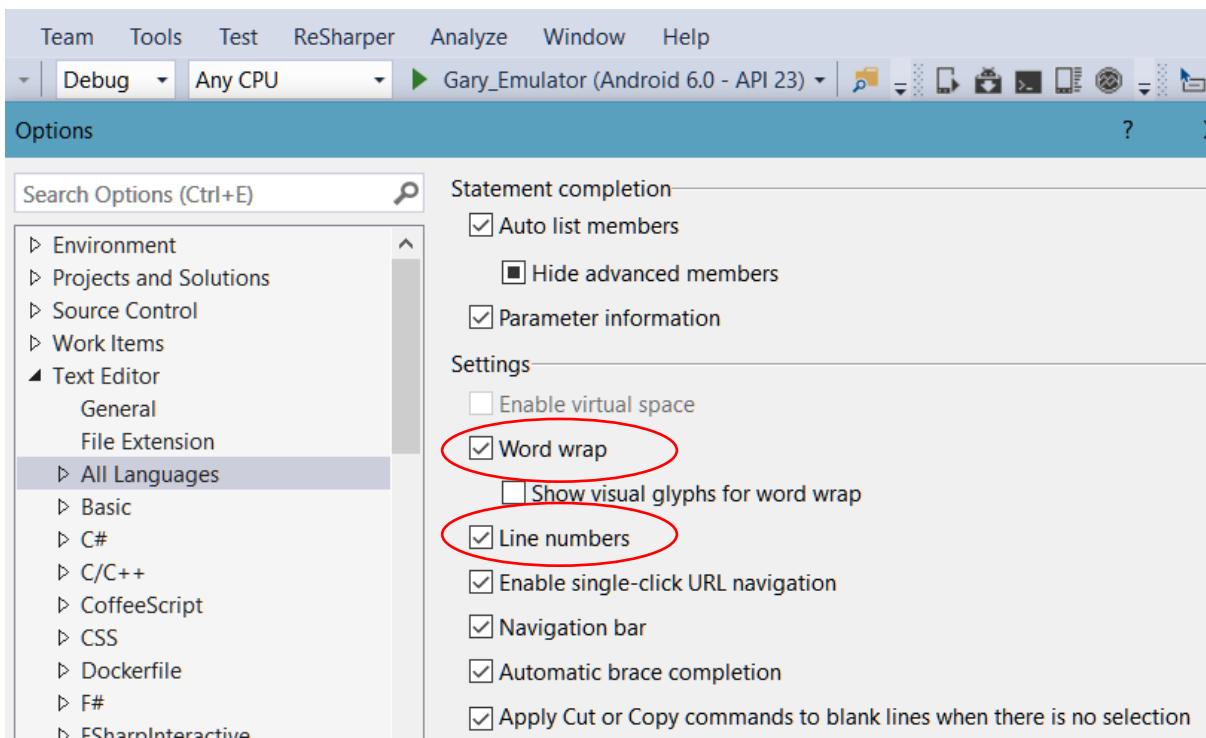
 **.NET desktop development**
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F#.

 **ASP.NET and web development**
Build web applications using ASP.NET, ASP.NET Core, HTML/JavaScript, and Containers including Docker support.

 **.NET Core cross-platform development**
Build cross-platform applications using .NET Core, ASP.NET Core, HTML/JavaScript, and Containers including Docker...

Set up the Text Editor for Word Wrap and Line Numbers

Go Tools / Options then Click on **Text Editor**, and **All Languages**.

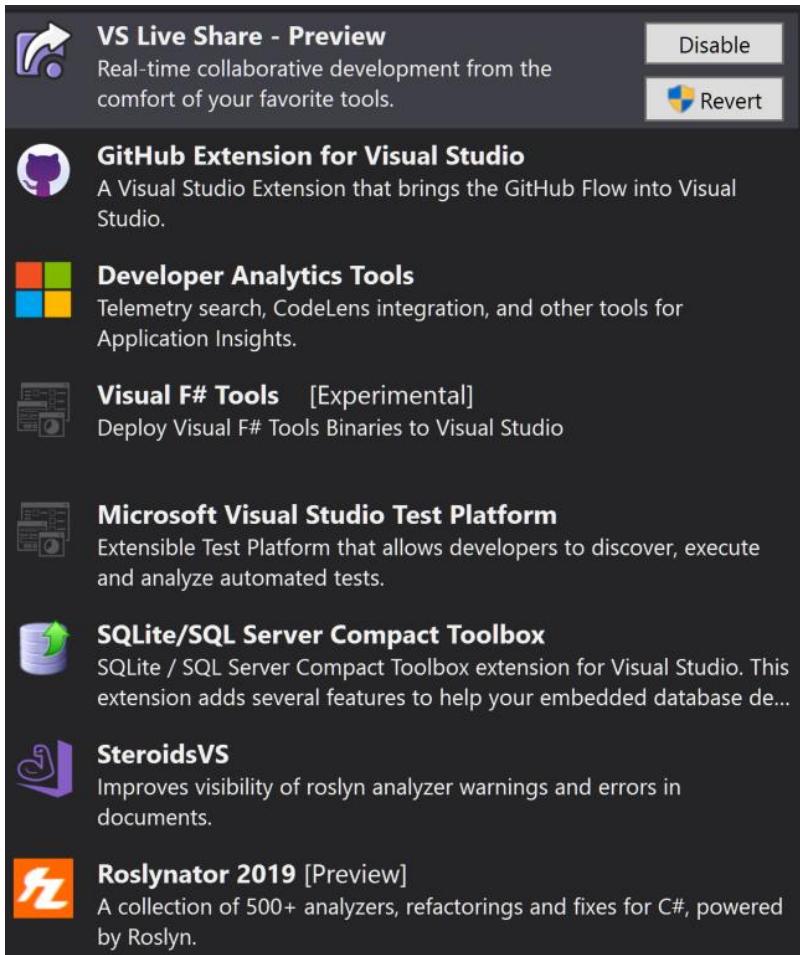


Set the ticks that you see above. I like **Word Wrap**, and it means you don't have to scroll to the right when looking through your code. **Line numbers** is a definite tick, Error messages etc. will say "Go to Line #" and you need to know where it is.

Visual Studio Extensions

Extensions are tools you add to VS to help your coding. You can install them under **Tools / Extensions and Updates**. These are my current ones from VS2019. Some are there by default and others I have added in, find the ones you don't have and install them.

Someone [here](#) is asking what developers favourite extensions are if you want to try some more.



More over the page.

 **Format document on Save**
Enables auto formatting of the code when you save a file. Visual Studio supports auto formatting of the code with the CTRL+E,D or C.

 **Visual Studio IntelliCode - Preview**
AI-assisted developer productivity

 **Viasfora**
Add color to your Visual Studio Text Editor!

 **Project File Tools**
Provides Intellisense and other tooling for XML based project files such as .csproj and .vbproj files.

 **Productivity Power Tools 2017/2019**
Installs the individual extensions of Productivity Power Tools 2017/2019

 **Power Commands for Visual Studio**
PowerCommands is a set of useful extensions for the Visual Studio IDE.

 **JavaScript Snippet Pack**
A snippet pack to make you more productive working with JavaScript

 **Microsoft Library Manager**
Install client-side libraries easily to any web project

 **JetBrains ReSharper Ultimate 2018.3.2**
A productivity extension for Visual Studio 2019

Viasfora

This is one of my favourite tools, it really makes reading code easy.

Rainbow Braces

This feature makes it easy to keep track of nested braces.

```
if (((i+1) % groupedBy) == 0) {
    finalItems.push(thisGroup);
}
```

Keyword Highlighting

Customize the appearance of control flow, visibility and LINQ keywords:

```
if ( values == null || values.Length == 0 )
    values = defaults;
return values;
```

```
String result = String.Format("{0}\n{1}\n{2}",
    firstName, lastName, address);
```

Keyboard Shortcuts

SEARCH AND NAVIGATION

| | |
|--|------------------------------------|
| Quick Launch | Ctrl+Q |
| Go to All | Ctrl+T or Ctrl+Comma |
| Go to Type / File / Member / Symbol | Ctrl+1, [T / F / M / S] |
| Navigate Backward / Forward | Ctrl+Hyphen / Ctrl+Shift+Hyphen |
| Quick Find / Quick Replace | Ctrl+F / Ctrl+H |
| Find / Replace in Files | Ctrl+Shift+F / Ctrl+Shift+H |
| Go to Definition | F12 |
| Peek Definition | Alt+F12 |
| Go to Implementation | Ctrl+F12 |
| Find All References | Shift+F12 |
| Go to Next Result in List (Find Results, Error List, etc.) | F8 / Shift+F8 (forward / back) |
| Go to Next Error | Ctrl+Shift+F12 |

EDITING AND REFACTORING

| | |
|--|----------------------------------|
| Quick Actions / Refactoring Suggestions | Alt+Enter or Ctrl+Period |
| IntelliSense Code Completion | Ctrl+Space |
| Parameter Info / Signature Help | Ctrl+Shift+Space |
| Method Info | Ctrl+K, Ctrl+I |
| Comment | Ctrl+K, Ctrl+C |
| Uncomment | Ctrl+K, Ctrl+U |
| Delete Line (without copying it) | Ctrl+Shift+L |
| Paste from Clipboard Ring (Paste from buffer of previously copied items) | Ctrl+Shift+V |
| Move Code Up / Down | Alt+Up arrow / Alt+Down arrow |
| Format Document | Ctrl+K, Ctrl+D |
| Format Selection | Ctrl+K, Ctrl+F |
| Surround with (... if/try/foreach) | Ctrl+K, Ctrl+S |
| Remove and Sort Usings | Ctrl+R, Ctrl+G |
| Rename | Ctrl+R, Ctrl+R |
| Encapsulate Field | Ctrl+R, Ctrl+E |
| Remove Parameters | Ctrl+R, Ctrl+V |
| Reorder Parameters | Ctrl+R, Ctrl+O |
| Extract Method | Ctrl+R, Ctrl+M |
| Extract Interface | Ctrl+R, Ctrl+I |

DEBUGGING AND TESTING

| | |
|---|----------------|
| Debug | F5 |
| Run (without debugging) | Ctrl+F5 |
| Stop Debugging | Shift+F5 |
| Pause Debugging | Ctrl+Alt+Break |
| Toggle Breakpoint | F9 |
| Step Over | F10 |
| Step Into | F11 |
| Step Out | Shift+F11 |
| Run All Tests | Ctrl+R, A |
| Debug All Tests | Ctrl+R, Ctrl+A |
| Run Tests in Context | Ctrl+R, T |
| Debug Tests in Context | Ctrl+R, Ctrl+T |
| Execute Code Snippet in Interactive Window | Ctrl+E, E |

WINDOW MANAGEMENT

| | |
|---|---|
| Open Tool Windows: | |
| Properties | F4 |
| Solution Explorer | Ctrl+Alt+L |
| Output Window | Ctrl+Alt+O |
| Error List | Ctrl+\, E |
| Team Explorer | Ctrl+\, Ctrl+M |
| Breakpoints | Ctrl+Alt+B |
| Immediate Window | Ctrl+Alt+I |
| Cycle Through Tool Windows | Alt+F6 / Shift+Alt+F6 (forward / back) |
| Access any Open Tool Windows and Documents | Ctrl+Tab (hold Ctrl), Up / Down / Left / Right arrows |
| Close Current Tool Window | Shift+Esc |
| Go to Document to the Left / Right | Ctrl+Alt+Page up / Ctrl+Alt+Page down |
| Go to Most Recently / Least Recently Accessed Document | Ctrl+Tab / Ctrl+Shift+Tab |
| Select Active File in Solution Explorer | Ctrl+[, S |
| Keep Preview Window Open | Ctrl+Alt+Home |
| Full Screen (max window size / reduced menus) | Shift+Alt+Enter |

Want to change your keyboard mappings quickly? Ctrl+Q, then type **keyboard**

2. Things we need on the Course

Dropbox

Get a Dropbox account from here dropbox.com/

Install it on your machine and use your Dropbox folder to store all your programming in it.

Microsoft Virtual Academy

Great resources and videos for programming

<https://docs.microsoft.com/en-us/learn/>

New Microsoft Learn website

<https://docs.microsoft.com/en-us/learn/>

Watch this introduction to C#

[C# Fundamentals for Absolute Beginners](#)

GitHub

Get an account to host your projects so others, eg: potential employers, can see them.



<https://github.com/>

Watch this series of using [Github](#)

Github is a website that provides a particular service: it stores and maintains a git repository that lives independently of your computer. Github is not the only website that provides this service -- Bitbucket and Gitlab are two other websites that also do the same.

Git is a version control system -- it's a kind of software that, if used properly, lets you keep track of the changes to your code over time (and view past versions), lets you keep track of multiple different versions of your code in a sane and principled manner, and makes sure that all of the changes you and your collaborators make stay synchronized.

Without a version control system, you'd have to resort to doing things like manually copying-and-pasting your code and emailing snippets of code to your collaborators and hoping that everything works out. This is extremely medieval.

Note that Git is not the only version control system -- Mercurial and Subversion are two other commonly used ones. Git is certainly very popular though, and is a default choice for many developers.

Github, despite being a service designed to help people collaborate/keep track of changes to their code, ended up evolving to become a sort of implicit, de-facto portfolio website. It's important to note that not everybody uses their Github profile as a portfolio (for example, I use mine as a dumping ground for random projects), but many people do.

Some people actively dislike making their code public, or need to keep their code secret to retain a competitive advantage, and will either pay Github to have private repos, will use a service like Bitbucket or Gitlab that allows free private repos, or will host their own remote repo instead of relying on some 3rd party service to do it for them. Read this → [Should beginners use Github](#)

Why Learning is Your Job

<http://wilderthum.com/opinionatedbook>

Whether you are working with older technologies or live on the bleeding edge, our jobs as developers change almost daily.

We are not assembly line workers. Building software is not that simple. Our process should be that at every release we should take the time to look back and see what we should and could be doing better. Because of this, we rarely do the same task the same way twice.

It is been said that most developers would rather build a tool to solve a thirty minute problem, even if it takes them an hour.

My experience working with developers reinforces this story. But that is not a bad thing. Developers tend to hate repetitive tasks. If they can automate anything, they will. This is a benefit of being a developer.

Developers who want to automate, innovate, and change the way a business works are the developers I want to hire. This usually means developers who understand that the basic skill of all developers is to learn.

A culture of constant learning not only keeps your skills sharp, but also means you are willing to experiment and fail in getting a task accomplished. Failing is not a bad thing. You learn by success, but you can learn even more by failing. That is the nature of learning: experimentation and iteration. People who are not afraid of this are more able to be critical of the "way it is done".

I cannot tell you how many times I have been brought into an organization that is stuck by its own entrenched culture, not by their technology choices. This inability to adapt to change is key to a poisonous culture. Organizations that reward creative thinking and open dissent usually do better in my experience.

Never agreeing does not make you a great developer. There is a difference between speaking your mind and being inflexible. Making your opinion heard is important. But trying to derail a project because your idea was not chosen is toxic and immature.

Any developer worth their salt knows that they are wrong some or even most of the time. That is where working on a team really helps. Decisions are often made through a variety of discussion, experimentation, and iteration. What this means in practice is that when you have the meeting where you voice your opinion, know that you may be wrong. But pushing the team to test out the hypothesis (or even hypotheses) will help find the right solution.

I am not saying that spending a year trying new things is the right approach either. You need to strike a balance between diving in the deep end without a plan and analysis-paralysis. As I like to say, "Everything in moderation-except moderation-use moderation in excess."

Soft Skills

If you are talking to anyone but technical recruiters, you likely know that soft skills are critical to the success of any team. What is not always obvious is that software development is not a career of solitude that Hollywood would have you believe (e.g. Jurassic Park). Even if you are building software with a very small team of people, your users and stakeholders are part of the equation.

Do not get me wrong, I like slinging code-but I also love the process of working with people to figure out what they really need. Asking stakeholders in a solution to explain the problem is fascinating. This is often the time when they introduce me to their domain. I ask them to explain the problem in their own words, then I break down the concepts one at a time. This process of active listening is a key skill for any developer. Luckily, this crucial skill is obvious at interview time, but not all interviewers can hear it.

Active listening is simply not spending the time you are not talking thinking about what you are going to say. Listening as intently as you are when you are speaking. You want to not only hear, but understand what people are saying. For me it takes the form of "What I hear you saying is that you do X, Y then Z."

Repeating what the stakeholders tell me in my own words tells them that I heard what they said and helps me validate what they mean, not just repeating the words they used.

You want to delve into a conversation, not get the answer you are looking for. I like to have these conversations without expectations of what they will tell me. This is true of the discovery phase of a project - though this also happens throughout the process as you need more information about how a system should behave.

Being a developer with great communication skills is much more valuable to me than knowing a particular technology. Good people can learn new technologies, but understanding how to communicate is much harder to teach.

The **Must Have** books for every developer....

The internet will make those bad words go away



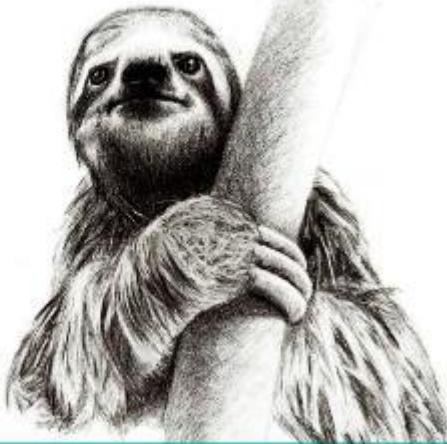
Essential

Googling the Error Message

O RLY?

*The Practical Developer
@ThePracticalDev*

Cutting corners to meet arbitrary management deadlines



Essential

Copying and Pasting from Stack Overflow

O'REILLY*

*The Practical Developer
@ThePracticalDev*

https://www.reddit.com/r/ProgrammerHumor/comments/4day9p/must_have_book_for_every_developer/

About Time – The Pomodoro Method

The most common question I am asked these days is how I have time to look at certain new technologies. This is a hard question because I do not work that hard (as far as I am concerned) but I am passionate about technology so I find the time is fun, not work.

This idea of how to use your time is a subject that most developers should think about.

Being Productive

Developers are not like most employees. Back in a more ancient time (the 80's) you would get paid by the line of code you wrote. The idea was that you needed to be writing code to accomplish your task. These days that seems crazy and it was.

These days we've (hopefully) learned that all work can be accomplished at the keyboard. I find it interesting in that at most companies I worked, developers were treated differently from other employees.

They were a bit like the misfits of the company and treated with kid gloves at times. This often caused friction with other groups as perks like foosball and whiteboards separated the development teams from other teams in the company.



not

What good companies have figured out is that developers can accomplish quite a lot when they are given a little freedom in work habits. I still consider myself productive if I can spend as much as 4 hours a day coding (less if I am managing a team where I spend more time in communication than coding). That means that the other half of the time should be plenty to handle the mundanity of most jobs. But that also includes time spent helping other developers and asking for help yourself.

Back in the day there was not as much to distract. The attraction of a web browser or the ding of an email can sap out your working time pretty easily. How can you manage that time better and keep focused?

There are many ways but one technique that has really helped me is the Pomodoro method. In this method you work in cadences of work and breaks called a "Pomodoro".

A single Pomodoro is made up of twenty-five minutes of work with a five minute break. Every third Pomodoro, the break is increased to 15 minutes. The idea is that when you are working, just work. Do not stop for any kind of distraction. Do not answer emails, turn off your phone and focus. The length of the work cycle at twenty-five minutes is longer than you might imagine if you work straight through. Then during the break, you cannot work. If you think "Hey it is going well, I am going to just push through," do not. Take the break.

I have found that using this method can dramatically increase my productivity in a single day. <http://pomodorotechnique.com/>

How to be an Excellent Programmer

Watch the video

<http://pluralsight.com/training/Courses/TableOfContents/writing-clean-code-humans>

Read the book in the Goodies Folder

Read the Reviews and comments

http://www.reddit.com/r/learnprogramming/comments/29tl74/thoughts_on_clean_code/

Clean code changed the way I program. I would say I went from a low intermediate developer to someone whose code could be passed off as an expert.

Although it took a lot more time and planning it feels so good when everything just works in the best possible way

When people start out coding, they think "how can I do this today?" But after you've had to maintain and iterate on a code base for 18 months or so, you realize that most of the value in code you write today is in whether it makes your job 30 days from now a cakewalk or a kick in the nuts.

The number one rule I encourage people to follow if they want to write cleaner code is to think of the difference between writing a journal and writing a novel.

If you're writing a private journal, you don't need to spell everything out in detail, just enough to help you remember things you've experienced. A novelist, on the other hand, needs to provide a lot more detail, because a novel reader doesn't know anything that the author doesn't write.

Write code as if you are explaining how your program works to whoever will work on it later (and consider your future self as a different person, because in 6 months, you will have no memory of what you are thinking as you code now).

You don't need to do this by spending hours writing detailed documentation. If you choose good names for your functions and variables, and break functions up into logical blocks, your code will explain itself with minimal documentation.

How to become a successful programmer - from Reddit

[Java]I just got my first job as developer and I don't know what to do. ([self.learnprogramming](#))

Aa submitted 1 day ago * (last edited 6 hours ago) by Kanaxai 

171 comments [source](#) [share](#) [unsave](#) [hide](#) [give gold](#) [report](#) [hide all child comments](#)

Turns out the job requires knowledge in web development, but I've never done any of that, where can I start learning so I don't make a fool of myself?

Edit: Thanks for all the answers, it's still simple but I managed to build my first web application, my HTML could use a bit of work, but it seems my employee is not too worried about the appearance.

I have two grads working for me at the moment. One of them is really excelling, while the other is being left behind.

The good grad always jumps on to Google or asks constructive questions when given a task he doesn't know how to do. The other grad immediately says "I

didn't learn that at Uni" like that means anything. Then he expects us to teach him, or he'll take a week of "learning" and not really get anywhere.

The good grad will ask just enough to get started, then he will learn as he goes. He will also seek feedback and is not afraid to ask more questions (the other grad will just hide away not making any progress if he gets stuck).

Anyway I just wanted to share so that you know what not to do. Do not make excuses and expect to be trained before jumping in to a project.

Everyone expects you to have a ramp up time, so don't feel bad about asking questions as you go. If you are a smart person and a good developer then you can quickly adapt to anything new they throw at you.

http://www.reddit.com/r/learnprogramming/comments/2s9u3a/javai_just_got_my_first_job_as_developer_and_i/

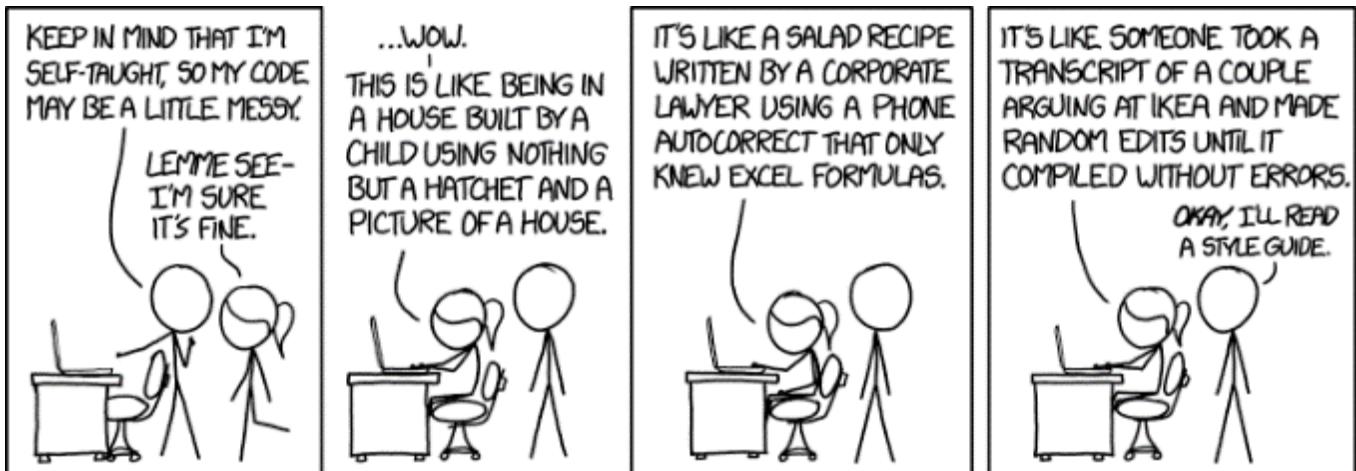
C# Programming Yellow Book – good starter

The C# Yellow Book is used by the [Department of Computer Science](#) in the [University of Hull](#) as the basis of the First Year programming course. You can download your own copy from [here](#).

<http://www.csharpcourse.com/>

Clean Code – Planning for the ‘stupid you’

Read the book, watch the videos, read the reviews and other programmers input.



CommitStrip.com

Watch the video

<http://pluralsight.com/training/Courses/TableOfContents/writing-clean-code-humans>

Read the book in the Goodies Folder

Read the Reviews and comments

http://www.reddit.com/r/learnprogramming/comments/29tl74/thoughts_on_clean_code/

Clean code changed the way I program. I would say I went from a low intermediate developer to someone whose code could be passed off as an expert.

Although it took a lot more time and planning it feels so good when everything just works in the best possible way

When people start out coding, they think "how can I do this today?" But after you've had to maintain and iterate on a code base for 18 months or so, you realize that most of the value in code you write today is in whether it makes your job 30 days from now a cakewalk or a kick in the nuts.

The number one rule I encourage people to follow if they want to write cleaner code is to think of the difference between writing a journal and writing a novel.

If you're writing a private journal, you don't need to spell everything out in detail, just enough to help you remember things you've experienced. A novelist, on the other hand, needs to provide a lot more detail, because a novel reader doesn't know anything that the author doesn't write.

Write code as if you are explaining how your program works to whoever will work on it later (and consider your future self as a different person, because in 6 months, you will have no memory of what you are thinking as you code now).

You don't need to do this by spending hours writing detailed documentation. If you choose good names for your functions and variables, and break functions up into logical blocks, your code will explain itself with minimal documentation.

Some coding ideas for new programmers

Write code for clarity, not brevity

Whenever you can choose between concise (but potentially confusing) code and clear (but potentially tedious) code, use code that *reads* as intended, even if it's less elegant. For example, split complex arithmetic operations into a series of separate statements to make the logic clearer.

Think about who might read your code. It might require maintenance work by a junior coder, and if he can't understand the logic, then he's bound to make mistakes.

Complicated constructs or unusual language tricks might prove your encyclopaedic knowledge of operator precedence, but it really butchers code maintainability. *Keep it simple.*



The eternal argument contuse, read it [here](#)

Don't let anyone tinker with stuff they shouldn't

Things that are internal should stay on the inside. Things that are private should be kept under lock and key. Don't display your code's dirty laundry in public. In object-oriented languages, prevent access to internal class data by making it private.

Keep all variables in the tightest scope necessary; don't declare variables globally when you don't have to. Don't put them at file scope when they can be function-local. Don't place them at function scope when they can be loop-local.

Check every return value

If a function returns a value, it does so for a reason. Check that return value. If it is an error code, you *must* inspect it and handle any failure. Don't let errors silently invade your program; swallowing an error can lead to unpredictable behaviour.

Initialize all variables at their points of declaration

This is a clarity issue. The intent of each variable is explicit if you initialize it. It's not safe to rely on rules of thumb like *If I don't initialize it, I don't care about the initial value*. The code will evolve. The uninitialized value may turn into a problem further down the line.

Declare variables as late as possible

By doing this, you place the variable as close as possible to its use, preventing it from confusing other parts of the code. It also clarifies the code using the variable. You don't have to hunt around to find the variable's type and initialization; a nearby declaration makes it obvious.

Don't reuse the same temporary variable in a number of places.

Even if each use is in a logically separate area. It makes later reworking of the code awfully complicated. Create a new variable each time—the compiler will sort out any efficiency concerns.

How programmers precode, or prepare for coding.

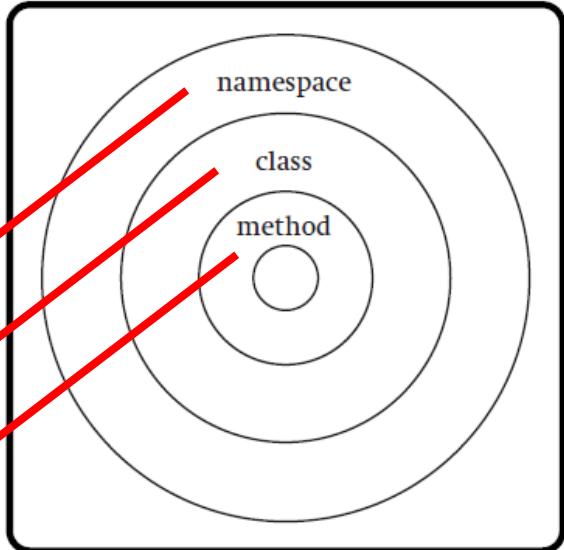
I like the “I'll cross that bridge when I get to it” strategy.

http://www.reddit.com/r/dailyprogrammer/comments/2ao9y3/weekly_2_precoding_work/

3.What does all the code layout mean?

Let's look at the seeming spaghetti mess of code and start to break it down and show what it is doing.

The basic concepts behind C# programming are apparent in even the simplest programs. Essentially, a C# program can be thought of as an onion with a bunch of layers.



The diagram shows three concentric circles representing layers of an onion. The outermost circle is labeled "namespace", the middle circle is labeled "class", and the innermost circle is labeled "method". Three red arrows point from the text "namespace", "class", and "method" in the explanatory text below to the corresponding labels on the onion diagram.

```
namespace Calculator2019
{
    public class Operations
    {
        //set your method to public so you can see it in the other code
        public string Divide(string Num1, String Num2)
        {
            Single sngNum1, sngNum2, sngAnswer;
```

The outer, most general, layer is the namespace. Inside a namespace, you find a series of classes, which contain methods, which contain statements.

Namespaces

The various layers of programming help you organize your programs. Think of the layers as something like an address on an envelope.

When you address an envelope, you write specific information, such as the house number. You also put the street name, which is more general, and the state, which is broad. The post office can deliver your letter by getting it to the correct state, then the correct city, then the right part of the city, and finally the specific house.

Namespaces in the C# language work very much like this.

The largest landscape in the C# universe is a namespace. You can think of a namespace as a state in the postal analogy. A namespace is an element that enables you to group together a series of other things.

Each project you create is usually a namespace. In addition, all the various things you can use in your programs—including the computer system itself, and Windows elements, such as text boxes and buttons—are separated into namespaces.

Frequently, you specify which namespaces you want to work with, for example, to define whether a program should use Windows forms or a special library of math functions. If all this seems unclear to you, don't worry about it. Soon you will see examples that make it clear.

Classes

A namespace is usually made up of one or more classes. A class is a definition for a specific kind of object. Classes and objects are essentially the same they are used to describe some type of entity. Anything a computer can describe (a database, a file, an image, a cow, whatever) can be encoded as an object.

The things an object can do are called its methods, and the characteristics of an object are called its properties.

Methods

Classes always have methods. A method is a structure that contains instructions. All the commands in a program are housed in various methods of objects. Most programs have a special method named Main () (method names always end in parentheses), which is meant to execute as soon as the program begins running.

If you are familiar with other languages, such as C or Visual Basic, you will see that methods are a lot like functions or subprograms in those languages.

Statements

Inside a method, you write the instructions you want the computer to execute. A statement is an instruction. Many statements (sometimes also called commands) involve using methods of built-in objects. Of course, a computer scientist wouldn't usually say using a method, because everyone would understand that. Often C# folks will refer to the process as invoking a method. Maybe at dinner tonight rather than asking somebody to pass the salt, you could say "Could you please invoke the salt shaker object's pass method?" It should liven up the conversation.

Private and Public

All types and type members have an accessibility level, which controls whether they can be used from other code in your program or even other programs. You can use the following access modifiers to specify the accessibility of a type or member when you declare it:

public

The type or member can be accessed by any other code in the same assembly or another assembly that references it.

private

The type or member can be accessed only by code in the same class or struct.

protected

The type or member can be accessed only by code in the same class or struct, or in a class that is derived from that class.

internal

The type or member can be accessed by any code in the same assembly, but not from another assembly.

protected internal

The type or member can be accessed by any code in the assembly in which it is declared, or from within a derived class in another assembly. Access from another assembly must take place within a class declaration that derives from the class in which the protected internal element is declared, and it must take place through an instance of the derived class type.

void

A void is nothing. In programming terms the void keyword means that the method we are about to describe does not return anything of interest to us. The method will just do a job and then finish. In some cases, we write methods which return a result (in fact we will use such a method later in the program).

However, in order to stop someone else accidentally making use of the value returned by our Main method, we are explicitly stating that it returns nothing. This makes our programs safer, in that the compiler now knows that if someone tries to use the value returned by this method, this must be a mistake.

(AN UNMATCHED LEFT PARENTHESIS
CREATES AN UNRESOLVED TENSION
THAT WILL STAY WITH YOU ALL DAY.)

(Brackets) ()

This is a pair of brackets enclosing nothing. This may sound stupid, but actually tells the compiler that the method Main has no parameters. A parameter to a method gives the method something to work on. When you define a method, you can tell C# that it works on one or more things, for example sin(x) could work on a floating-point value of angle x.

{ brace } {}

This is a brace. As the name implies, braces come in packs of two, i.e. for every open brace there must be a matching close. Braces allow programmers to lump pieces of program together. Such a lump of program is often called a block. A block can contain the declaration of variables used within it, followed by a sequence of program statements which are executed in order. When the compiler sees the matching close brace at the end it knows that it has reached the end of the method and can look for another (if any). The effects of an un-paired brace are invariably fatal....

Operators

There are various types of operators in C#. Here are the Arithmetic operators (for example, the expression `5 + 4` yields a value of **9**):

- ^** Exponentiation
- *** Multiplication
- /** Division
- ** Integer division
- Mod** Modulus
- +** Addition
- Subtraction

These are the Assignment operators (for example, `temperature = 72` stores the value **72** in the variable **temperature**):

- =** Assignment
- ^=** Exponentiation followed by assignment
- *=** Multiplication followed by assignment
- /=** Division followed by assignment
- \=** Integer division followed by assignment
- +=** Addition followed by assignment
- =** Subtraction followed by assignment
- &=** Concatenation followed by assignment

Here are the Comparison operators (these values yield true or false values—for example, `5 > 4` yields a value of **True**):

- <** (Less than) —**True** if *operand1* is less than *operand2*
- <=** (Less than or equal to)—**True** if *operand1* is less than or equal to *operand2*
- >** (Greater than)—**True** if *operand1* is greater than *operand2*
- >=** (Greater than or equal to)—**True** if *operand1* is greater than or equal to *operand2*
- =** (Equal to)—**True** if *operand1* equals *operand2*
- !=** (Not equal to)—**True** if *operand1* is not equal to *operand2*
- Is** —**True** if two object references refer to the same object
- Like**—Performs string pattern matching

These are the Logical/Bitwise operators, where *bitwise* means working bit by bit with numerical values.

These types of operators can work on logical values

(for example, if **bInValue1** is set to **True** and **bInValue2** is set to **False**, then **bInValue1 Or bInValue2** returns a value of **True**)

or numbers for bitwise operations, which work on their operands bit by bit (for example, if `intValue1` is set to 2 and `intValue2` is set to 1, then `intValue1 Or intValue2` yields 3):

And— Performs an **And** operation (for logical operations: **True** if both operands are **True**, **False** otherwise; the same for bit-by-bit operations where you treat **0** as **False** and **1** as **True**).

Not— Reverses the logical value of its operand, from **True** to **False** and **False** to **True**, for bitwise operations, turns **0** into **1** and **1** into **0**.

Or — Operator performs an **Or** operation (for logical operations: **True** if either operand is **True**, **False** otherwise; the same for bit-by-bit operations where you treat **0** as **False** and **1** as **True**).

Xor— Operator performs an **exclusive-Or** operation (for logical operations: **True** if either operand, but not both, is **True**, and **False** otherwise; the same for bit-by-bit operations where you treat **0** as **False** and **1** as **True**).

AndAlso— Operator **A** "short circuited" **And** operator; if the first operand is **False**, the second operand is not tested.

OrElse— Operator **A** "short circuited" **Or** operator, if the first operand is **True**, the second is not tested.

And here are two other miscellaneous operators:

AddressOf— Gets the address of a procedure.

GetType— Gets information about a type.

+= is a combination of **+** and **=**, which means that you can write `intValue1 = intValue1 + 1` as `intValue1 += 1`.

In a similar way, you can write `intValue1 = intValue1 * 5` as `intValue1 *= 5`, providing an easy shortcut.

Variables

It's not enough to have data moving from textboxes to labels and out of the program. You need to be able to work with data in the program. To do this you need to create variables, storage places to hold information that you want to use.

A variable is a place where you can store something. You can think of it as a box of a particular size with a name painted on the box. You chose the name to reflect what is going to be stored there. You also need to choose the type of the variable (particular size and shape of box) from the range of storage types which C# provides.

You tell C# about a variable you want to create by declaring it. The declaration also identifies the type of the thing we want to store. Think of this as C# creating a box of a particular size, specifically designed to hold items of the given type.

Data Types

Data Types define the type of data that a variable can store. Some variables store numbers, others store names. The built-in C# type aliases and their equivalent .NET Framework types follow:

Integers

| C# Alias | .NET Type | Size | Range |
|----------------------|---------------|-------------------|---|
| SByte | System.SByte | 8 bits (1 byte) | -128 to 127 |
| Byte | System.Byte | 8 bits (1 byte) | 0 to 255 |
| Short | System.Int16 | 16 bits (2 bytes) | -32,768 to 32,767 |
| UShort | System.UInt16 | 16 bits (2 bytes) | 0 to 65,535 |
| Integer (int) | System.Int32 | 32 bits (4 bytes) | -2,147,483,648 to 2,147,483,647 |
| UInteger | System.UInt32 | 32 bits (4 bytes) | 0 to 4,294,967,295 |
| Long | System.Int64 | 64 bits (8 bytes) | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| ULong | System.UInt64 | 64 bits (8 bytes) | 0 to 18,446,744,073,709,551,615 |

Floating-point

| C# Alias | .NET Type | Size | Precision | Range |
|----------|----------------|---------------------|----------------------|---|
| Float | System.Single | 32 bits (4 bytes) | 7 digits | 1.5×10^{-45} to 3.4×10^{38} |
| Double | System.Double | 64 bits (8 bytes) | 15-16 digits | 5.0×10^{-324} to 1.7×10^{308} |
| Decimal | System.Decimal | 128 bits (16 bytes) | 28-29 decimal places | 1.0×10^{-28} to 7.9×10^{28} |

Storing real values with Float

"Real" is a generic term for numbers which are not integers. They have a decimal point and a fractional part. Depending on the value the decimal point floats around in the number, hence the name float.

C# provides a type of box which can hold a real number. A standard float value has a range of 1.5E-45 to 3.4E48 with a precision of only 7 digits (i.e. not as good as most pocket calculators).

If you want more precision you can use a double box instead (double is an abbreviation for double precision). This takes up more computer memory but it has a range of 5.0E-324 to 1.7E308 and a precision of 15 digits.

An example of a float variable could be something which held the average price of ice cream:

```
float averageIceCreamPriceInPence;
```

An example of a double variable could be something which held the width of the universe in inches:

```
double univWidthInInches;
```

Finally, if you want the ultimate in precision but require a slightly smaller range you can use the decimal type. This uses twice the storage space of a double and holds values to a precision of 28-29 digits. It is used in financial calculations where the numbers are not so large but they need to be held to very high accuracy.

```
decimal MyOverdraft;
```

Other pre-defined types

| C# Alias | .NET Type | Size (bits) | Range |
|----------|-----------------|---|---|
| Char | System.Char | 16 bits (2 bytes) | One Unicode symbol in the range of 0 to 65,535. |
| Boolean | System.Boolean | 32 bits (4 bytes) | True or False |
| Object | System.Object | 32/64 bits (4/8 bytes) | Platform dependant (a reference to an object). |
| Date | System.DateTime | 64 bits (8 bytes) | January 1, 0001 12:00:00 AM to December 31, 9999 11:59:59 PM |
| String | System.String | 80 + [16 * Length] bits (10 + [2 * Length] bytes) | A Unicode string with a maximum length of 2,147,483,647 characters. |

To use a variable, you first have to tell the program that it exists. This happens automatically when you put an object, such as button on a form, but you have to do it manually when you create a variable.

The Ternary Operator ?: (IF shortcut)

C# uses something called a *ternary operator* to provide a shortcut way of making decisions. The syntax of the ternary operator is as follows:

[condition] ? [true expression] : [false expression]

The way this works is that [condition] is replaced with an expression that will return either *true* or *false*. If the result is *true* then the expression that replaces the [true expression] is evaluated. Conversely, if the result was *false* then the [false expression] is evaluated. Let's see this in action:

```
int x = 10;
int y = 20;
System.Console.WriteLine(x > y ? x : y);
```

The above code example will evaluate whether x is greater than y. Clearly this will evaluate to false resulting in y being returned to the WriteLine method for display to the user.

How to declare a variable

A variable can be compared to a storage room, and is essential for the programmer. In C#, a variable is declared like this:

```
<data type> <name>;
```

An example could look like this:

```
string Firsname;
```

That's the most basic version. Usually, you wish to assign a value to it at the same time. It can be done like this:

```
<data type> <name> = <value>;
```

And with an example:

```
string Firstname = "John";
string lastName = "Doe";
```

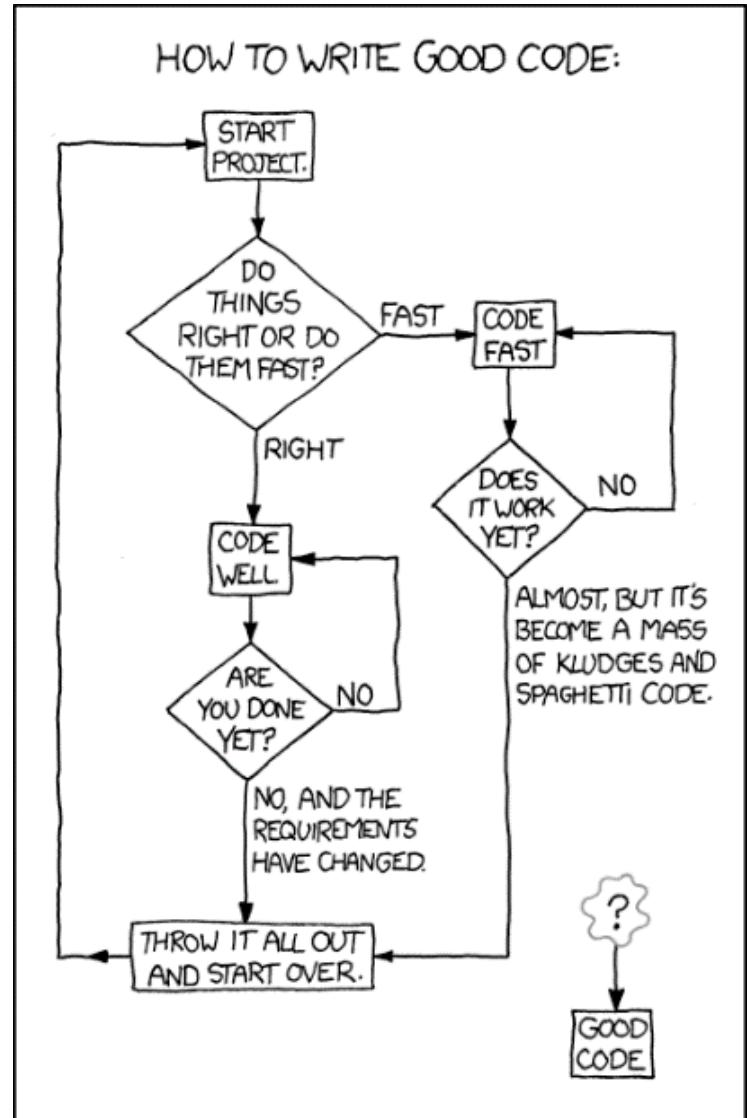
A **string** is a variable that holds words (this is a word) or text (asd sdf), or data that doesn't need to be operated on like phone numbers (3455-45-32). It wouldn't be used to hold ages (23), or dates (2/2/2012), or prices (\$4.98).

Renaming your code automatically - Refactoring

If you want to rename your code, such as your variables, you can do it all at once. When you change the name click on the small icon that appears on the bottom right and choose the option shown.

This seems to Bork at changing the same name multiple times however. This is a really good tool for your programming.

```
string strname = "Gary";
int age = 0;
int years = 2012;
pu { Rename 'year' to 'years'
      Rename with preview...
```



4. Introduction to C#

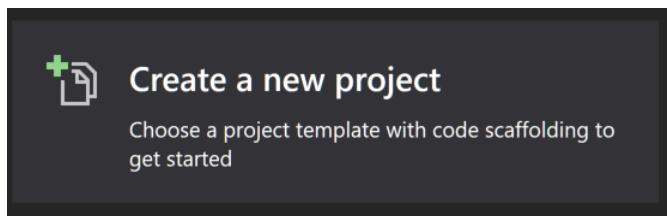
Check out the [Visual Studio 2017 interface](#) Read this to get an idea of what the IDE does.

Hello World Lesson

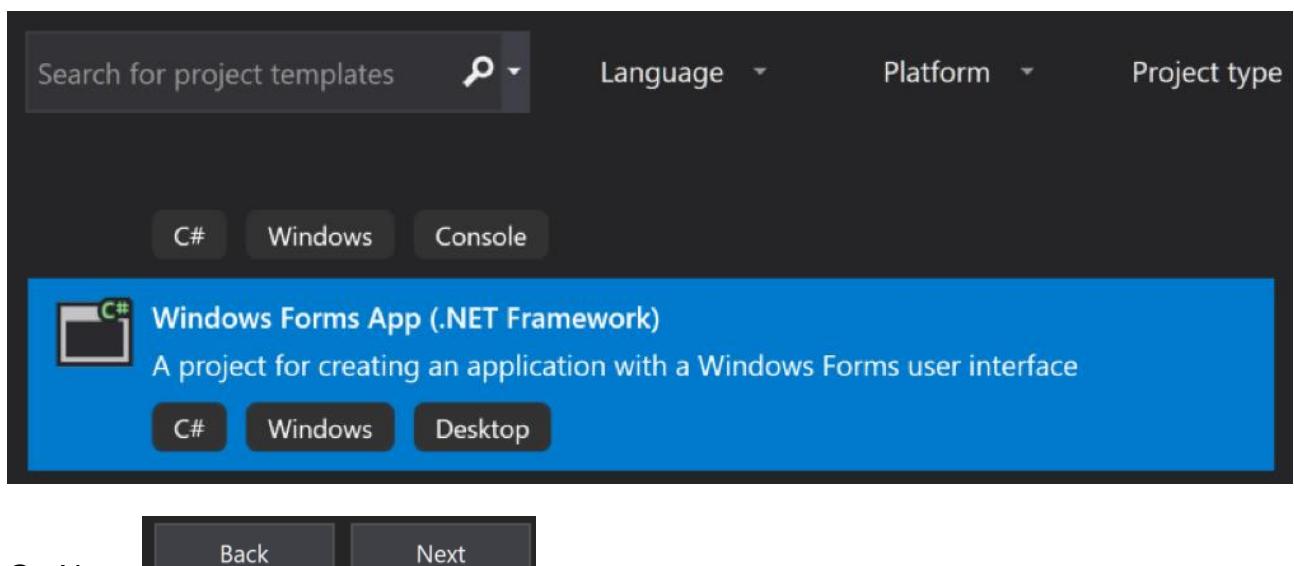
Open a new C# project

You might see **Windows Form App** showing in the window below on the start screen. Otherwise click on **More Project Templates** and choose from there.

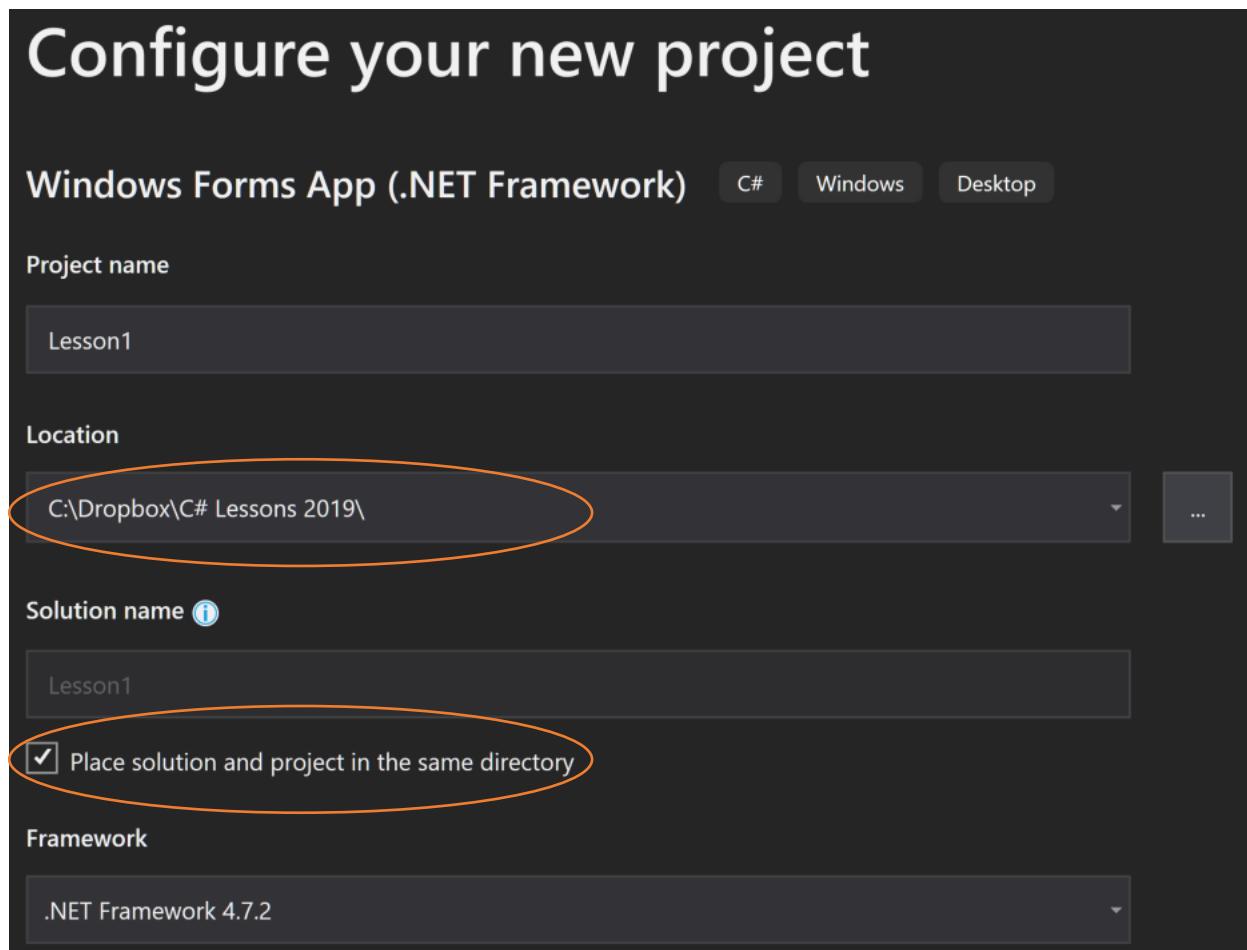
Open Visual Studio and click on Create a New Project



Scroll down the options and find **Windows Forms App**



Save it with a name Lesson1, to a folder that holds all your projects, or wherever you want, as Lesson1. My folder is called **C# lessons 2019**.

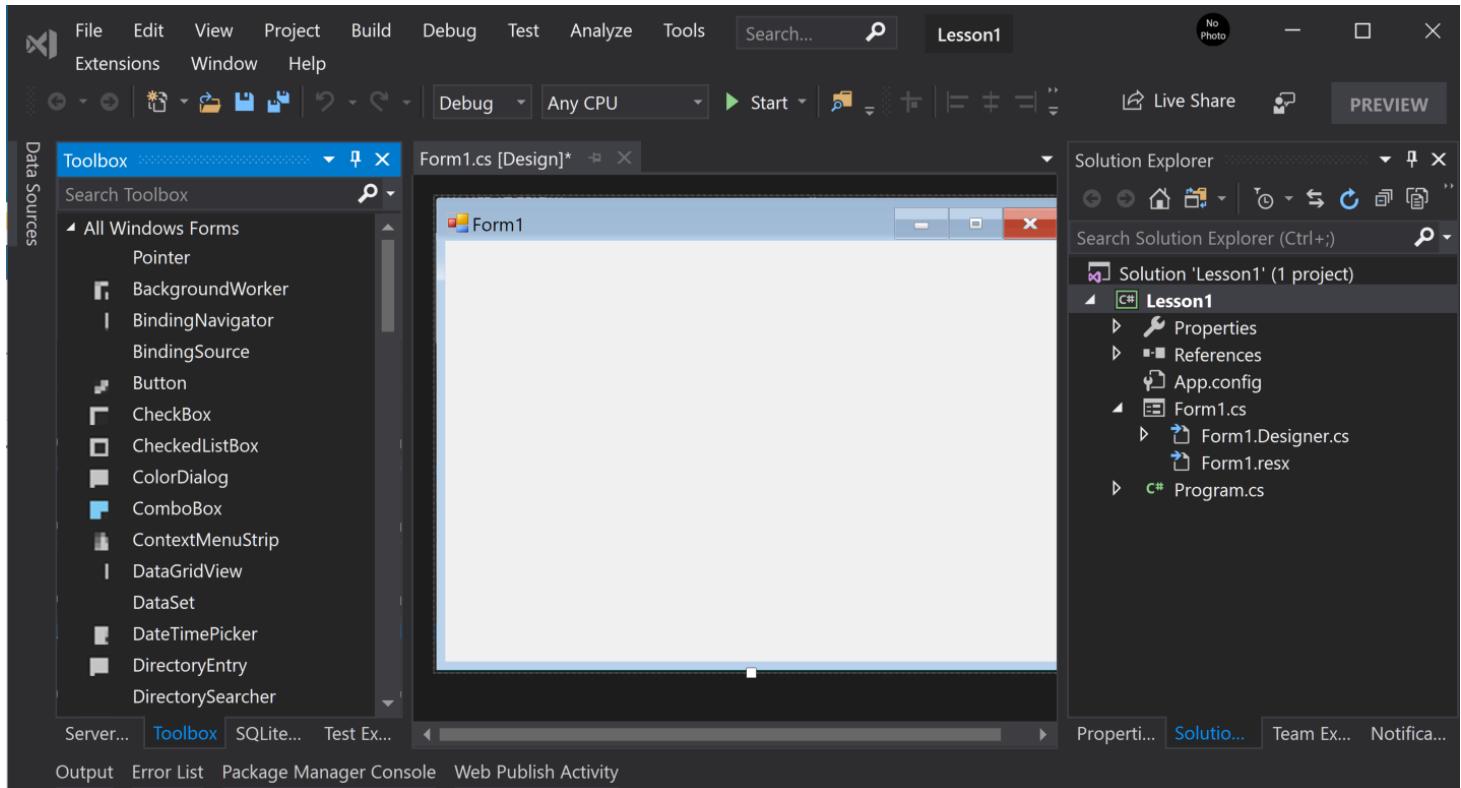


Name

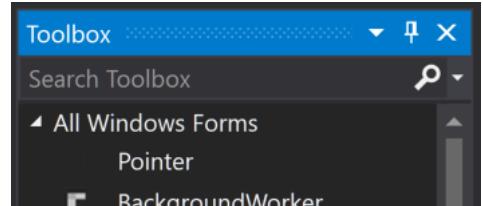
- CounsellingTriads
- Hangman React
- Lesson1
- UnitTestProject1

Always save a new project in a new folder as chaos will ensue unless you are clear about your saving.

Set up your workspace to look like this.

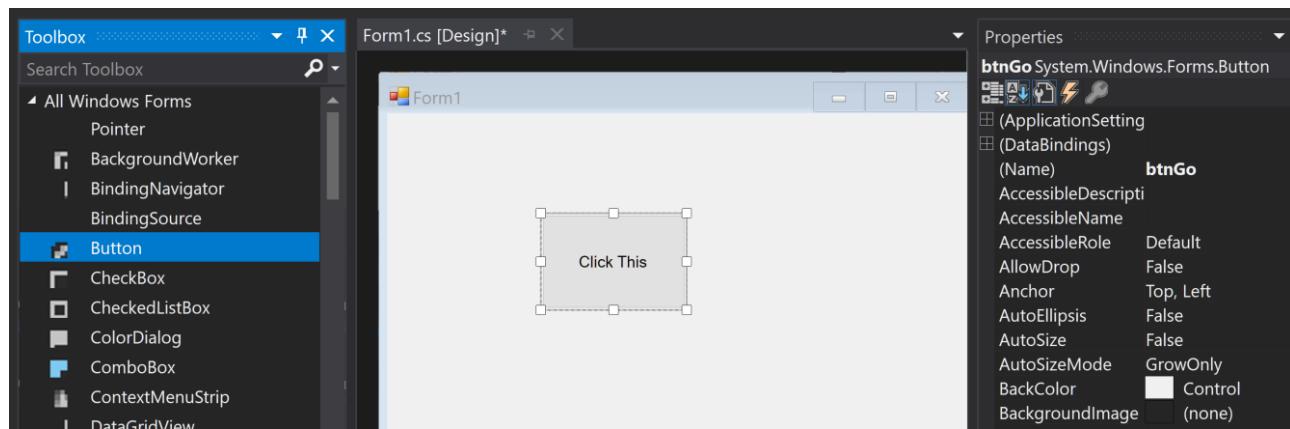


Click on the Toolbox press the pin to pin it to the side. As we have a good-sized screen we have the space available to leave it open.



From the Toolbox in Common Controls drag on the following and set the properties on the right to the marked settings.

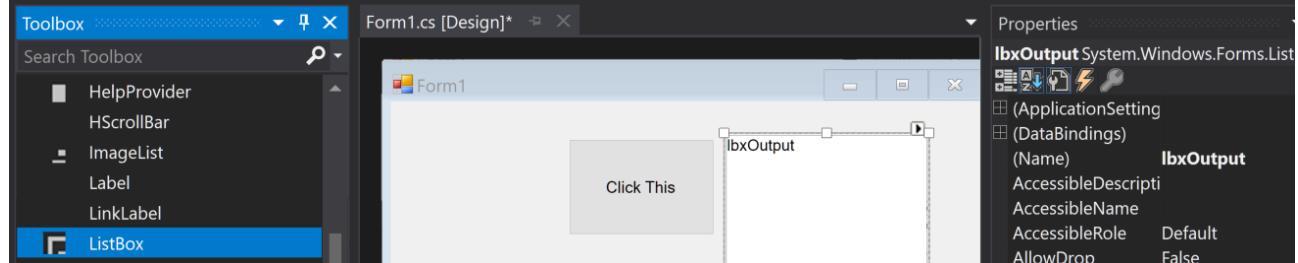
1 Button - named btnGo **btn** = button
Set **Name** to **btnGo** and Set **Text** field to Click This



1 ListBox – named **IbxOutput**

Set **Multicolumn** to **true**

Ibx = **ListBox**



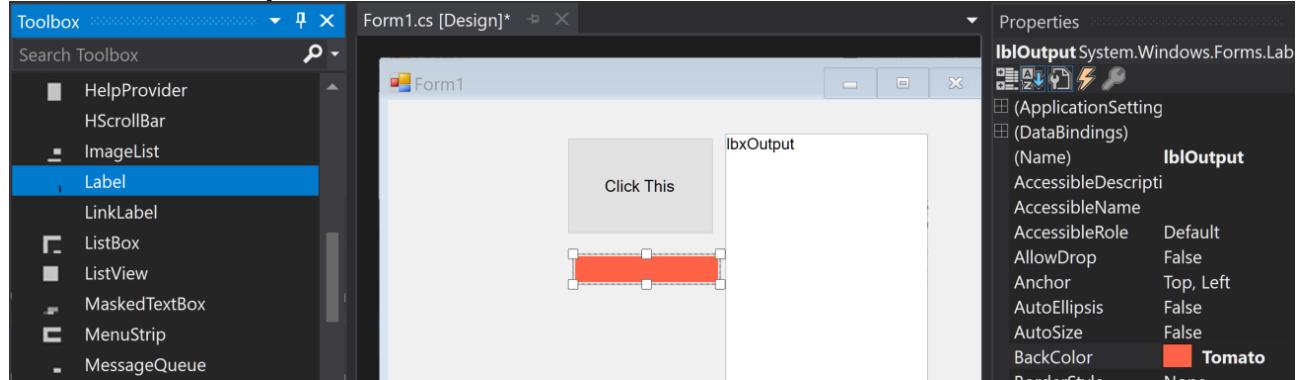
1 label named **IblOutput**

Set **Text** field to empty – clear it

Set **Autosize** to **false**

Set **Backcolor** to your choice

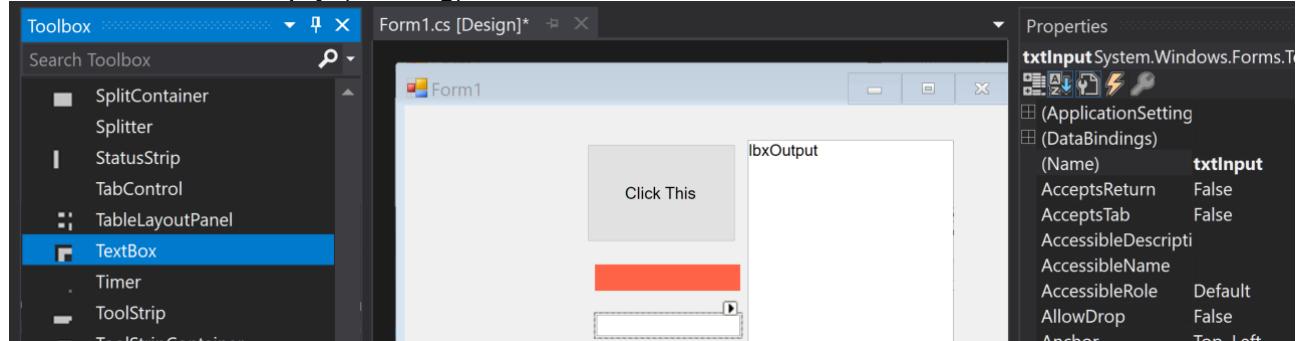
Ibl = **label**



1 text box – named **txtInput**

Set **Text** field to empty (nothing)

txt = **textbox**

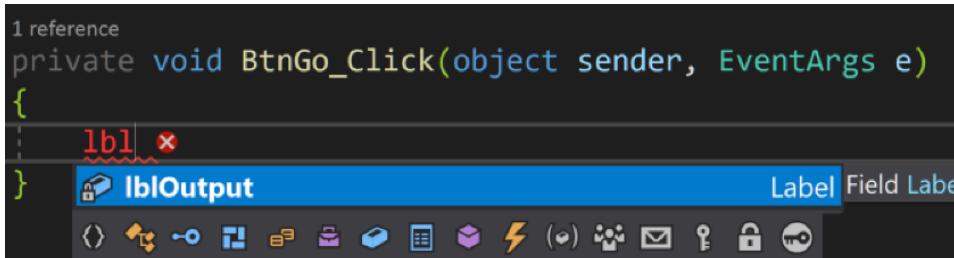


Double click on the button to open the code window.

Although this only shows the code for the Button at present all code (at this level at least) will go in this window.

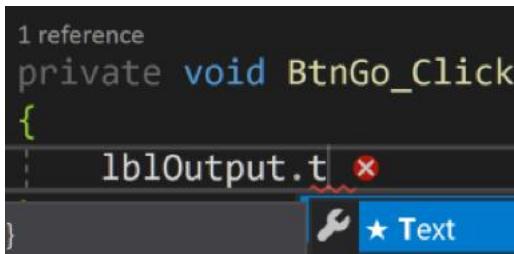
Next, we will add text, **use the autocomplete every time you make an entry, then you know you won't have any errors.**

So, start to type below and when the word **lbl** appears press the **Down arrow** and then press **Tab**.



```
1 reference
private void BtnGo_Click(object sender, EventArgs e)
{
    lbl x
}
```

The code editor shows the beginning of a C# method definition. The variable 'lbl' is typed, and an intellisense dropdown is open, showing 'Label' as the top suggestion. Other suggestions like 'Field' and 'Label' are also visible.



```
1 reference
private void BtnGo_Click
{
    lblOutput.t x
}
```

The code editor shows the continuation of the method body. The variable 'lblOutput.t' is typed, and an intellisense dropdown is open, showing 'Text' as the top suggestion. Other suggestions like 'Field' and 'Text' are also visible.

Then press the **full stop** press the **Down arrow** and then press **Tab** to insert Text..

Always type in this way and you won't get any errors.

IF YOU DON'T GET THE INTELLIGENCE YOU ARE DOING IT WRONG

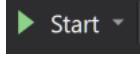
Type in `lblOutput.Text = "Hello World";`



```
20
21
22
23
```

```
1 reference
private void BtnGo_Click(object sender, EventArgs e)
{
    lblOutput.Text = "Hello World";
}
```

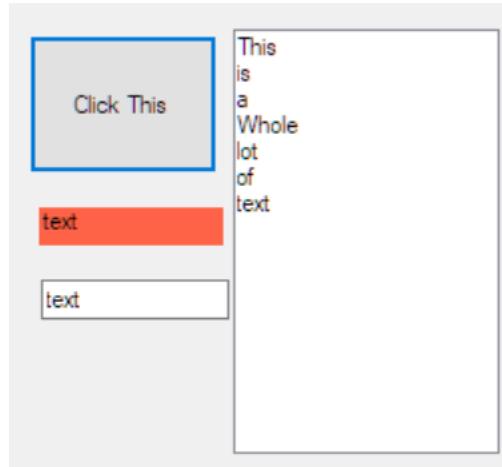
The code editor shows the completed line of code. The string value 'Hello World' has a red wavy underline underneath it, indicating a potential error or warning.

Click the green run button  to see it working, then click on your button.

Now alter it to look like this...

The first line has been commented out with a **//** **It's better to comment out code than delete it when you are learning**, you never know when you need it again.

```
1 reference
private void BtnGo_Click(object sender, EventArgs e)
{
    //    lblOutput.Text = "Hello World";
    lblOutput.Text = txtInput.Text;
    lbxOutput.Items.Add(txtInput.Text);
}
```



Run it. Play with it.

Add more entries to the text box and see them list in the ListBox. Listboxes are fun, and really handy tools to use for debugging as well.

Back in your designer Double click on the ListBox and then in the code window enter `lbxOutput.Items.Clear()`

Remember to use the Intellisense. This will clear the ListBox when you click on it when it is running

```
1 reference
private void LbxOutput_SelectedIndexChanged(object
| sender, EventArgs e)
{
    lbxOutput.Items.Clear();
}
```

Now when you run the program if you click the ListBox it clears out the text.

Add your Name

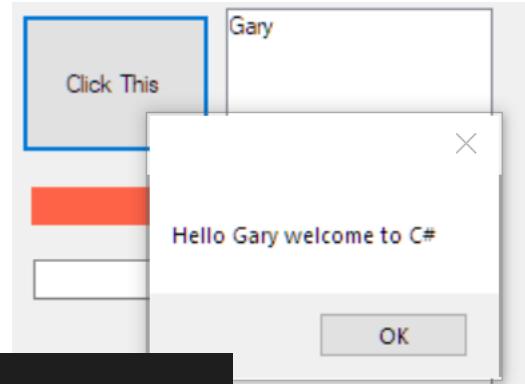
Add the following. Use your **OWN NAME** and use the **Intellisense**

```
1 reference
private void BtnGo_Click(object sender, EventArgs e)
{
    string Name = "Your Name"; ⓘ
    // lblOutput.Text = "Hello World";
    // lblOutput.Text = txtInput.Text;
    lbxOutput.Items.Add(Name);
    MessageBox.Show("Hello " + Name + " welcome to C#");
}
```

Add your Age

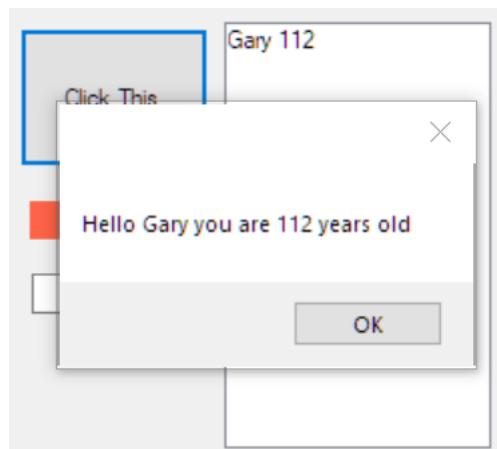
Add and change the following – use your age.

Int (integer) is a variable that holds a whole number 1, 2, or 3 etc., but **not** numbers with decimal points (123.234). Ages are good things to use **Int** with as is counting stuff.



```
1 reference
private void BtnGo_Click(object sender, EventArgs e)
{
    string Name = "Gary"; ⓘ
    int Age = 112; ⓘ
    // lblOutput.Text = "Hello World";
    // lblOutput.Text = txtInput.Text;
    lbxOutput.Items.Add(Name + " " + Age);
    MessageBox.Show("Hello " + Name + " you are " + Age + " years old");
}
```

When using the **MessageBox** you put the text in the brackets () text goes in between Quotations "You are " and a + is used to join the text with the variables.

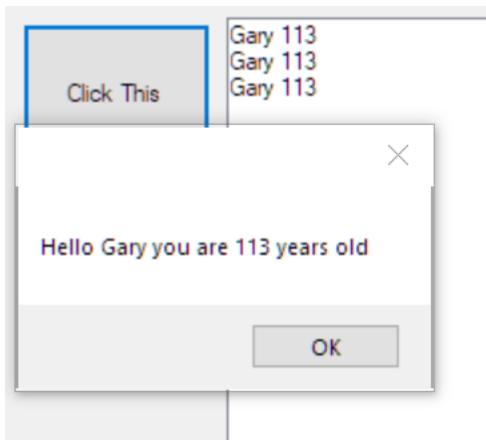


Change the code under your button to look like this `Age = Age + 1;`

```
string Name = "Gary";
int Age = 112;
Age = Age + 1; i
```

The first calculation says take the Age (112) then add 1 to it and make the Age equal to 113.

If you run the code now **IT WILL NOT WORK**. (well it won't increment anyway)



Why not?

Every time you click the button age is equal to 112.

Then you add 1 and print out 113.

Sure that's good, then you click the button again and what happens?

Age is equal to 112, then you add 1 and

Each time you click the button 112 is added back to the variable again. Its like the last click never existed. It's a whole new start. It's an error you will make repeatedly.

Make it and see.....

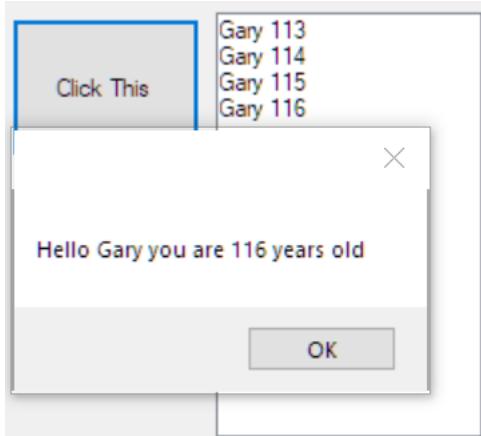
So how do we fix this? **We have to take the variable code out from under the button click, so that it keeps its value all the time.**

So highlight `int Age = 112;` and drag it up to under public **partial class Form1** there it will able to sit at peace and hold its variable values. It is a **Global Variable**

```
public partial class Form1 : Form
{
    int Age = 112; .. i
    1 reference
    public Form1()
    {
        InitializeComponent();
    }

    1 reference
    private void BtnGo_Click(object sender, EventArgs e)
    {
        string Name = "Gary"; i
        i
        Age = ...Age + 1; i
        //    lblOutput.Text = "Hello World";
        //    lblOutput.Text = txtInput.Text;
        lbxOutput.Items.Add(Name + " " + Age);
        MessageBox.Show("Hello " + Name + " you are " + Age + " years old");
    }
}
```

Now run the code...It works!!!



Congrats on your first calculation.

Exercises

Using Buttons, Textboxes, Labels, and Listboxes, complete the following exercises, until you get it ☺

1. Write a C# Sharp program to print the sum of two numbers
2. Write a C# Sharp program to print the result of dividing two numbers
3. Write a C# Sharp program to print the output of multiplication of three numbers which will be entered by the user
4. Write a C# Sharp program that takes four numbers as input to calculate and print the average

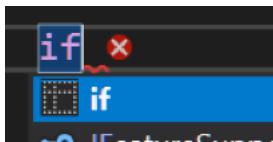
5. Conditionals - If Statements

If Statements are one of the fundamental tools of programming, something you will use in every project.

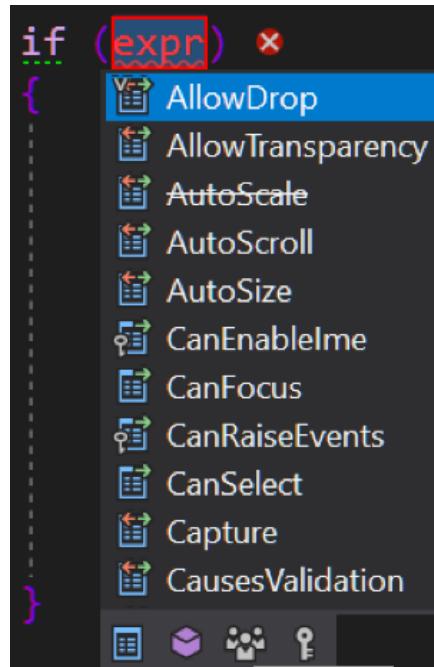
```
if (condition)
{
    Do stuff;
}
```

How to make an IF Statement.

Type in if

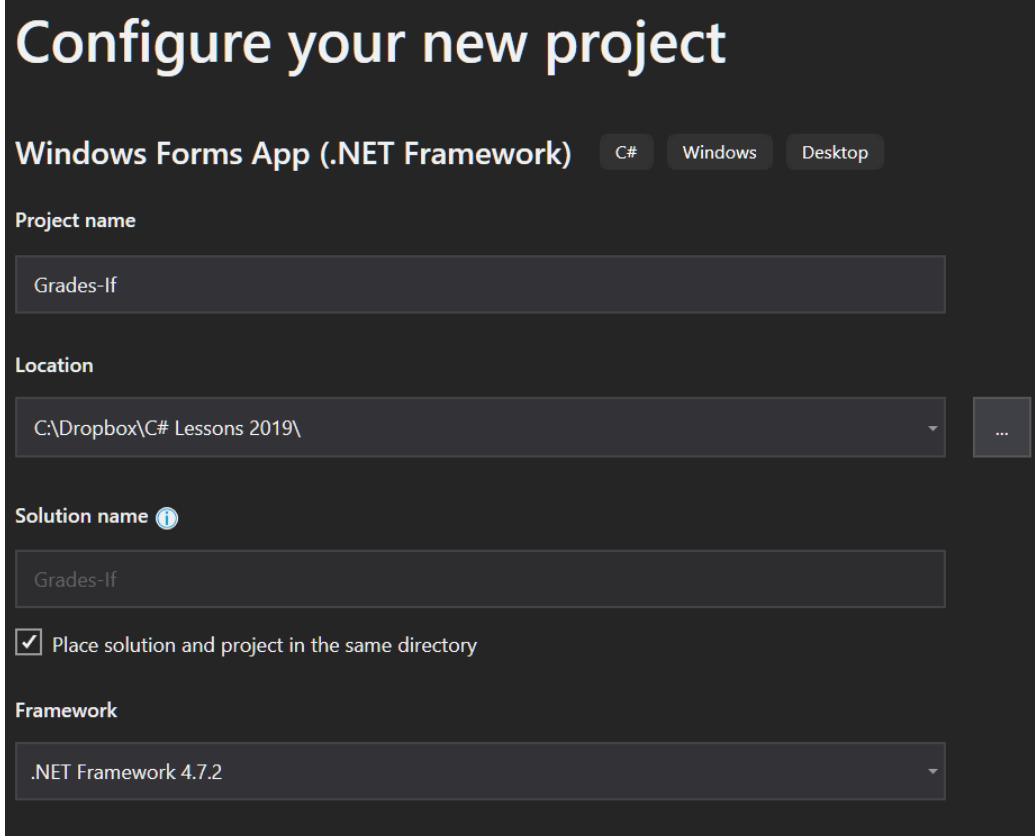


Press **Tab** If you already have some data to use it will appear in the list otherwise just **Tab** again and delete out the text and replace with your own.

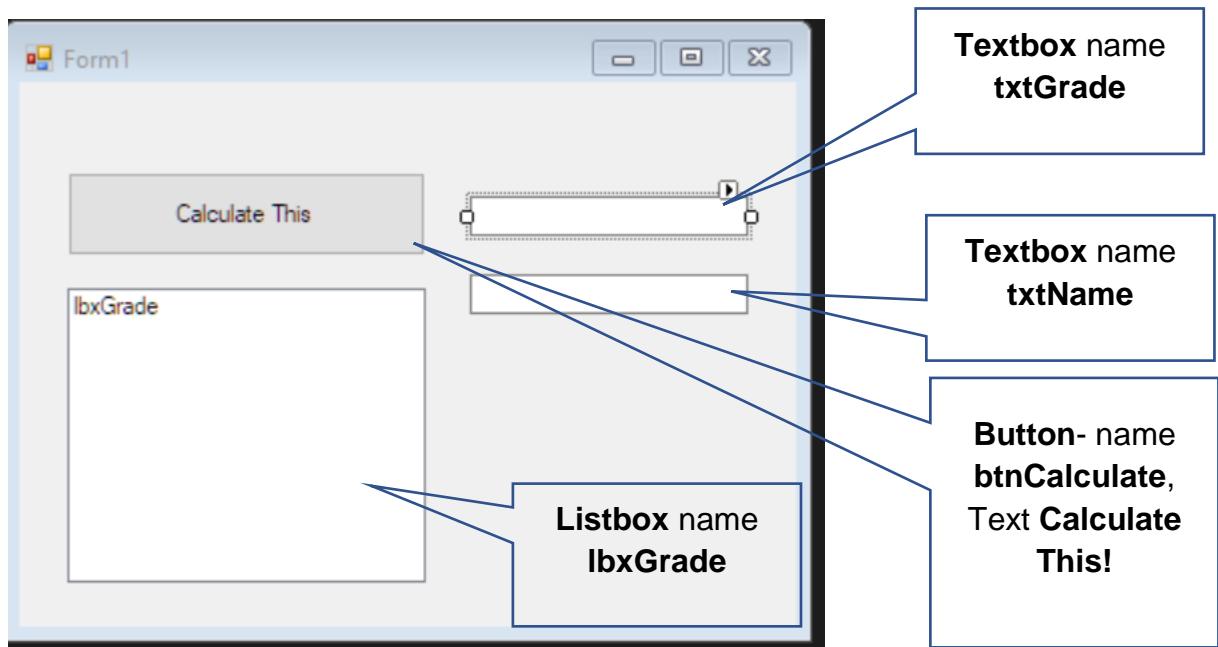


Grades Project using IF

Create a new project, call it **Grades – IF**, and remember to put it in your Projects Folder.



Create a



We want to calculate the grade of a mark when entered into the text box.

Double click on the button to open the code window.

```
1 reference
private void BtnCalculate_Click(object sender, EventArgs e)
{
}
```

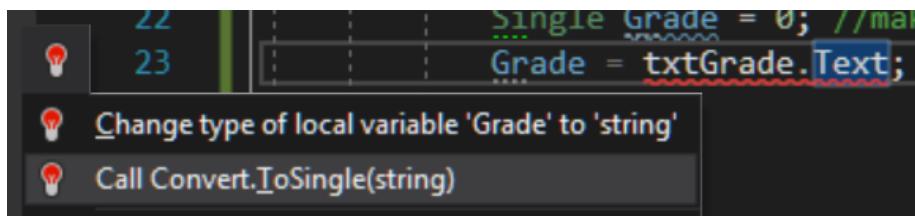
We want to put a grade into the textbox BUT the textbox only holds a string, - text – it can't hold a number. Yet we want to treat it like it's a number and see if it is less than 50.

As a result, we have to **convert the data in the textbox to single**, so that it can be used. See this section on [Converting data](#).

```
1 reference
private void BtnCalculate_Click(object sender, EventArgs e)
{
    Single Grade = 0; //make a variable to hold the grade
    Grade = txtGrade.Text;
}
```

How do we convert the text to a number?

Use the force Luke....



That then gives you the correct code

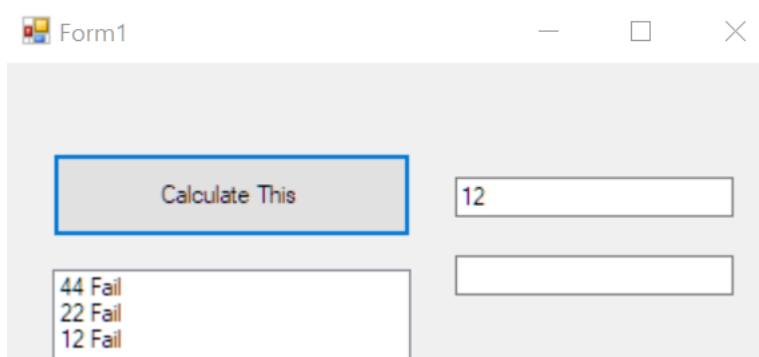
```
private void BtnCalculate_Click(object sender, EventArgs e)
{
    Single Grade = 0; //make a variable to hold the grade
    Grade = Convert.ToInt32(txtGrade.Text);
}
```

Now add in the **if** statement

```
1 reference
private void BtnCalculate_Click(object sender, EventArgs e)
{
    Single Grade = 0; //make a variable to hold the grade
    Grade = Convert.ToInt32(txtGrade.Text);

    if (Grade < 50)
    {
        lbxGrade.Items.Add(Grade + " Fail");
    }
}
```

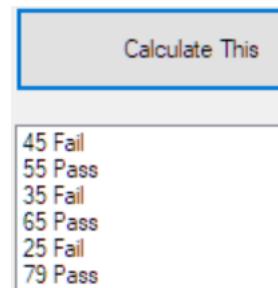
Try it



Now to create the Pass side

```
private void BtnCalculate_Click(object sender, EventArgs e)
{
    Single Grade = 0; //make a variable to hold the grade ⓘ
    Grade = Convert.ToSingle(txtGrade.Text);

    if (Grade < 50) //if the number is less than 50
    {
        lbxGrade.Items.Add(Grade + " Fail");
    }
    if (Grade > 50) //if the number is greater than 50
    {
        lbxGrade.Items.Add(Grade + " Pass");
    }
}
```



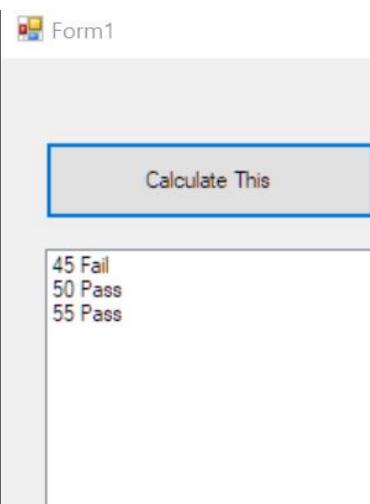
Add in the Greater than 50 If statement and see if it works.

BUT one number doesn't work, which one?

How do we fix it?

```
private void BtnCalculate_Click(object sender, EventArgs e)
{
    Single Grade = 0; //make a variable to hold the grade ⓘ
    Grade = Convert.ToSingle(txtGrade.Text);

    if (Grade < 50) //if the number is less than 50
    {
        lbxGrade.Items.Add(Grade + " Fail");
    }
    if (Grade >= 50) //if the number is greater than 50
    {
        lbxGrade.Items.Add(Grade + " Pass");
    }
}
```



Using Else

Now this works, but it looks clunky. Lets simplify it so that it says "**If it's less then 50 do this else do that.**"

```
if (Grade < 50) //if the number is less than 50
{
    lbxGrade.Items.Add(Grade + " Fail");
}
else //if the number is greater than or equal to 50
{
    lbxGrade.Items.Add(Grade + " Pass");
}
```

Lets add the Name in.

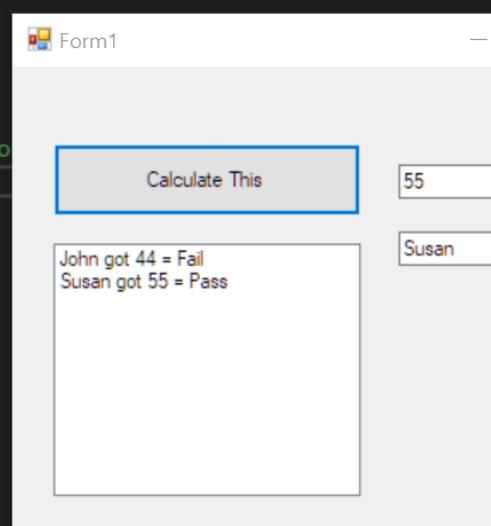
Make sure you add in all the comments in Green. 48 hours later you will forget what you did and might need to look it up again

Why don't we need to convert txtName? (important to know)

```
private void BtnCalculate_Click(object sender, EventArgs e)
{
    Single Grade = 0; //make a variable to hold the grade
    String Name = ""; //make a variable to hold the name

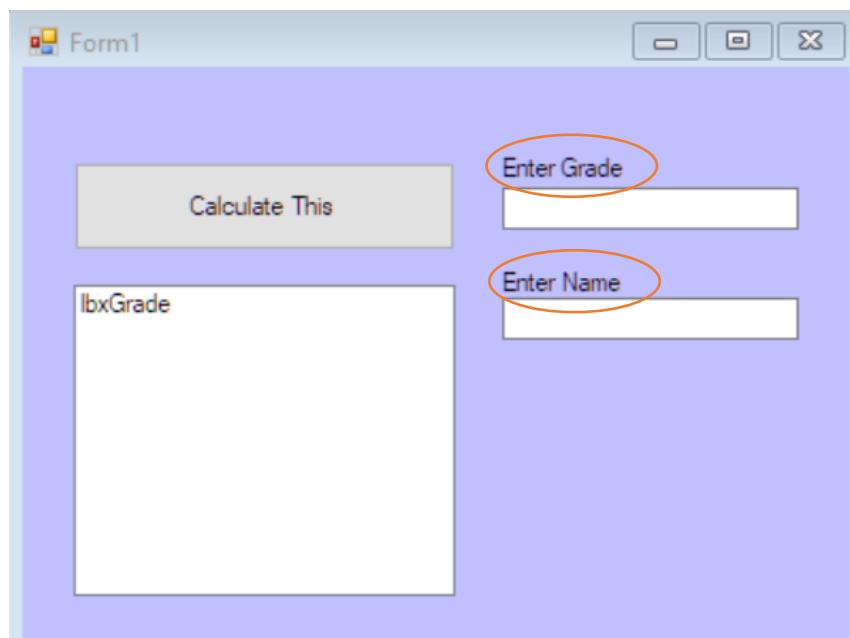
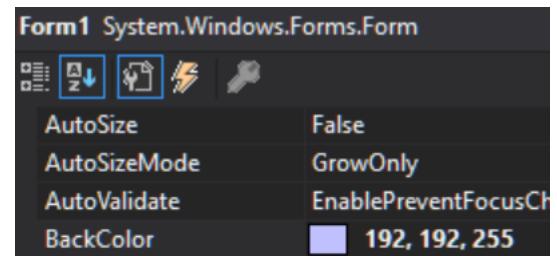
    Grade = Convert.ToSingle(txtGrade.Text); //pass the number to
    Name = txtName.Text; //pass the name to the Name

    if (Grade < 50) //if the number is less than 50
    {
        lbxGrade.Items.Add(Name + " got " + Grade + " = Fail");
    }
    else //if the number is greater than or equal to 50
    {
        lbxGrade.Items.Add(Name + " got " + Grade + " = Pass");
    }
}
```

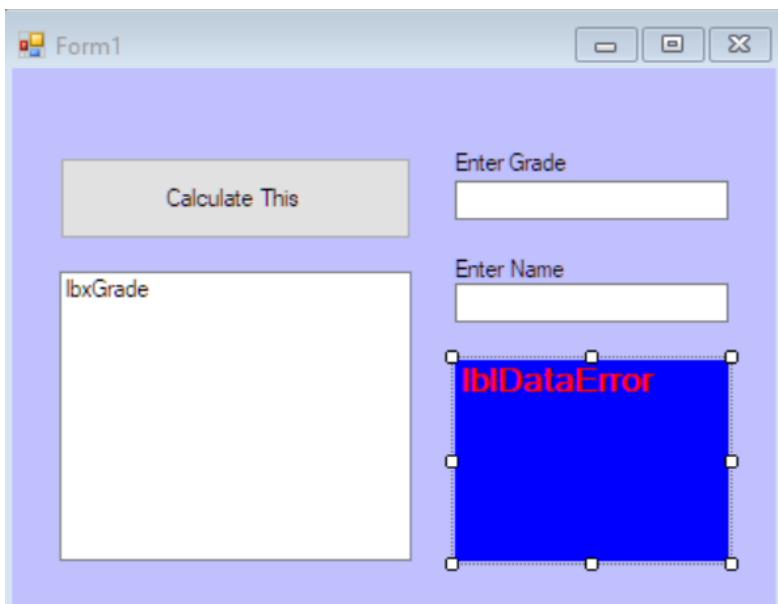


Add in a couple of Labels, and set their text as shown below. Call them **lblName**, **lblGrade**

Set the **BackColor** of the **Form** to a color of your choice



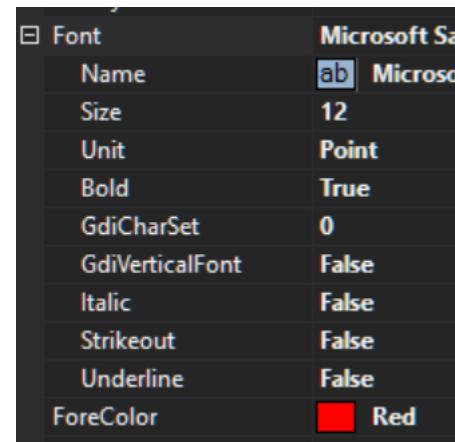
We have a problem. The grade calculator can only have **between 0 and 100**. We need to catch people who have error scores.



Add a new **label** = **lblDataError**

Set **Autosize** = **false**'

Delete the text property (Note I put the label name in the text field so you could see it, but don't do that it looks bad)



Give the Font some nice big warning text and color

Set the Font Backcolor to something as well.

Else IF

Add in the code below We are using an **Else IF** statement

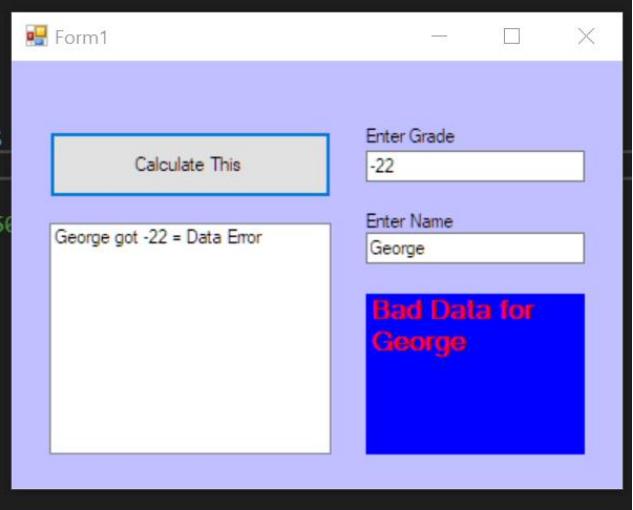
else if allows you to add new nested If statements.

```
Grade = Convert.ToInt32(txtGrade.Text); //pass the number to the Grade
Name = txtName.Text; //pass the name to the Name

if (Grade < 0) //if the number is less than 0
{
    lbxGrade.Items.Add(Name + " got " + Grade + " = Data Error");
    lblDataError.Text = "Bad Data for " + Name;
}

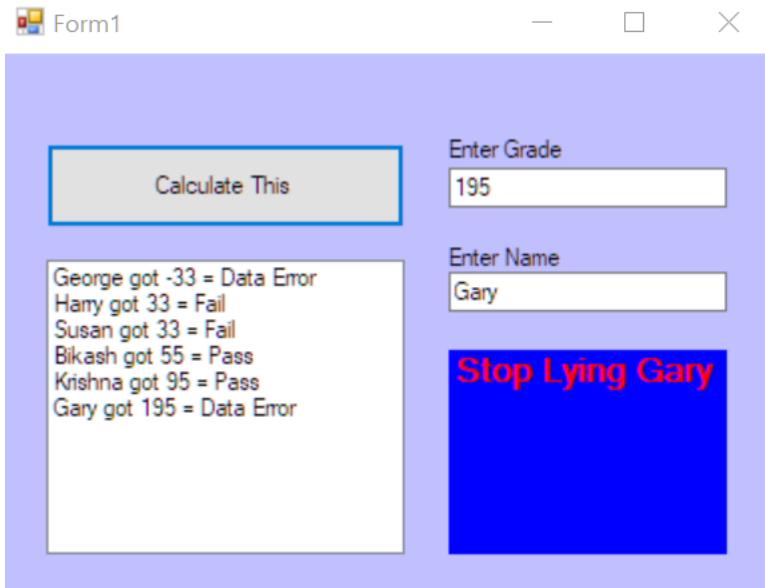
else if (Grade < 50)//if the number is greater than or equal to 50
{
    lbxGrade.Items.Add(Name + " got " + Grade + " = Fail");
    lblDataError.Text = ""; //clear the message box
}

else
{
    lbxGrade.Items.Add(Name + " got " + Grade + " = Pass");
}
```



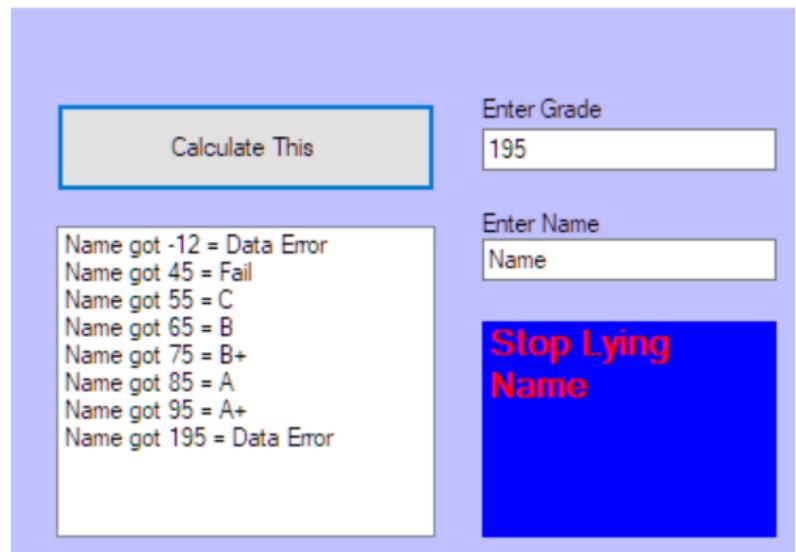
OK you have got this far but you need to catch people who lie and get over 100.

How will you do this? You will need another **else if** in your statement. And change things around a bit.



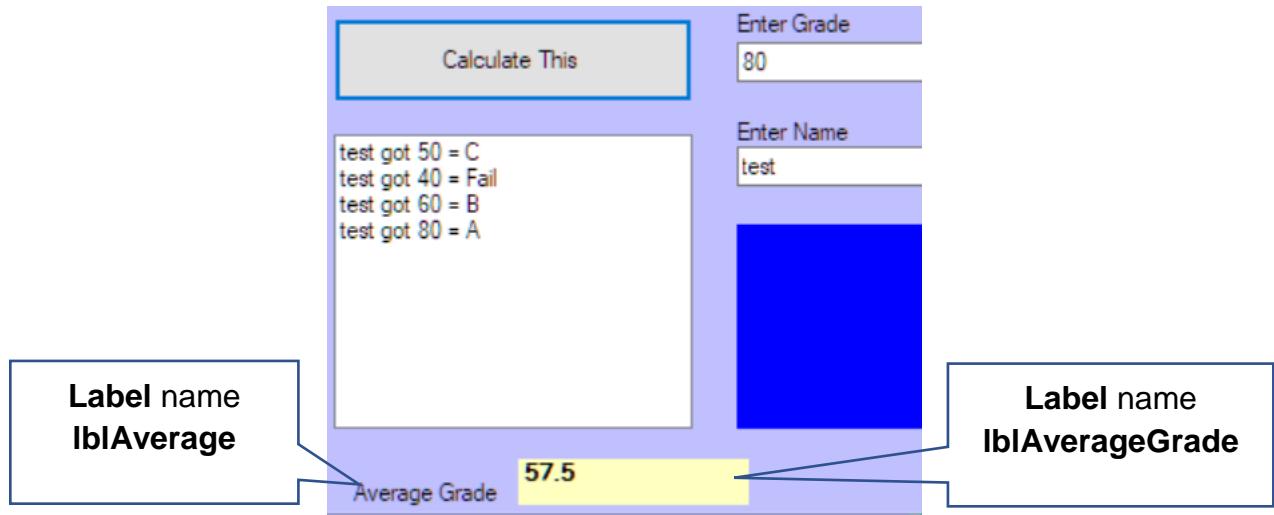
Congratulations! Now can you add in this?

- **Grade < 60 = D**
- **Grade < 70 = C**
- **Grade < 80 = B**
- **Grade < 90 = A**
- **Grade < 100 = A+**



Finding out the Average of Grades entered

Using some simple calculations, we can generate an average score of the numbers entered into your program.



Create a variables **Count** as Integer. **RunningTotal** and **GradeAverageTotal** as Single. Give **Count** a start number of 0.

These must be **global variables** that hold the values and are not reset every time you add a person.

```
public partial class Form1 : Form
{
    int Count = 0; //Increments every time you add a person
    Single RunningTotal = 0, GradeAverageTotal = 0; //totals used in calculation
```

This code goes under the button click before the IF statements

```
private void BtnCalculate_Click(object sender, EventArgs e)
{
    Single Grade = 0; //make a variable to hold the grade
    String Name = ""; //make a variable to hold the name

    Grade = Convert.ToSingle(txtGrade.Text); //pass the number to the Grade
    Name = txtName.Text; //pass the name to the Name

    RunningTotal = RunningTotal + Grade; //add grade to the variable and get a running total
    Count = Count + 1; //count how many times the button is clicked
    GradeAverageTotal = RunningTotal / Count; //divide the total by how many people have been counted
    lblAverageGrade.Text = Convert.ToString(GradeAverageTotal); //convert it to a string and show it
```

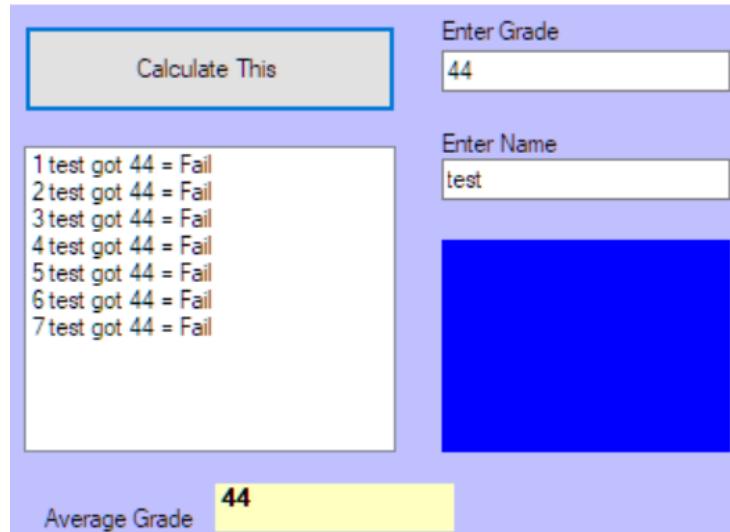
This line `RunningTotal = RunningTotal + Grade;` adds the score in the grade to the `RunningTotal` variable each time you click the button.

This line `Count = Count + 1;` makes the counter increase by 1 each time the button is clicked. This is a really practical piece of code that you use often to count stuff, such as in loops.

In C# is better known as `count +=1;` and even `count ++;`

Exercise: In your ListBox, add a count so that you can see how many entries you have made, such as the pic shows on the right.

Hint: Add Count to your ListBox entry field.



C# - Logical Operators

| Operator | Description | Example |
|----------|--|--------------------|
| && | Called Logical AND operator. If both the operands are non zero then condition becomes true. | (A && B) is false. |
| | Called Logical OR Operator. If any of the two operands is non zero then condition becomes true. | (A B) is true. |
| ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !(A && B) is true. |

OR

In the Grade Calculator we can catch the edge cases using an OR ||

If the Grade is **less** than Zero **OR** the Grade is **greater** than 100.

```
if (Grade < 0 || Grade >100) //if the number is less than 0
{
    lbxGrade.Items.Add(Count + " " + Name + " got " + Grade + " = Data Error");
    lblDataError.Text = "Bad Data for " + Name;
}
```

We would put this first in our calculation, so we can **Fail Fast**. Which is better than going all the way through the calculation before reaching it.

Many new programmers will forget to have the Grade in both places and get this error.

```
if (Grade < 0 || >100)
{
```

Here is a great check to put at the beginning of your code

If txtGrade.Text is empty **OR** txtName.Text is empty, then stop running the code because it will crash.

You could use **txtGrade == ""** instead of `string.IsNullOrEmpty`

```
if (string.IsNullOrEmpty(txtGrade.Text) || string.IsNullOrEmpty(txtName.Text))
{
    return;
}
```

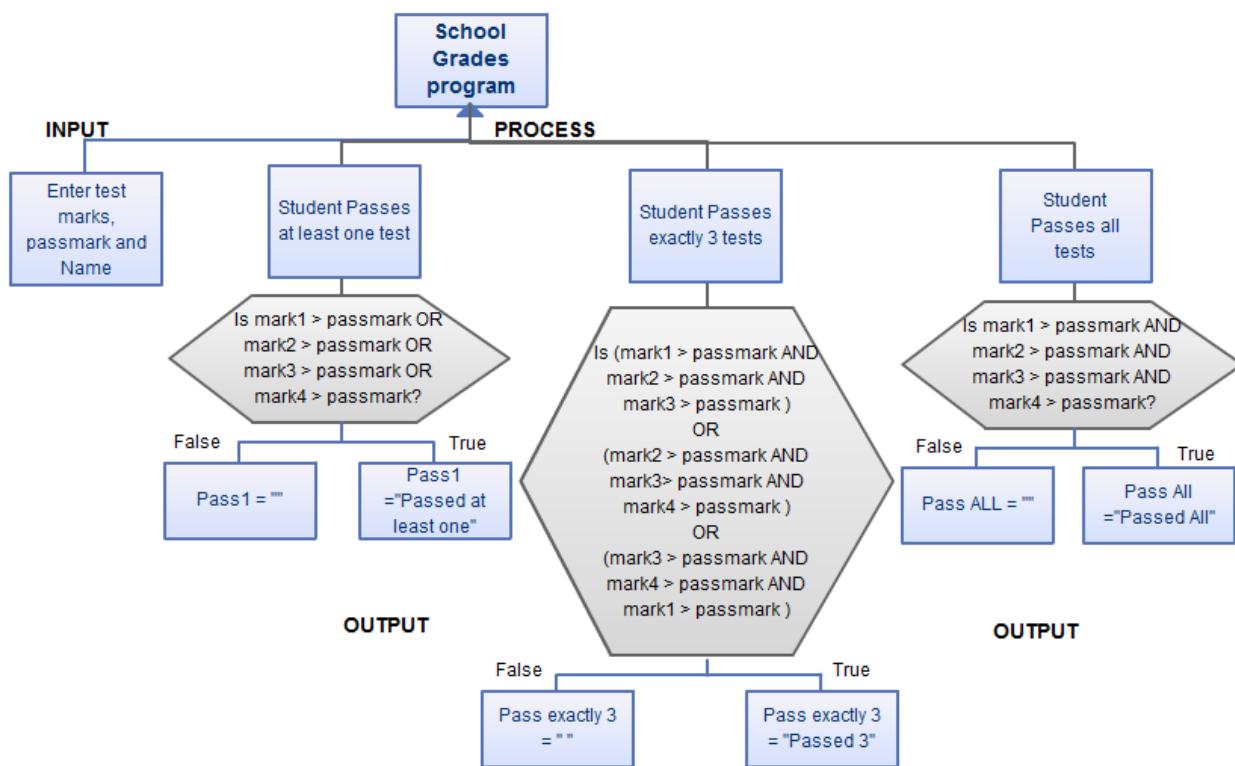
IF with AND and OR – Student grades

A School wants to add grades to a program that enters the student's name, the results from 4 tests, and calculates the following.

Show if the student

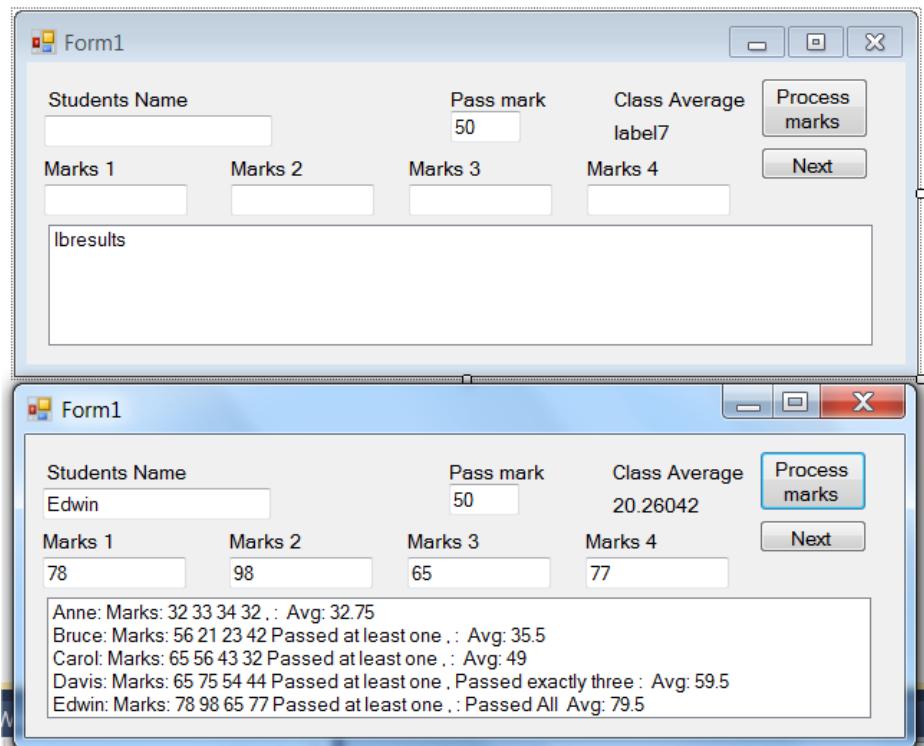
- Passed 1 test
- Passed 3 tests
- Passed ALL tests
- What is his average mark, and what is the average mark of all students to date.

We could output the results in a number of ways, but for ease of programming and to focus on the main point of this exercise, using AND **&&** and OR **||** we will just output to a ListBox.



Once you have created this, show which of the four tests the person has passed for each of the three conditions.

More over the page



These are the variables that I used.

```
float marks1, marks2, marks3, marks4, passmark, studentaverage = 0f;
string name = null;
string pass1, passall, pass3 = null;
```

About that small f ... By default, the number on the right-hand side of the assignment operator is treated as **double**.

Therefore, to initialize a float variable, use the suffix f or F. If you do not use the suffix you will get a compilation error because you are attempting to store a **double** value into a **float** variable.

Note that Anne passed 0, Bruce passed 1, Carol passed 2, Davis passed 3, Edwin passed 4.

Anne: Marks: 32 33 34 32 ,: Avg: 32.75
 Bruce: Marks: 56 21 23 42 Passed at least one ,: Avg: 35.5
 Carol: Marks: 65 56 43 32 Passed at least one ,: Avg: 49
 Davis: Marks: 65 75 54 44 Passed at least one , Passed exactly three : Avg: 59.5
 Edwin: Marks: 78 98 65 77 Passed at least one ,: Passed All Avg: 79.5

Here is the output code. This gives you an idea of the variables you need to make.

```
//output to listbox
lbresults.Items.Add(name + ":" + "Marks: " + marks1 + " " + marks2 + " " + marks3
+ " " + marks4 + " " + pass1 + ", " + pass3 + ":" + passall + " Avg: " +
studentaverage);
```

Calculator Project-

Create a new project. Call it calculator, **remember to make a new folder also called calculator**

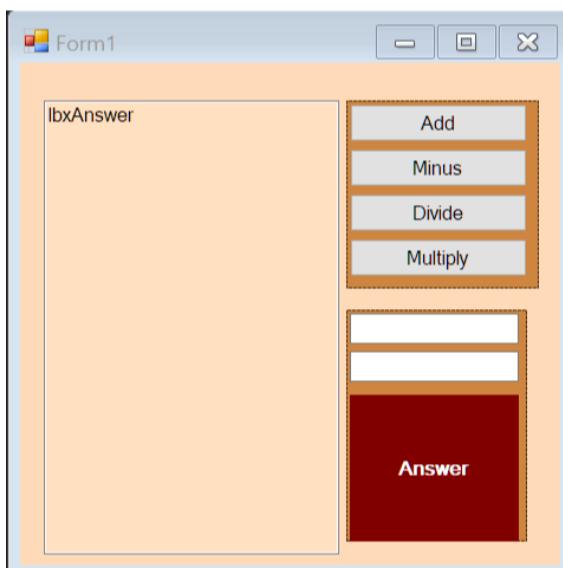
Buttons = btnPlus, btnMinus, btnDivide, btnMultiply. Use your intelligence to work out the text on them

Textboxes = txtNum1, txtNum2

ListBox – lbxAnswer

Label = lblAnswer

For the label, **lblAnswer**, look at the **Properties** of FONT and BACKCOLOR to make it look nice.



Create three variables to hold the numbers. We will use **Float** as it gives decimals. Num1 holds the first number, Num2 holds the second number, and naturally Answer holds the answer.

```
public partial class Form1 : Form
{
    Single sngNum1, sngNum2, sngAnswer;
```

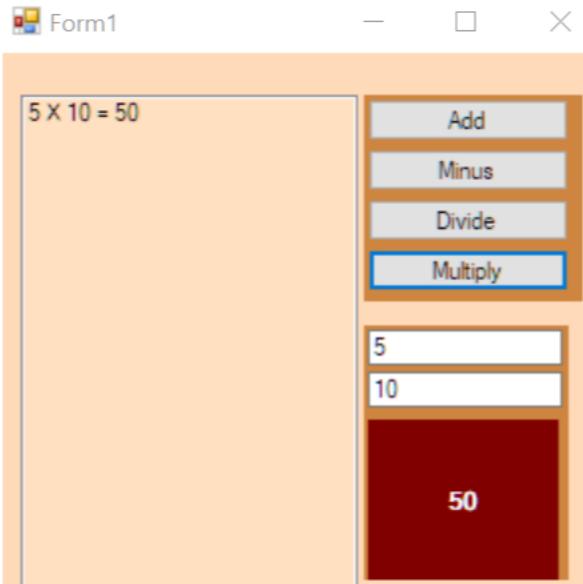
Add the following code under the Multiple button (Double click on the Multiple button to add the code)

Remember `Convert.ToDouble` is used to allow the numbers in the text box, which are strings, to move to the variable num1 which is a float, or double.

```
private void BtnMultiply_Click(object sender, EventArgs e)
{
    //pass numbers across to the variables
    sngNum1 = Convert.ToDouble(txtNum1.Text);
    sngNum2 = Convert.ToDouble(txtNum2.Text);

    //the calculation
    sngAnswer = sngNum1 * sngNum2;
    //output the answer to the label
    lblAnswer.Text = sngAnswer.ToString();
    lbxAnswer.Items.Add(sngNum1 + " X " + sngNum2 + " = " + lblAnswer.Text);
}
```

Run it and see if you get it to look like this



If so, then use the code to make the next 3 buttons +, /, - as well.

Controlling data entry in the calculator

Run the program and use letters, or a combination of letters and numbers in the two data fields.

It's not a happy program.

We need to stop it crashing by checking that only numbers are entered so we will use the Try command again. Type in Try and press TAB

```
try
{
}
catch (Exception exception)
{
    Console.WriteLine(exception);
    throw;
}
```

Drag the Code into the Try section and add a **Messagebox** to the catch to show the error.

Write a better error message

```
private void BtnMultiply_Click(object sender, EventArgs e)
{
    try
    {
        //pass numbers across to the variables
        sngNum1 = Convert.ToSingle(txtNum1.Text);
        sngNum2 = Convert.ToSingle(txtNum2.Text);

        //the calculation
        sngAnswer = sngNum1 * sngNum2;
        //output the answer to the label
        lblAnswer.Text = sngAnswer.ToString();
        lbxAnswer.Items.Add(sngNum1 + " X " + sngNum2 + " = " + lblAnswer.Text);
    }
    catch (Exception exception)
    {
        MessageBox.Show("You have put a non number " + exception);
    }
}
```

6.Types of Methods

Methods are important in keeping your code clean, organized and un-repetitive. Methods also simplify the program so others can easily read it.

- A Method performs the action in its statement block
- A statement block is the code inside of the curly Q brackets.

There are 3 types of Methods,

1. methods that **return no data**, and just run what's inside them,
2. methods that **take in data** and work on it but **don't return anything**, and
3. methods that **take in data**, work on it and **return back data** to the program using Return, Out or Reference

You use methods to break your code into easily read blocks that can be used repeatedly.

What is the method signature?

People take a while to get their head around this part, but you need to know what the parts of the method signature, the first line actually means first.

Here is the simplest method you can make....

```
private void MethodName()
{
}
```

private This method can only be seen on this form, we can't use it elsewhere in the program. (This will become important later). The other option is **public**.

void This says you can't send data back to whatever called the method.

MethodName It's the name of the method **Use verbs or verb phrases** to name methods. Use Pascal case.

() If you want to send data into your method, then it goes in here. Like this below

```
public void MethodName (string FirstName, string LastName)
{}
```

FirstName and LastName are variable what will only live between the { and } of your method.

{...} This holds all of the detail and code in your method, the Body part.

Think of this as a closed box You can't put anything in or out through the lid but you can run whatever is in there.



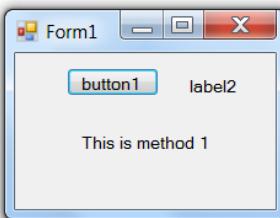
Methods that don't return data

This is simply taking a block of code and running it from a command.

Note our method is named **Message** and it is **private** which means can only be used on this form and **void** which means it doesn't return anything.

```
private void button1_Click(object sender, System.EventArgs e)
{
    message();
}

private void message()
{
    label1.Text = "This is method 1";
}
```



Methods that takes data in

This method takes the text in the msg variable into it and shows it on the label.

Note again that it doesn't return anything therefore it's still **void**. Instead of just printing out the string it could do calculations and change it to whatever you want first.

```
private void button1_Click(object sender, System.EventArgs e)
{
    string msg = " This is method 2";
    message(msg);
}

private void message( string msg)
{
    label1.Text = msg;
}
```



`string msg`



Methods that return data – Pure Methods

This is the third type, a method that takes two numbers (**num1** and **num2**) then adds them together and returns an answer. Note this time that **void** has been replaced by **int** because it's returning a number.

A pure function (or Method in C#) has two useful characteristics:

- **It has no side effects.** The function does not change any variables or the data of any type outside of the function.
- **It is consistent.** Given the same set of input data, it will always return the same output value.

What do we mean by Return data?

Say we have a variable **int answer**.

We can pass to it the answer that comes directly from the method, so we call the method and get the result back into **int answer**.

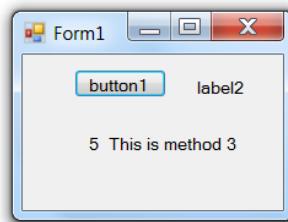
Two things happen at once, firstly it runs the method and then passes the result to **int answer**.

```
int answer = message(num1, num2);
```

When you use **Return** then you must not use void, because it has to return the data with a type to tell the program what it is (int, string, etc.).

```
private void button1_Click(object sender, System.EventArgs e)
{
    string msg = " This is method 3";
    int num1 = 2;
    int num2 = 3;
    int answer = message(num1, num2);
    label1.Text = Convert.ToString(answer) + " " + msg;
}
```

private int message(int num1, int num2)
{
return num1 + num2;
}

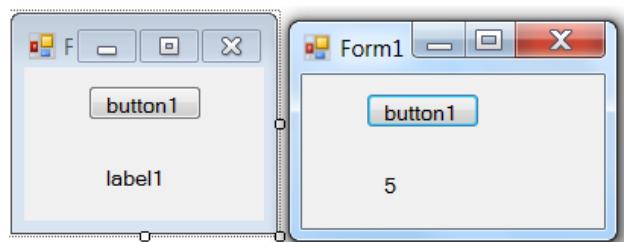


There are three different ways to return a value from a method, that is by **reference**, **out**, or **return**. The following examples look at each of them.

Method Type 3 - Reference

This is another example where the answer is declared as another parameter `int answer = 0;` and passed to the method then passed back as a parameter again. Because we are not using `Return` the method is still `Void`.

```
private void button1_Click(object sender, System.EventArgs e)
{
    int num1 = 2;
    int num2 = 3;
    int answer = 0;
    //calling our addition method
    addition(num1, num2, ref answer);
    label1.Text = Convert.ToString(answer);
}
private void addition(int num1, int num2, ref int answer)
{
    answer = num1 + num2;
}
```



The `num1`, `num2` and `answer` are declared under the button click. When the program reaches `addition(num1, num2, ref answer);` it passes the 3 variables to the method `addition`. The numbers are then calculated in that method and the answer is passed back to `answer` and then shown on the label.

Value and Reference types

```
private void button1_Click(object sender, System.EventArgs e)
{
    int num1 = 2;
    int num2 = 3;
    int answer = 0;
    //calling our addition method
    addition(num1, num2, ref answer);
    label1.Text = Convert.ToString(answer);
}
static void addition(int num1, int num2, ref int answer)
{
    answer = num1 + num2;
}
```

We first initialize our variables, then we call our method.

When calling a method, you need to include in the parenthesis the names of the variables you what passed to the method. You may be asking yourself, "but what about that ref thing before answer?"

The word **Val** is short for "By Value". What it means is that you are **passing a copy** of a variable to your Subroutine, or in other words **creating a new instance** of the variable. You can make changes to the copy and the original will not be altered.

Any changes made to the value of this variable have no effect on the value of the original variable that was passed in. By Val (by value) means that the called routine uses the value purely for input, to start a "**one way**" **conversation**.

Ref is the alternative. This is short for By Reference. This means that you are not handing over a **copy** of the original variable but pointing to the **original** variable.

It's not a copy, this is it! Any changes you make to this variable also change its value outside the routine.

When you do not include ref in front of the argument the compiler assumes that you want what it passed in by value, but when you include ref, you are telling the compiler that you want to reference the memory address to the method to be modified.

Now, in our method we need to declare the variables we need to use. The format is, "data type" "variable name". If you have more than one variable, like we do, you separate each one with a comma. You can name the variables virtually anything, it doesn't need to be the same is the variables in Main. The important thing is that you keep the same order.

You need to initialize your variables before you pass them to the method. As you can see I set answer to 0, but once the method returns the correct answer it will overwrite that. We will get to a way around having to initialize variables later.

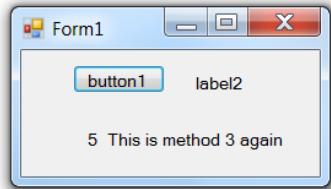
Method Type 3 – OUT (rarely used)

Here is another version using the **Out** so that you don't need to use Ref. Out acts just like a ref, but I do not need to initialize answer

```
private void button1_Click(object sender, System.EventArgs e)
{
    string msg = " This is method 3 again";
    int num1 = 2;
    int num2 = 3;
    int answer = 0;
    message(num1, num2, out answer);

    label1.Text = Convert.ToString(answer) + " " + msg;
}

private void message( int num1, int num2, out int answer)
{
    answer = num1 + num2;
}
```



New C#7 Methods within Methods (Local Functions).

In a very JavaScript manner, imagine we want a method, that is only going to be used with another method. Well we can now put one inside of the other.

Why would we want to do this? Because we can.

Here is a good explanation of it by Resharper.

<https://blog.jetbrains.com/dotnet/2017/02/17/state-union-resharper-c-7-vb-net-15-support/>

```
//here is an outside method
    public static int Factorial(int number)
    {
//oh look another method, it only can be used in here, it doesn't live outside of the
Factorial Method
        int Multiply(int num)
        {
            return (num == 1)
                ? 1
                : num * Multiply(num - 1);
        }
//here it is being called
        return Multiply(number);
    }
```

But wait! There's more!

Named Parameters in your methods

New to C# you can have named parameters and default parameters.

Named arguments free you from the need to remember or to look up the order of parameters in the parameter lists of called methods. The parameter for each argument can be specified by parameter name.

For example, a function that calculates body mass index (BMI) can be called in the standard way by sending arguments for weight and height by position, in the order defined by the function.

Names can make it easier to read your code

```
CalculateBMI(height: 180, weight: 75);
CalculateBMI(weight: 75, height: 180);
}

static int CalculateBMI(int weight, int height)
{
    return (weight * 703) / (height * height);
}
```

Default Parameters

Also new to C# you can have default parameters that activate if the parameter isn't called in the code.

```
CalculateBMI(height: 190, weight: 90); //overrides the default
CalculateBMI(175); //height is the default called, not weight
CalculateBMI(); //both defaults called
}

static int CalculateBMI(int weight = 180, int height = 75)
{
    return weight / (height * height);
}
```

<http://msdn.microsoft.com/en-us/library/dd264739.aspx>

Returning more than one value from a method

Many times you don't just want to return one piece of data, you want to return a whole bunch of data. There are a number of ways you can do that.

For example, you could use an **Array** or a **List**, or a **Dictionary** to hold data and then get it out afterwards. However, the most popular way is to make a **class** or structure and use that.

Using a Class to return multiple data.

We haven't got this far in the manual yet, but it's a good reference for later, this is very popular and goes under many different names such as a **DTO** or Data Transfer Object.
(THERE ARE MANY WAYS TO DO THIS)

1 First make a class

```
public class StudentDetails
{
    public string FirstName;
    public string LastName;
}
```

2 Then create an instance of the class so that you have a **studentName** object, make sure it's added in at the top of your code so you can use in all sub sections.

```
public partial class Form1 : Form
{
    StudentDetails studentName = new StudentDetails();
```

3 Then make your method replacing **void** with **StudentDetails**.

This tells the compiler that a **StudentDetails** object will be passed through. The pass the data through instead of "**John**" you might have **txtFirstName.text** or some other input means.

```
public StudentDetails Students()
{
    studentName.FirstName = "John";
    studentName.LastName = "Smith";
    return studentName;
}
```

4 The finally run the Method in the class and studentName will hold the information and can be used, in this case sent to the label.

```
private void btnClassTest_Click(object sender, EventArgs e)
{
    Students(); //call the method
    lblFullName.Text = string.Format("{0} {1}", studentName.FirstName,
    studentName.LastName);
    //send the name to the label
}
```

Using a Tuple to return multiple data.

A tuple is a data structure that provides an easy way to represent a single set of data. **A tuple is a finite ordered list of elements** The term originated as an abstraction of the sequence: single, double, triple, quadruple, quintuple, sextuple, septuple, octuple, ..., *n*-tuple [Wikipedia](#)

Tuples allow us to,

- Create, access, and manipulate a data set
- Return a data set from a method without using out parameter
- Pass multiple values to a method through a single parameter

.NET framework supports tuples with up to seven elements. However, they all return back with **Item1**, **Item2**, **Item3** etc.

Example: This is all you have to do!

```
private void btnTupleTest_Click(object sender, EventArgs e)
{
    // Create a 3-tuple
    var student = new Tuple<string, string, string>("Ms", "Susan", "Smith");

    // Display student info
    lblFullName.Text = string.Format("{0} {1} {2}", student.Item1, student.Item2,
    student.Item3);
}
```

<http://www.c-sharpcorner.com/article/tuples-in-c-sharp/>

New Tuples in C# 7 – custom names

<https://visualstudiomagazine.com/articles/2017/01/01/tuples-csharp-7.aspx>

You can get rid of **Item1**, **Item2**, etc. by naming the return types.

Instead of having a method return type of `string, string, int`

```
public (string, string, int) TupleStudentReturn(int year)
```

Give each type a name, and then you can use that name instead of Item1, Item2, etc.

```
public (string FirstName, string LastName, int Year) TupleStudentReturn(int year)
```

Here is the method taking in the Year (2017) and returning back the name

```
// tuple return type
public (string FirstName, string LastName, int Year) TupleStudentReturn(int year)
{
    string FirstName = string.Empty;
    string LastName = string.Empty;
    if (year == 2017)
    {
        FirstName = "John";
        LastName = "Smith";
    }
    return (FirstName, LastName, year);
}
```

Here is the button click, and instead of `student.Item1`, `student.Item2`, `student.Item3` we can use the names in the Method Signature. `studentyear.FirstName`, `studentyear.LastName`, `studentyear.Year`

```
private void btnTupleTest_Click(object sender, EventArgs e)
{
    var studentyear = TupleStudentReturn(2017);
    // Display student info
    lblFullName.Text = string.Format("{0} {1} {2}", studentyear.FirstName,
    studentyear.LastName, studentyear.Year);
}
```

From now on use methods in every exercise

(I haven't done so but you must)

Methods are very important they keep your code tidy and allow it to be easily scanned

7. Unit Testing – Test Driven Development (TDD)

An easy way to check your code and practice coding without the fuss of breaking your program.

<http://www.visualstudio.com/en-us/get-started/create-and-run-unit-tests-vs.aspx>

Basic Unit Testing for C# Developers



TDD ensures quality code from the start. Developers are encouraged to write only the code needed to make the test pass and thus fulfil the requirement. If a method has less code, it's only logical that the code has fewer opportunities for error.

TDD ensures a high degree of fidelity between the code and the business requirements. If your requirements are written as tests, and your tests all pass, you can say with a high degree of confidence that your code meets the needs of the business.

TDD helps keep unused code out of the system. Most developers have written applications in which they designed interfaces and wrote methods based on what might happen. This leads to systems with large parts of code or functionality that are never used.

This code is expensive. You expend effort writing it, and even though that code does nothing, it still has to be maintained. It also makes things cluttered, distracting you from the important working code. TDD helps keep this parasite code out of your system.

In the simplest terms, a unit test is a test designed to test one unit of work. In this case “one unit of work” means one requirement for one method.

Stylistically, unit tests can be written in a variety of ways, but all unit tests share some common characteristics. They are:

- Isolated from other code
- Isolated from other developers
- Targeted
- Repeatable
- Predictable

<http://blog.stevensanderson.com/2009/08/24/writing-great-unit-tests-best-and-worst-practises/>

“TDD is a design process, not a testing process TDD is a robust way of designing software components (“units”) interactively so that their behaviour is specified through unit tests.”

There are **3 ways** we will incorporate Unit Testing in this course, from basic to the most comprehensive. These steps follow the coding ability of the students.

First, we will use **stand-alone unit tests**, where you copy code manually back and forth from the unit test to the main code base. This obviously has its limitations and we use it primarily as a teaching tool. In this scenario, Unit Testing makes a really handy ‘scratch pad’ for quickly debugging and creating code.

The second method will be to **connect to Functions**, passing data from a function in the code base to the Unit Test. This is more practical and is also a great way to practice creating functions.

Finally, the third way is to incorporate Unit Tests **into the Classes** where we just instantiate the class in the unit Test and create fake data. This is the true environment for Unit Testing,

Fools Don’t write Unit Tests

TradeMe tells us about their use of Unit Tests.

Hi Gary,

Thanks for your enquiry, we're always happy to help out in the wider community – especially when it helps foster technical knowledge in New Zealand.

I'm afraid whoever told you that number was wrong, yesterday alone we ran **1,855,467** unit tests.

Now don't get me wrong that's not ~2 million run in a single batch, that's **145** separate run-throughs of the Trade Me unit tests suite (which on average contains 12796 tests – the exact number varies).

The way we work at Trade Me is that we use a [Mercurial](#) repository to manage and version our code.

Using this each team / developer can work on their own version of the code base (called a branch) in isolation and when complete they merge their changes back into the main repository.

We need to have confidence that developers are not accidentally introducing issues with their changes so anytime they make a change to their branch our Continuous Integration Server (we use [TeamCity](#)) will compile their changes and execute the full unit test suite over the code.

1. Developer creates a 'branch' from the main repository 'trunk'
2. Developer makes changes to code
3. Developer commits changes to their branch
4. CI Server detects changes automatically
5. CI Server builds and runs unit tests
6. Developer can now merge their branch back into the main trunk

Going back to figure of 145 run-throughs, we have at any time 60+ branches active within Trade Me.

This is partly because we have a lot of developers working on the system, and partly because priorities change – a developer may begin the day working on one feature and switch to something else later on in the day leaving the initial work unfinished.

The average number of suite executions per branch is somewhere between 2 and 3 – the number should generally be low as we large changes are really hard to manage and merge back into the main trunk.

TeamCity has a number of build Agents (separate servers) which it delegates work to – this allows it to run different suites in parallel hence how we can run so many things throughout the day.

So how do we organise these tests? The way we do it, and the generally unofficially industry standard, is that we would create one unit test per class within the system so for example we might have (sorry I cannot provide a screenshot for legal reasons):

- MyProject
 - MyClass.cs
 - MyOtherClass.cs
 - SomeNamespace
 - ClassWithinNamespace.cs
- MyTestProject
 - MyClassTests.cs
 - MyOtherClassTests.cs
 - SomeNamespace
 - ClassWithinNamespaceTests.cs

The rule is that our test project shadows the structure of the project(s) we're testing.

We use [NUnit](#) as our test framework, and we supplement this with [Moq](#) for Mocking, and [AutoFixture](#) for anonymous data creation.

As you might expect for a company that's been around for ~16 years we have a lot of variance in our tests but a typical test might look like:

```
[TestFixture]
public class MyClassTests {
    [Test]
    public void When_null_is_provided_null_is_returned() {
        // Arrange
        var testInstance = new MyClass();

        // Act
        var result = testInstance.DoSomething(null);

        // Assert
        Assert.That(result, Is.Null);
    }
}
```

There's more to it than that, and I've only given you a very high level overview but hopefully this helps explain some things. Don't hesitate to ask if you have further questions.

Regards

Andy McDowell

Development Manager Trade Me

Calculator Methods we are Unit Testing in this example

Depending on how we made our methods they may be in the Form1 code, or in their own Class. In this case I moved it to the Operations Class BUT yours might be elsewhere. Bug your Tutor over this.

```
public class Operations
{
    //set your method to public so you can see it in the other code
    public string Divide(string Num1, String Num2)
    {
        Single sngNum1, sngNum2, sngAnswer;
        try
        {
            //pass numbers across to the variables
            sngNum1 = Convert.ToSingle(Num1);
            sngNum2 = Convert.ToSingle(Num2);

            //the calculation
            sngAnswer = sngNum1 / sngNum2;
            return sngAnswer.ToString();
        }

        //if it fails show a message
        catch (Exception exception)
        {
            return "This calculation has borked";
        }
    }

    /// <summary>
    /// Adds Two numbers together
    /// </summary>
    /// <param name="Num1">String number</param>
    /// <param name="Num2">String Number</param>
    /// <returns></returns>
    public string Add(string Num1, String Num2)
    {
        Single sngNum1, sngNum2, sngAnswer;
        try
        {
            //pass numbers across to the variables
            sngNum1 = Convert.ToSingle(Num1);
            sngNum2 = Convert.ToSingle(Num2);

            //the calculation
            sngAnswer = sngNum1 + sngNum2;
            return sngAnswer.ToString();
        }

        //if it fails show a message
        catch (Exception exception)
```

```
        {
            return "This calculation has borked";
        }
    }

public string Minus(string Num1, String Num2)
{
    Single sngNum1, sngNum2, sngAnswer;
    try
    {
        //pass numbers across to the variables
        sngNum1 = Convert.ToSingle(Num1);
        sngNum2 = Convert.ToSingle(Num2);

        //the calculation
        sngAnswer = sngNum1 - sngNum2;
        return sngAnswer.ToString();
    }

    //if it fails show a message
    catch (Exception exception)
    {
        return "This calculation has borked";
    }
}

public string Multiply(string Num1, string Num2)
{
    Single sngNum1, sngNum2, sngAnswer;
    try
    {
        //pass numbers across to the variables
        sngNum1 = Convert.ToSingle(Num1);
        sngNum2 = Convert.ToSingle(Num2);

        //the calculation
        sngAnswer = sngNum1 * sngNum2;
        return sngAnswer.ToString();
    }

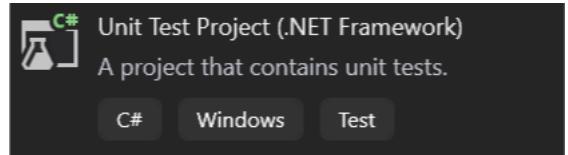
    //if it fails show a message
    catch (Exception exception)
    {
        return "This calculation has borked";
    }
}
```

Unit Testing the Calculator

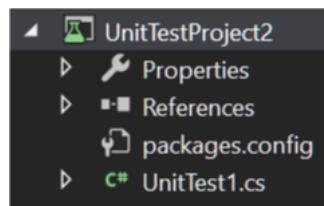
NOTE: Visual Studio 2017 and Visual Studio 2019 have slightly different interfaces and responses. **Be ready to check yours.** Black screenshots are 2019

Create a new Unit Test.

Right Click on **Solution**  Solution 'Calculator2019' and go **Add → New Project**

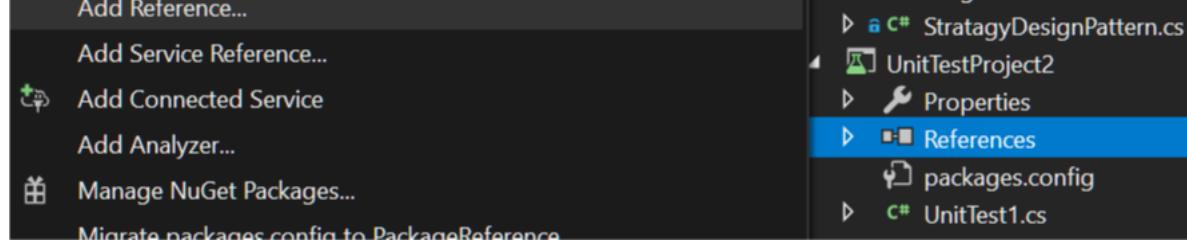


Choose Unit Test Project and make sure you have the **.NET Framework** version



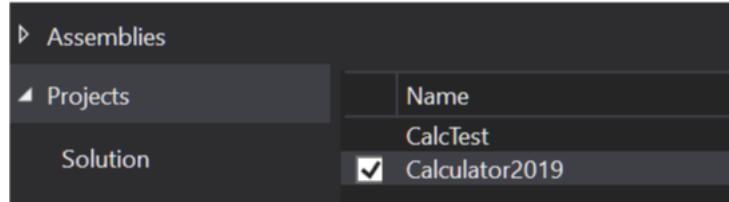
This will generate a new Project →

Add in a Reference to the Calculator project **when you are in the Unit Test Project – Right click on References and Add Reference**



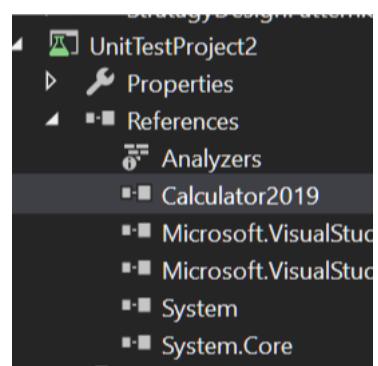
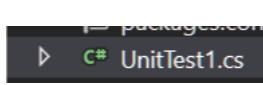
Click on **Projects** and add your Calculator Project – You will only have one to add. Then click OK.

Reference Manager - UnitTestProject2



You will see that a reference has been added to your project

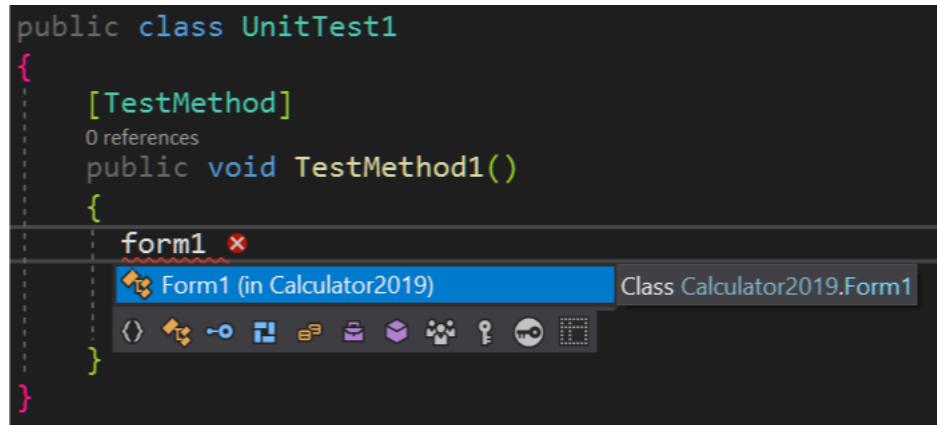
Open up your Unit Test1.cs



Then we need to make a link to your methods that you want to test.

Depending on how you wrote your program they will be either in your **Form** code or on **their own class**.

Assuming they are in your Form code. Type `Form1 myForm1 = new Form1();` as below



```
public class UnitTest1
{
    [TestMethod]
    0 references
    public void TestMethod1()
    {
        form1 x
        Form1 (in Calculator2019) Class Calculator2019.Form1
        { } 
    }
}
```

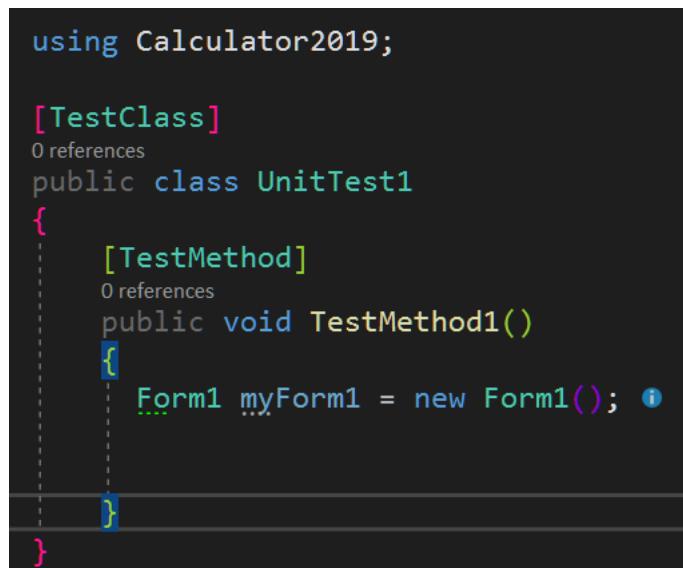
The screenshot shows a code editor with the following code:

```
public class UnitTest1
{
    [TestMethod]
    0 references
    public void TestMethod1()
    {
        form1 x
        Form1 (in Calculator2019) Class Calculator2019.Form1
        { } 
    }
}
```

A tooltip is displayed over the word "form1", showing "Form1 (in Calculator2019)" and "Class Calculator2019.Form1". Below the tooltip are several small icons representing different actions like copy, paste, and search.

See that it knows its in your calculator project.

See that the `using Calculator2019;` has been added to your code.

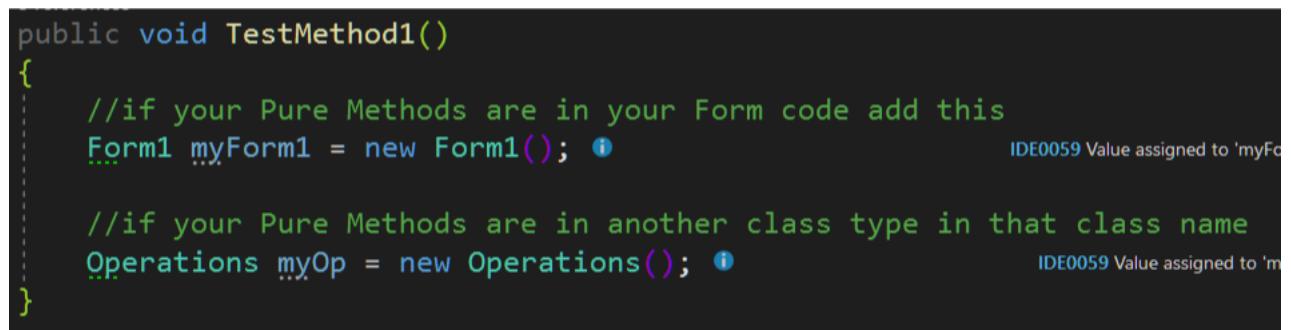


```
using Calculator2019;

[TestClass]
0 references
public class UnitTest1
{
    [TestMethod]
    0 references
    public void TestMethod1()
    {
        Form1 myForm1 = new Form1(); 
    }
}
```

The screenshot shows the same code as above, but now the "using Calculator2019;" statement is present at the top of the file.

If your Methods are in **another class** then just follow below, instantiate that class name in your code and don't bother with the Form1 code.



```
public void TestMethod1()
{
    //if your Pure Methods are in your Form code add this
    Form1 myForm1 = new Form1(); IDE0059 Value assigned to 'myFo

    //if your Pure Methods are in another class type in that class name
    Operations myOp = new Operations(); IDE0059 Value assigned to 'm
}
```

The screenshot shows the following code:

```
public void TestMethod1()
{
    //if your Pure Methods are in your Form code add this
    Form1 myForm1 = new Form1(); IDE0059 Value assigned to 'myFo

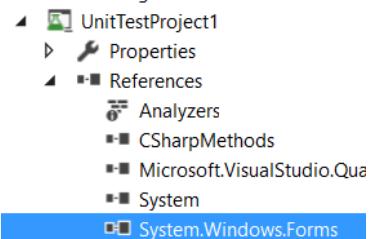
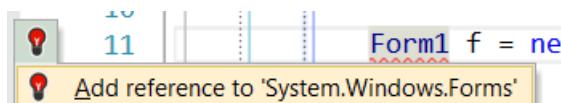
    //if your Pure Methods are in another class type in that class name
    Operations myOp = new Operations(); IDE0059 Value assigned to 'm
}
```

Two diagnostic messages are visible: "IDE0059 Value assigned to 'myFo" and "IDE0059 Value assigned to 'm".

You might have the error below with the lightbulb. Just add the reference if you do.



Just click it to choose the option



Then you will see the reference added to the list.

Now we can write our Unit Test

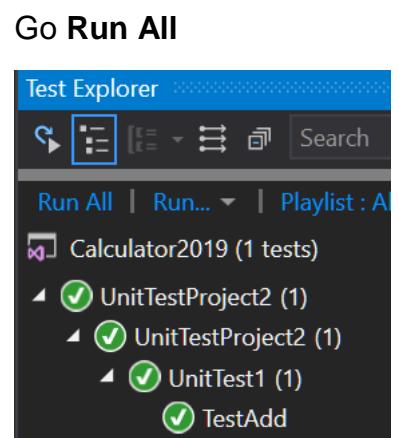
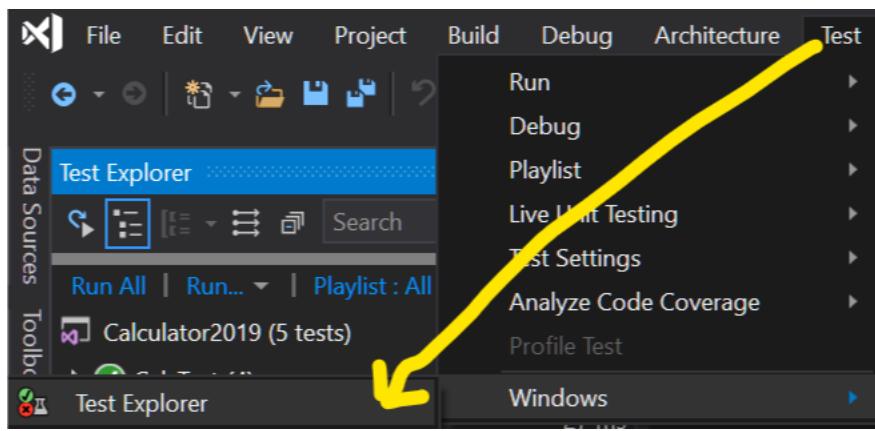
Rename **TestMethod1** to **TestAdd** or the name of your Method that you are Testing (**TestDivide**, **TestMultiply** etc)

```
[TestMethod]
0 references
public void TestAdd()
{
    //Arrange - we need to make an instance of the class
    Operations op = new Operations();

    //Act - we need to push some data through our method and get an output
    var Actual = op.Add("3", "3");
    //Assert - we need to check the the output we get is what we expected
    Assert.AreEqual(expected: "6", Actual);
}
```

Before we run our tests lets turn on the **Test Explorer**

Go **Test – Windows – Test Explorer**



A Happy Green Dot appears on the Method

```
[TestMethod]
10 references
public void TestAdd()
```

Create the rest of the Unit Tests

```
using System.Net.NetworkInformation;
using Calculator2019;

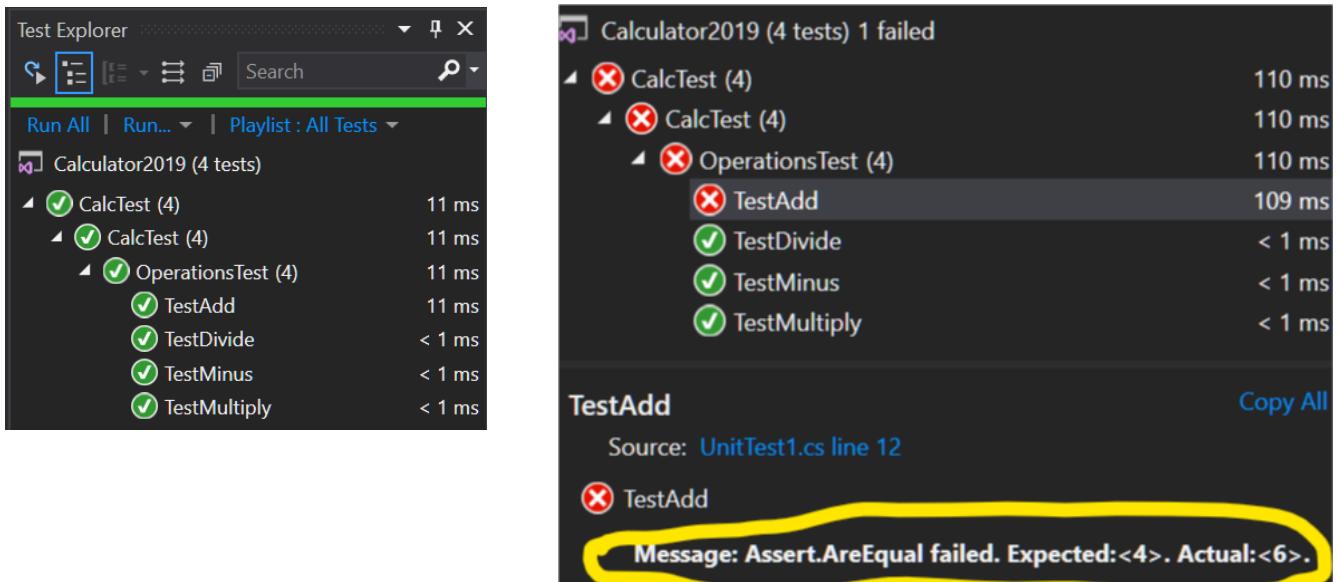
[TestClass]
public class OperationsTest
{
    [TestMethod]
    public void TestAdd()
    {
        //Arrange - we need to make an instance of the class
        Operations op = new Operations();
        //Act - we need to push some data through our method and get an output
        var Actual = op.Add("3", "3");
        //Assert - we need to check the output we get is what we expected
        Assert.AreEqual("6", Actual);
    }

    [TestMethod]
    public void TestDivide()
    {
        //Arrange - we need to make an instance of the class
        Operations op = new Operations();
        //Act - we need to push some data through our method and get an output
        var Actual = op.Divide("3", "3");
        //Assert - we need to check the output we get is what we expected
        Assert.AreEqual("1", Actual);
    }

    [TestMethod]
    public void TestMinus()
    {
        //Arrange - we need to make an instance of the class
        Operations op = new Operations();
        //Act - we need to push some data through our method and get an output
        var Actual = op.Minus("3", "3");
        //Assert - we need to check the output we get is what we expected
        Assert.AreEqual("0", Actual);
    }

    [TestMethod]
    public void TestMultiply()
    {
        //Arrange - we need to make an instance of the class
        Operations op = new Operations();
        //Act - we need to push some data through our method and get an output
        var Actual = op.Multiply("3", "3");
        //Assert - we need to check the output we get is what we expected
        Assert.AreEqual("9", Actual);
    }
}
```

Once we have completed our Unit Tests we can see them in the Test Explorer



If we have an **error** in our Unit Test then look at the bottom and it will tell you what is wrong. So is the error in the code, or in the Test itself.

Back in your Code you will see this....

After you run the tests you get the little **1/1 passing** showing.

```

13 //set your method to public so you can see it in the test
14 public string Divide(string Num1, string Num2)
15 {
16     Single sngNum1, sngNum2, sngAnswer;
17     try
18     {
19         //pass numbers across to the variables
20         sngNum1 = Convert.ToSingle(Num1);
21         sngNum2 = Convert.ToSingle(Num2);

```

This means you can run the unit tests from your method code just to check it.

It's fantastic!

Temp Converter - Pure Methods and Unit Tests

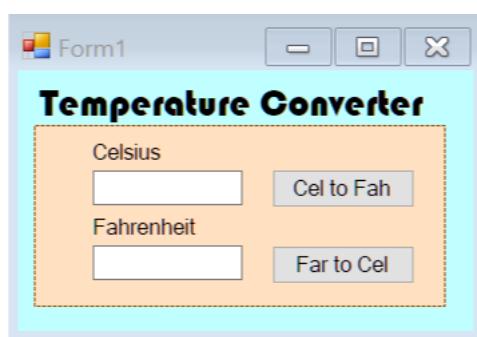
Create a temperature converter. You enter a number as either Celsius or Fahrenheit and the program converts from one to the other.

Below is a working program, use the code, or make your own and do the following

1. Create Pure Methods like [this](#) for both the calculations.
2. Remove the global variables, get all code outside of the button clicks apart from 1 line you will need.
3. Use the Unit Tests to connect directly into the Pure Methods.

```
public partial class Form1 : Form
{
    //todo: Get rid of the global variables, create pure methods to convert temps, Unit Test both methods.

    private Single Fahrenheit, Celsius;
    public Form1()
    {
        InitializeComponent();
    }
    /// <summary>
    /// Button that converts Celsius to Fahrenheit
    /// </summary>
    private void btnCelToFah_Click(object sender, EventArgs e)
    {
        Celsius = Convert.ToSingle(txtCel.Text);
        //work out the fahrenheit
        Fahrenheit = (float)(Celsius * (9.0 / 5.0) + 32);
        //show the fahrenheit on the form
        txtFah.Text = Convert.ToString(Fahrenheit);
    }
    /// <summary>
    /// Button that converts Fahrenheit to Celsius
    /// </summary>
    private void btnFahToCel_Click(object sender, EventArgs e)
    {
        Fahrenheit = Convert.ToSingle(txtFah.Text);
        Celsius = (float)((5.0 / 9.0) * (Fahrenheit - 32.0));
        txtCel.Text = Convert.ToString(Celsius);
    }
}
```



READ THE INFORMATION OVER THE PAGE!!!

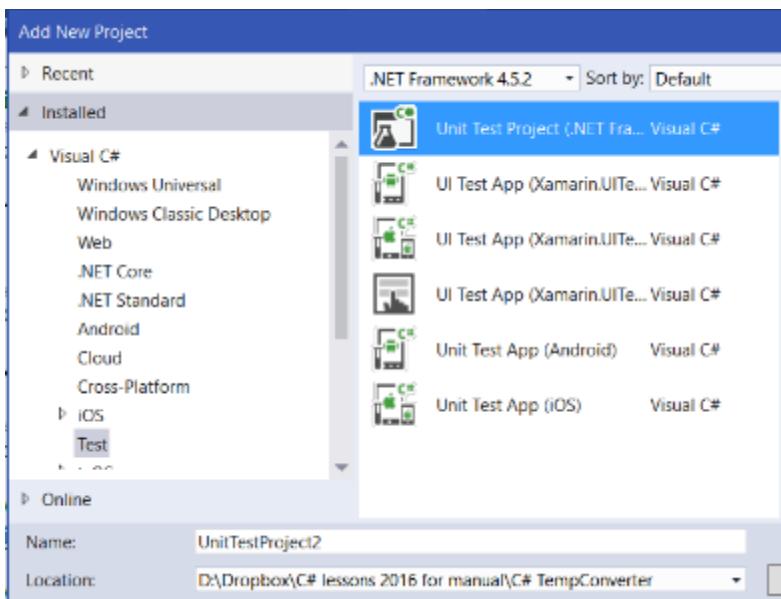
The problem in this case is that we might not know what the expected outcomes are. Find a website such as this <http://www.mathsisfun.com/temperature-conversion.html> and get some standard conversion numbers to use.

We want two unit tests

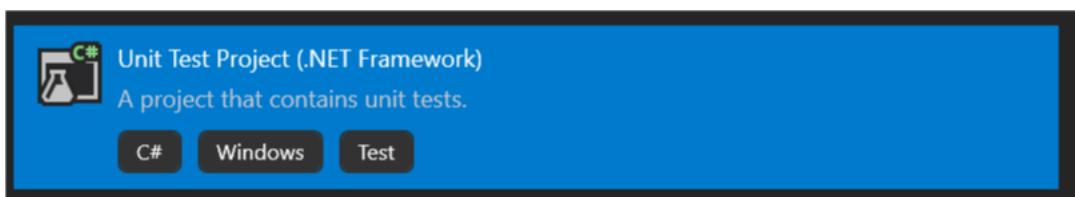
Attach your Unit Test by

1 Creating a Unit Test project

Right Click on Solution go **Add New Project**

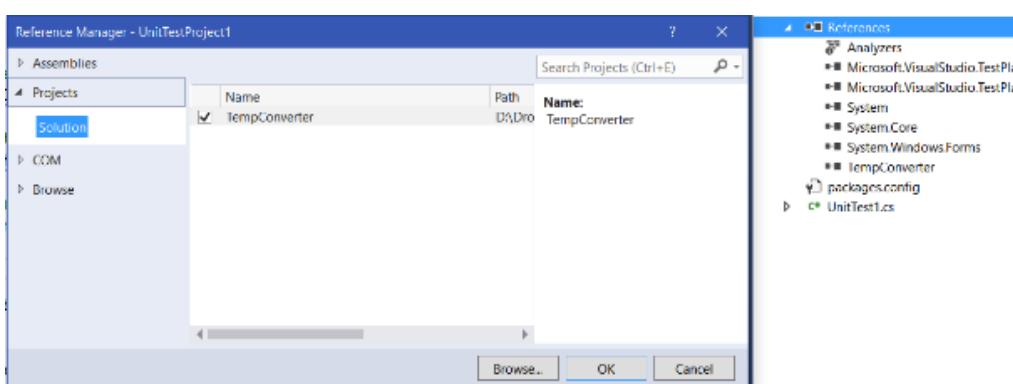


Or

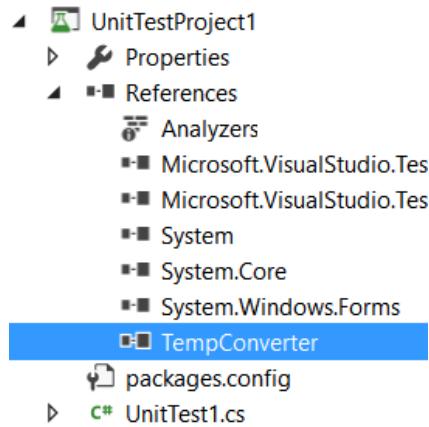


2 Connect the Unit Test project to the TempConvert Project

Right Click on References, Go **Add Reference**.



This will add the reference to the other project, so it can be accessed, otherwise they are just independent from each other.



In your Unit test **instantiate** your form1. Then you can use it as **myForm1.CellToFah** if you have made them public.

Instantiating means – “**Create a new instance of the class**” Form1 is just a class, so we need to make an instance of it so we can access it. Its structure is **Class myClass = new Class()**

myClass is just a name, you can name it anything you want but naturally you will want to remind yourself where the instantiation come from.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using TempConverter;

namespace UnitTestProject1
{
    [TestClass]
    public class UnitTest1
    {
        Form1 myForm1 = new Form1();

        [TestMethod]
        public void CelToFah()
        {
            // Test implementation
        }
    }
}
```

Body Mass Index (BMI) exercise

Create a Body Mass Index program. Go on, I know you can. Then Unit Test it.

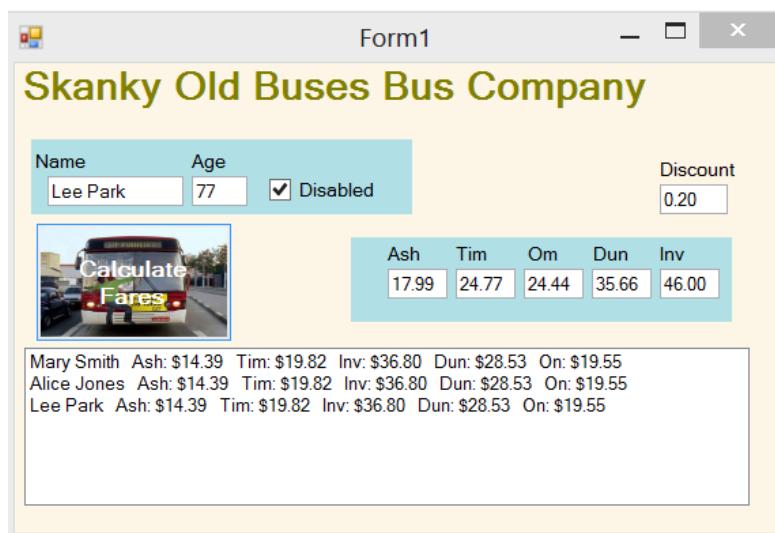
http://en.wikipedia.org/wiki/Body_mass_index

Skanky Old Bus exercise

The Comfortable Coach Company runs buses from Christchurch to Invercargill.

They give a 20% discount to:

- Children under 12
- Adults over 64
- Disabled people



So, if you are under 20 years old, OR over 64 OR disabled you get 20% off the full fare. Otherwise you don't and you have to pay the full fare.

Create a program that calculates the cost for each person on the list.

Don't hard code your values for fares and discount in case you want to change them. Put them into the text boxes as shown in the image.

Use radio buttons or tick boxes (or whatever means you want) to select the 3 options that determine a discount and show the price for each destination.

| A | B | C | D | E | F | G | H | |
|--|----|------------|-------------------------|------------------|---------------|---------------|----------------|--|
| 1 Comfortable Coaches Bus Company | | | | | | | | |
| 2 | | | | | | | | |
| 3 Reduction for: | | | | | | | | |
| 4 Children under | 12 | | Reduction of 20% | | | | | |
| 5 Adults over | 65 | | | | | | | |
| 6 Disabled | | | | | | | | |
| 7 | | | | | | | | |
| Destination | | | | | | | | |
| 8 Passenger List | | Age | Disabled | Ashburton | Timaru | Oamaru | Dunedin | |
| 9 Full Fare | | | | \$17.99 | \$24.77 | \$29.44 | \$35.66 | |
| | | | | | | | \$46.00 | |
| 10 | | | | | | | | |
| 11 Mary Smith | | 37 | Yes | 14.39 | 19.82 | 23.55 | 28.53 | |
| 12 Alice Jones | | 6 | No | 14.39 | 19.82 | 23.55 | 28.53 | |
| 13 Lee Park | | 77 | Yes | 14.39 | 19.82 | 23.55 | 28.53 | |
| 14 Tahu Jones | | 11 | No | 14.39 | 19.82 | 23.55 | 28.53 | |
| | | | | | | | 36.80 | |

Can you add another checkbox named weekend so that people getting a discount of 20% get 25% if they travel on the weekend?

If it's a weekend those with a Discount get 25% off, otherwise everyone gets 20% discount.

Unit Test your calculations to check that they are working

8.Debugging your program

It's exciting

Occasionally, when you run your program, there is a *software bug*. This is the most exciting part of computer programming: discovering an error in something you have made and then solving the problem.

Break mode halts the operation of an application and gives you a snapshot of its condition at any moment.

Variable and property settings are preserved, so you can analyse the current state of the application and enter changes that affect how the application runs. When an application is in break mode, you can:

- Modify code in the application.
- Observe the condition of the application's interface.
- Determine which active procedures have been called.
- Watch the values of variables, properties, and statements.
- Change the values of variables and properties.
- View or control which statement the application will run next.
- Manually control the operation of the application.



Note You can set breakpoints and watch expressions at design time, but other debugging tools work only in break mode.

In your code click on the line at the left where you want the program to stop, insert a breakpoint. Sometimes I make a new line of meaningless code such as int aaa=0; and use that as my breakpoint. If you make a breakpoint inside a loop, then the loop doesn't run.

```

57 //print out on the label
58
59 lblchange.Text = coincount[0] + " Twenty Dollars" + "\r\n" + coincount[1] + ?
At Form1.cs, line 59 character 13 (C_Sharp_calculate_change_loop.Form1.btnCalculate_Click(object sender, EventArgs e), line 40) |
One Dollars" + "\r\n" + coincount[4] + " Fifty Cents" + "\r\n" + coincount[5] + ?
Twenty Cents" + "\r\n" + coincount[6] + " Five Cents" + "\r\n" + coincount[7] + " One
Cents";

```

Then run your program like normal, and when the code reaches the place where you have a breakpoint all the objects are listed in the Locals window. It's fascinating and fantastic for coding.

Step into your code and see the values change with Step into.

Otherwise you can step through a line with Step Over

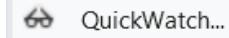


Use Step out when you don't want to execute the code, but get out of the method you are in.



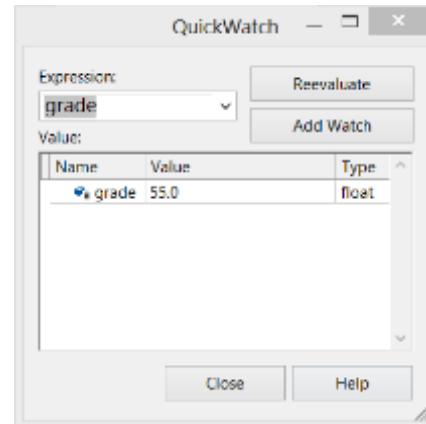
Watch this video for some great tips. <http://www.youtube.com/watch?v=QCPt9aOcd98>

Quickwatch



Right click on a variable you want to watch and choose Quickwatch

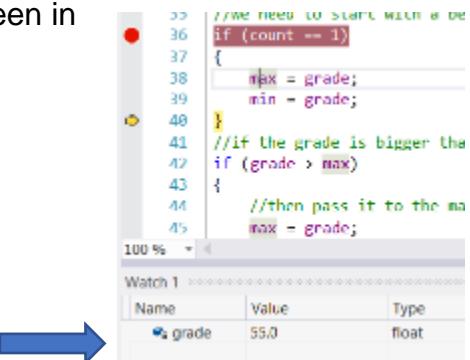
This will open the following window with your variable (Grade) in it.



AddWatch

Right click on a variable and choose AddWatch

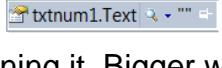
You will see your variable added to the bottom of the screen in the Watch area.



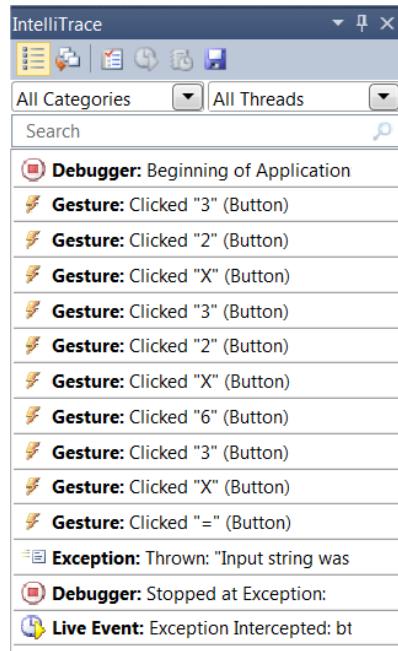
Help when the program crashes

When the program crashes, as happening here because the text field has no content, Intellisense tries to give you some help.

Mousing over the error will bring up a small text box showing the name of the object, the value and other features, such as pinning it. Bigger windows with often obscure messages appear as well see below.

```
private void btnequals_Click(object sender, EventArgs e)
{//pass the number to the second variable num2
    num2 = Convert.ToSingle(txtnum1.Text);
    txtnum1.Text = string.Empty; 
```

FormatException was unhandled
Input string was not in a correct format.



Help also shows on the right showing what you were doing when the program crashed.

Watch this Telerik video on Exception handling. Very good information.

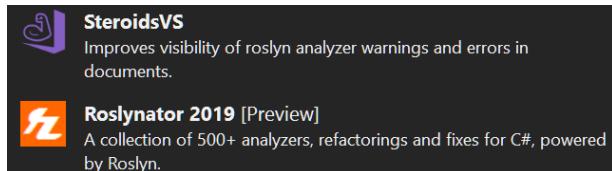
<https://www.youtube.com/watch?v=itkB-Xv6O-0>

Visual Studio 2019 Error Checking Tools

VS2019 comes with a wide range of built in tools to help you debug.

Roslyn

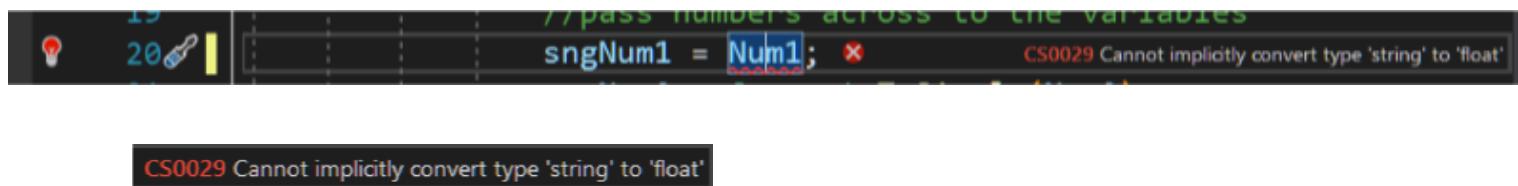
There are built in Roslyn tools and extensions that make them easier to understand.



Simply Mousing over the error will often tell you what the problem is.

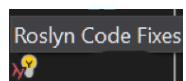
```
//pass numbers across to the variables
sngNum1 = Num1; *
sngNum2 = Cc
    ↪ (parameter) string Num1
//the calcul
sngAnswer =
    ↪ Cannot implicitly convert type 'string' to 'float'
return sngAn
    ↪ Cannot convert source type 'string' to target type 'float'
```

Roslyn gives you a quick hint as to what the error is.



Roslynator gives you the error ID – **CS0029**. **SteroidVS** gives the error message format to easily understand it.

Clicking on the error will bring up a range of tools to help you



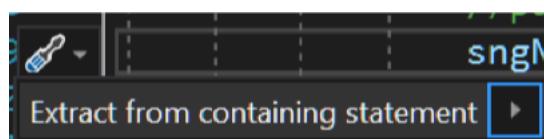
The Yellow Lightbulb with the Funny red Squiggle is also from Roslyn

You'll see an error light bulb icon if there's a red squiggle indicating an error and Visual Studio has a fix available for that error.

The yellow light bulb icon indicates there are **actions available that you should do to improve your code**.

The error light bulb icon indicates there's an **action available that fixes an error in your code**.

Some help is less helpful ...

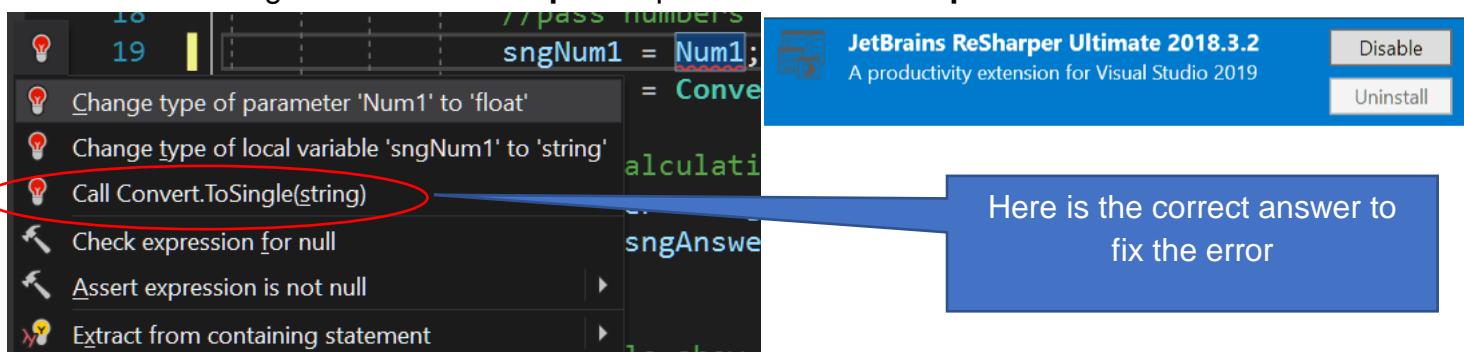


It's a Screwdriver!

The screwdriver icon  indicates just that there are actions available to change the code, **but you shouldn't necessarily use them.**

ReSharper

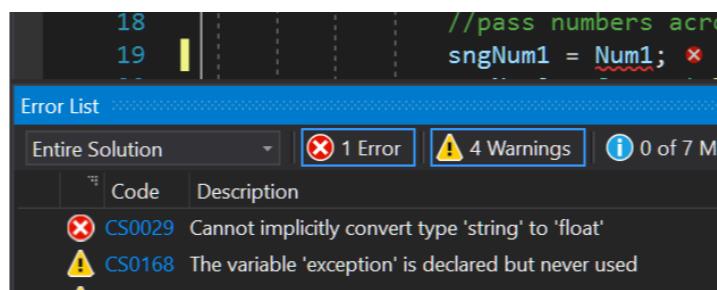
The red Lightbulb is a **ReSharper Helper** from the **ReSharper Extension**.



Error List Window

The error list Window also shows you what the error is and where it is.

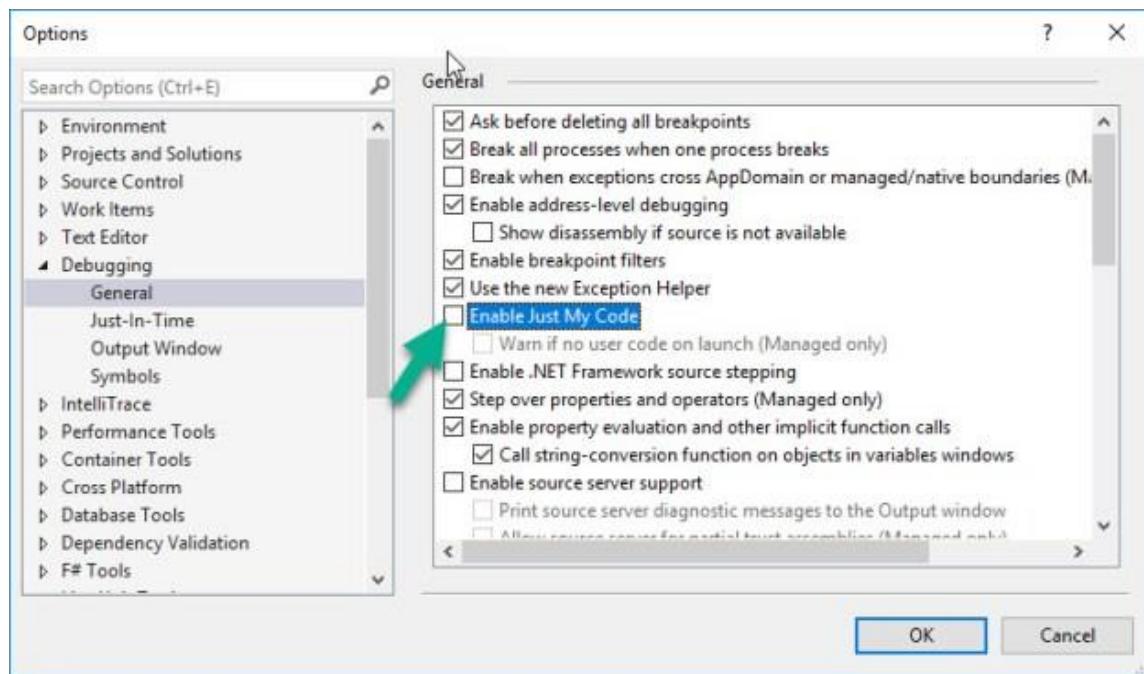
Double click on the error message to go to the code.



Enable and Disable Just My Code

By default, VS will break only on Exceptions thrown in your code. However, there are often problems that happen due to exceptions in one of your references. You might have used the API wrong or there's a bug in some NuGet package you are using.

To break on exceptions from all code, go to **Tools -> Options** and in **Debugging**, uncheck “Enable Just My Code”.



Now you'll break on all exceptions, including in the methods of your references. You probably won't see the code since you don't have the symbols of that library, but you will see the Call Stack and Exception type.

There's a way to debug inside the referenced library with a neat free tool called **DnSpy**.

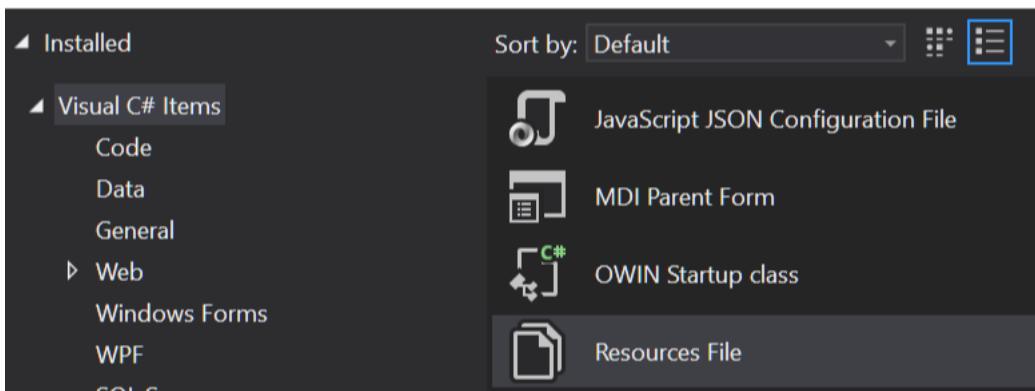
You'll probably want to keep **Enable Just My Code** enabled most of the time. Otherwise, Visual Studio will keep loading symbols (which is time consuming) and you might start breaking on an endless stream of exceptions that you don't care about.

9. Adding images to a Resource file

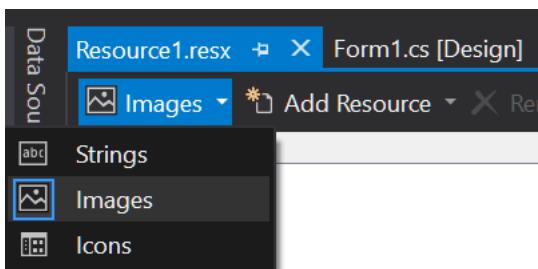
We can add images to our program, and use them on our buttons, and elsewhere. Adding images is an important tool in creating a good-looking program.

Right click on your project, go **Add Then New Item**. Then choose **Resource File**. The resource file will be where you keep all your images.

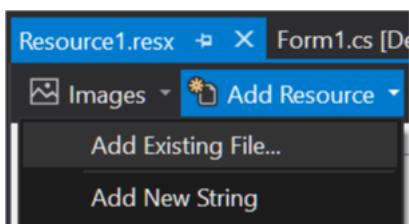
Add New Item - CalcTest



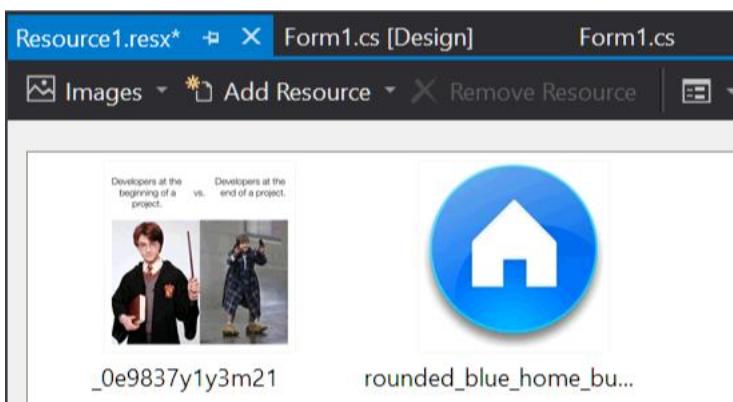
Change the top left from **Strings** to **Images**.



Go to the next menu **Add Resource** and choose **Add Existing File**.

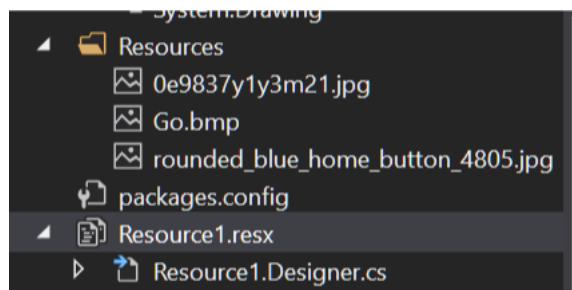


Choose **Add Existing File**, and then find it.

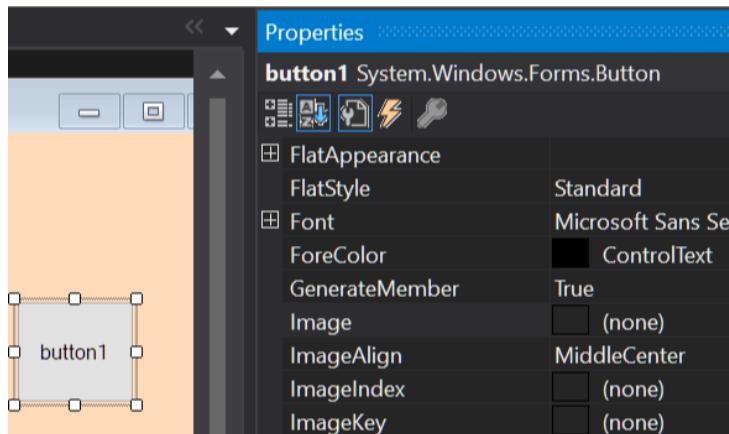


In your Solution Explorer you will see a new folder Resources with your images in them.

Below the Resources Folder is the **Resource Designer**, and if you double click on **Resource1.resx** it will open up the designer again.

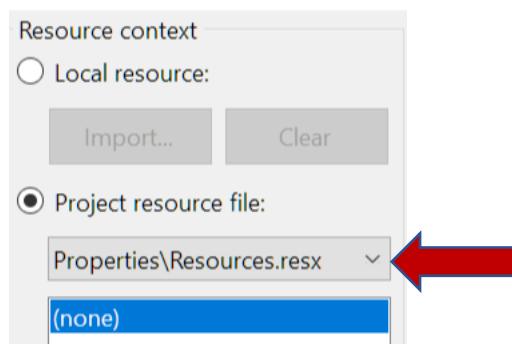


To add it to your Button, click on your **button** to bring up the **Properties** on the right.

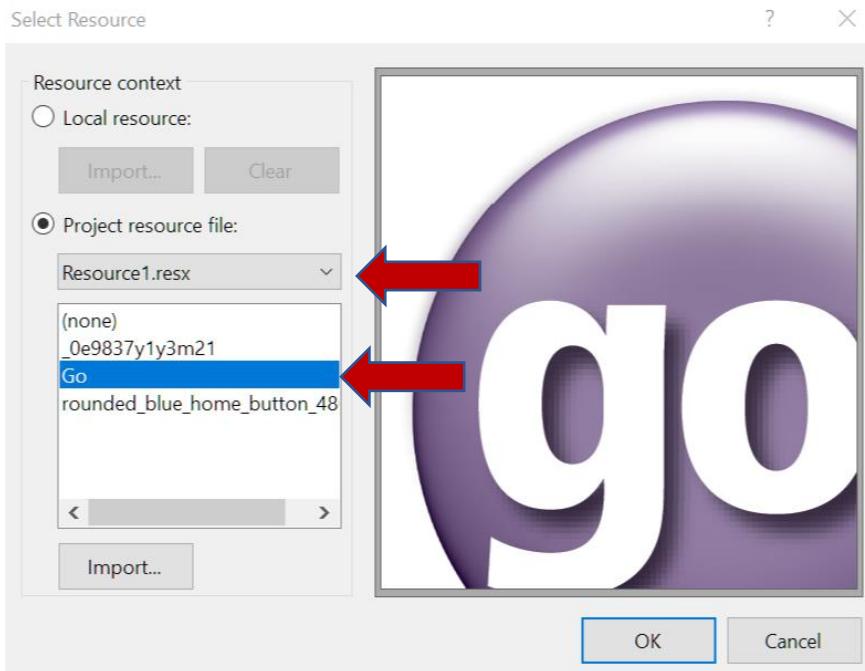


Click on Image to select a resource,

Choose resource1 (or whatever yours is called) from the pull-down menu.



The images will appear in the list below and you can choose whichever one you want.



Add the image to the Background Image and the layout to Stretch



10. Conditionals - Loops

Loops are code structures that allow parts of your program to repeat. There are a couple standard types of loops. One prominent type of looping structure repeats a code segment some specified number of times. This counting loop is called the For loop.

Loops are used for iteration purposes, i.e., performing a task multiple times (usually until a termination condition is met) Here is a good MS [overview](#)

There are two basic types of loops.

- A **pre-test** loop tests the condition before the body of the loop is executed. It is possible, using a pre-test loop, that the body of the loop might never be executed. **While**.
- A **post-test** loop tests the condition after the body of the loop is executed. **The number of iterations in a post-test loop is at least one. Do**

When writing a program, you might have to make a decision regarding which loop structure to use.

Watch the [Telerik Loops Video](#).

While Loop (Test at the beginning)

```
While (an expression that is either true or false (Boolean)
{
    Do something.
}
```

For loops are best for situations in **which you know how many times something will happen** or when you want to take advantage of the counting nature of the loop. For other kinds of looping situations, you can use a variation of the **While** loop.

In some of your programs you won't know how many times something is going to happen. In these instances, you need a while loop.

For example, you might need a program to continue asking a question until the user gets the correct answer, or an annoying kid in a car journey going “Are we there yet? Are we there yet? Are we there yet? etc”

In either case, you don't know beforehand how many times the loop will occur, so a while loop is the best solution for this type of situation.

To test at the end of the loop use a Do While loop.

The Do and Do While Loops

The **do...while** loop is written in the following form:

```
"do" body "while" "(" condition ")"
condition ::= boolean-expression
body ::= statement-or-statement-block
```

The **do...while** loop always runs its *body* once.

After its first run, it evaluates its *condition* to determine whether to run its *body* again.

If the *condition* is *true*, the *body* executes. If the *condition* evaluates to *true* again after the *body* has ran, the *body* executes again.

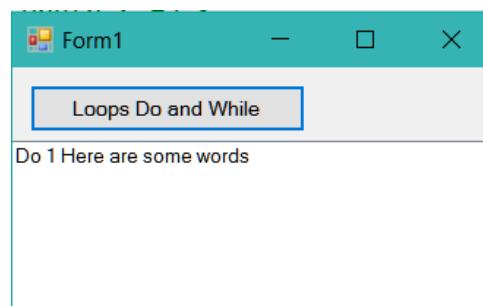
When the *condition* evaluates to *false*, the **do...while** loop ends.

[Clear out multiple spaces between words challenge](#)

Given a sentence with lots of spaces between the words can you cut them out and replace them all with one space?

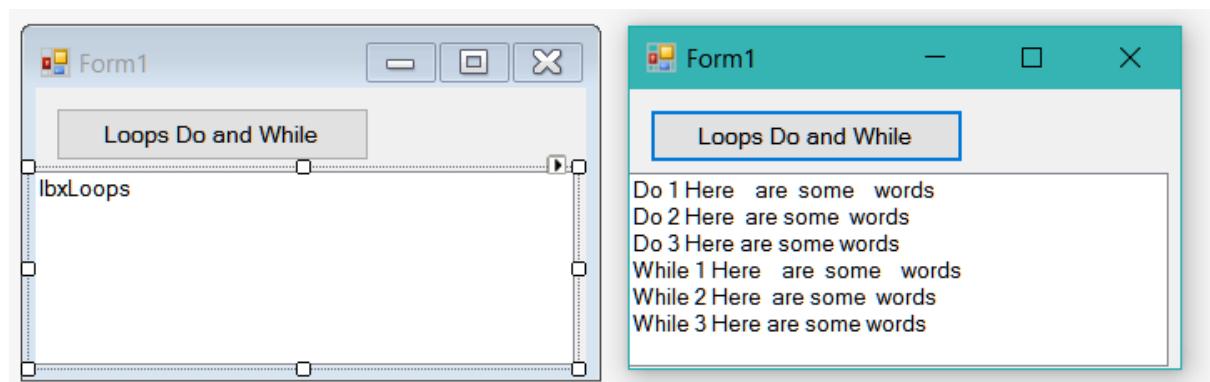
This might look tricky, but it's actually really simple with a Do While loop.

The Do loop runs once, even when there are no extra spaces!



Put some spaces between words

Lots of extra spaces.



Code over page...

```

private void btnLoops_Click(object sender, EventArgs e)
//This will always run at least once. Because the test is at the end.
    string OriginalSentence = "Here are some words";
    string Sentence = OriginalSentence;
    int counter = 0;
    do
    {
        counter++;
        Sentence = Sentence.Replace(" ", " ");
        lbxLoops.Items.Add("Do " + counter + " " + Sentence);
    } while (Sentence.Contains(" "));

//This will only run IF the condition is not met.
counter = 0;
Sentence = OriginalSentence;
while (Sentence.Contains(" "))
{
    counter++;
    Sentence = Sentence.Replace(" ", " ");
    lbxLoops.Items.Add("While " + counter + " " + Sentence);
}

```

Loops Exercises

These are the exercises from the Telerik video that the students have to do.

Write a program that prints all the numbers from 1 to `N`.

* Write a program that prints all the numbers from 1 to `N`, that are not divisible by 3 and 7 at the same time. FizzBuzz!

Write a program that reads a sequence of `N` integers and returns the minimal and maximal of them.

Write a program that prints all possible cards from a standard deck of 52 cards (without jokers). The cards should be printed with their English names. Use nested for loops and switch-case.

Write a program that reads a number `N` and calculates the sum of the first N members of the sequence of Fibonacci: **0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...** Each member of the Fibonacci sequence (except the first two) is a sum of the previous two members.

Write a program that calculates the **greatest common divisor** (GCD) of given two numbers. Use the Euclidean algorithm (find it on the Internet).

Do While – Fibonacci Sequence

Mathematician Leonardo Fibonacci posed the following problem in his treatise Liber Abaci:

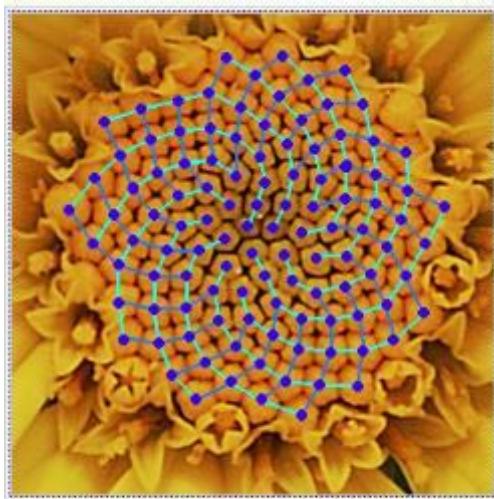
Assuming that a newly born pair of rabbits, one male, one female, are put in a field the rabbits are able to mate at the age of one month. So that at the end of its second month a female can produce another pair of rabbits.

If the rabbits never die and a mating pair always produces one new pair (one male, one female) every month from the second month on.

The puzzle that Fibonacci posed was: How many pairs will there be in one year?

- At the end of the first month, they mate, but there is still only 1 pair.
- At the end of the second month the female produces a new pair, so now there are 2 pairs of rabbits in the field.
- At the end of the third month, the original female produces a second pair, making 3 pairs in all in the field.
- At the end of the fourth month, the original female has produced yet another new pair, the female born two months ago produces her first pair also, making 5 pairs.

At the end of the n th month, the number of pairs of rabbits is equal to the number of new pairs (which is the number of pairs in month $n - 2$) plus the number of pairs alive last month ($n - 1$). This is the n th Fibonacci number



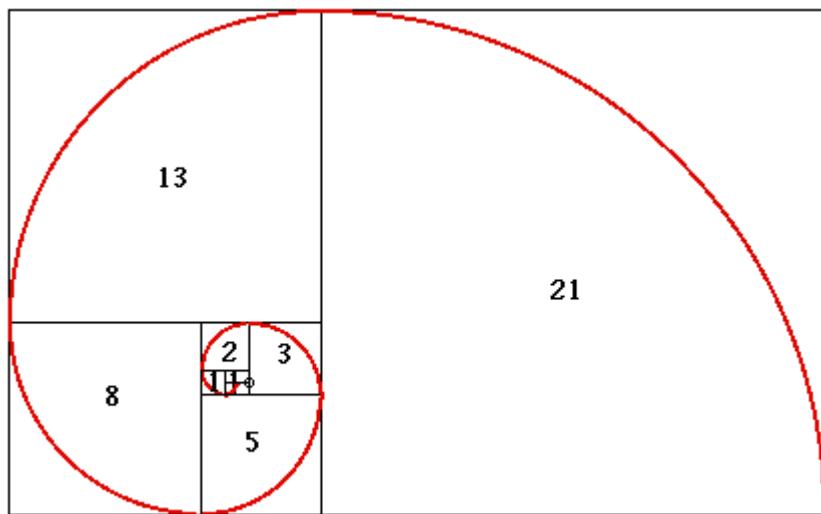
Yellow Chamomile head showing the arrangement in 21 (blue) and 13 (aqua) spirals. Such arrangements involving consecutive Fibonacci numbers appear in a wide variety of plants.



Fibonacci is found naturally in nature and also used by Stock traders <http://www.babypips.com/school/summary-of-fibonacci.html>.

All about Fibonacci http://en.wikipedia.org/wiki/Fibonacci_number

This is how the number sequence increases



<http://www.wikihow.com/Calculate-the-Fibonacci-Sequence>

The first number of the Fibonacci sequence is 1. It always starts on 1. It can work starting on other numbers, but if it doesn't start on 1, it is not the Fibonacci sequence.

The next number in the Fibonacci sequence is always the sum of the two previous.

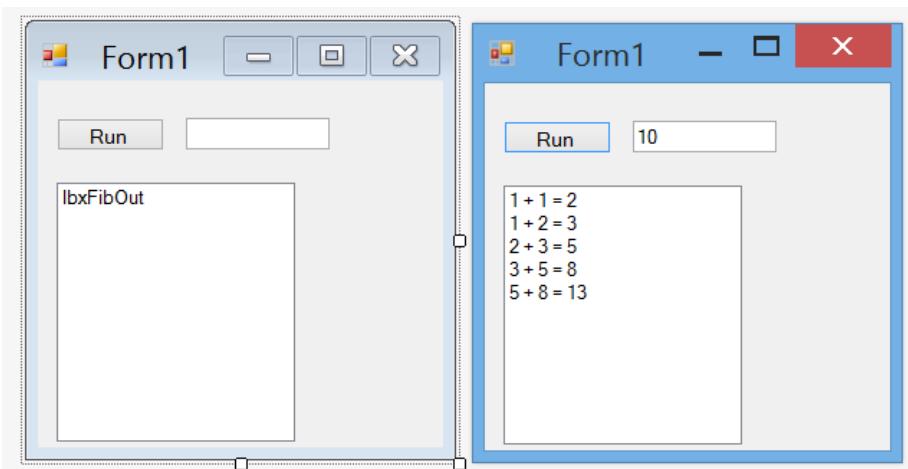
The two numbers before the second number are nothing and 1. So, the second number is 1.

The third number will be the sum of the previous two numbers as well, which now are 1 and 1. And since $1 + 1 = 2$, the third number is 2.

Then $1 + 2 = 3$

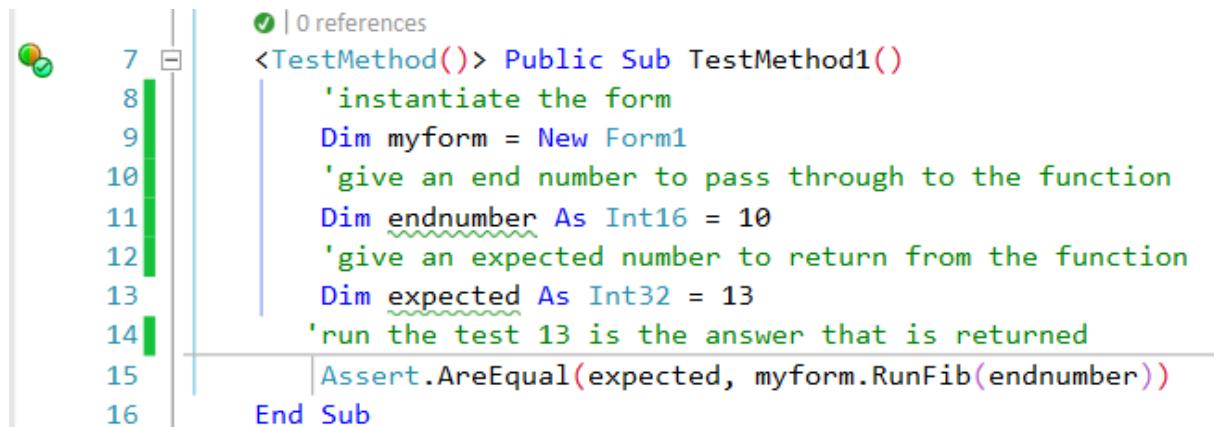
$$2 + 3 = 5$$

$$3 + 5 = 8$$



How will you make it? – You can use the net if you want. Hints over the page, **don't forget your unit test**

Here is my Unit test from the Function

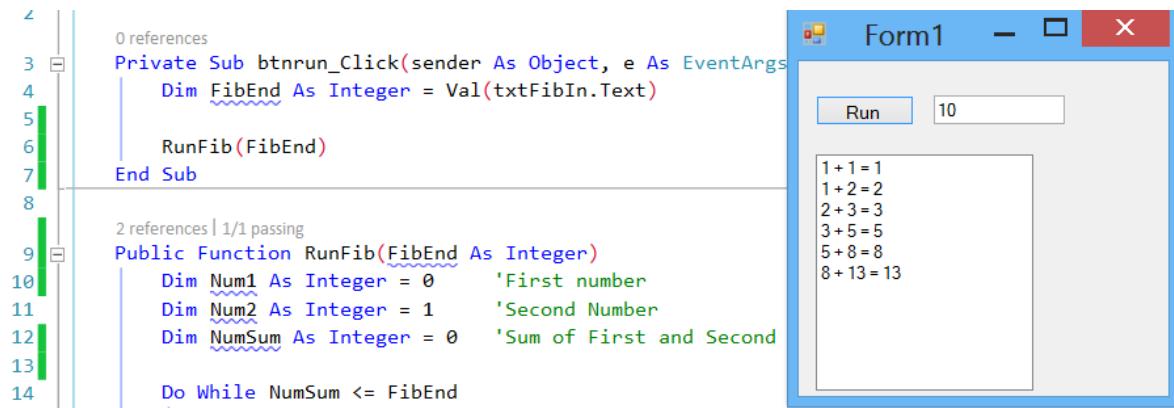


```

7 | 0 references
8 <TestMethod()> Public Sub TestMethod1()
9     'instantiate the form
10    Dim myform = New Form1
11    'give an end number to pass through to the function
12    Dim endnumber As Int16 = 10
13    'give an expected number to return from the function
14    Dim expected As Int32 = 13
15    'run the test 13 is the answer that is returned
16    Assert.AreEqual(expected, myform.RunFib(endnumber))
End Sub

```

Here is the beginning of my function



```

3 | 0 references
4 Private Sub btnRun_Click(sender As Object, e As EventArgs)
5     Dim FibEnd As Integer = Val(txtFibIn.Text)
6
7     RunFib(FibEnd)
8 End Sub
9
10 2 references | 1/1 passing
11  Public Function RunFib(FibEnd As Integer)
12      Dim Num1 As Integer = 0      'First number
13      Dim Num2 As Integer = 1      'Second Number
14      Dim NumSum As Integer = 0   'Sum of First and Second

```

This code doesn't show the first two given numbers 0 and 1. Fix it.

```

Do While NumSum <= FibEnd
    'add the first two numbers together (0 and 1)
    NumSum = Num1 + Num2
    lbfibOut.Items.Add(NumSum.ToString)
    'pass Num2 to Num1 to make a new Num1
    Num1 = Num2
    'Pass the total (NumSum) to Num2 to make a new Num2
    Num2 = NumSum
Loop

```

Many algorithms have this simple number swapping substitution system happening, once you get the hang of it it's easy to use elsewhere.

Find the even Fibonacci numbers only

Each new term in the Fibonacci sequence is generated by adding the previous two terms.
By starting with 1 and 2, the first 10 terms will be:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.

From: <http://projecteuler.net/problem=2>

What is the first Fibonacci number to contain 1000 digits?

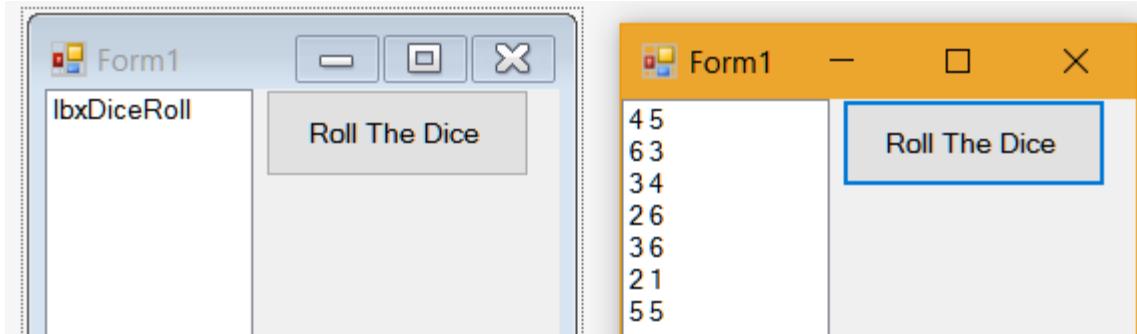
What is the first term in the Fibonacci sequence to contain 1000 digits?

<http://projecteuler.net/problem=25>

Dice rolls using Do While loop

A while loop checks a condition before the loop runs, but what if we want the loop to run and then check the condition. The Do While loop deals with this.

```
Do ( run the loop )
While { expression }
```



In this exercise, we make a program that rolls two dice and keeps on rolling until they reach the same number. You can see it happen on the image above.

All we need is a Button and a ListView on the form.

```
private void btnRoll_Click(object sender, EventArgs e)
{
    //Define a random generator that uses milliseconds as the seed
    Random myrandom = new Random(DateTime.Now.Millisecond);

    //Create and set the values of the dice to 0
    int Dice1, Dice2 = 0;

    //clear the listbox ready for a new roll run
    lbxDiceRoll.Items.Clear();
    do //roll the dice while ....
    {
        //pass the random number to two variables
        Dice1 = myrandom.Next(1, 7);
        Dice2 = myrandom.Next(1, 7);

        //Add them to the listview
        lbxDiceRoll.Items.Add(Dice1 + " " + Dice2);

        //while Dice 1 doesn't equal Dice 2
    } while (Dice1 != Dice2);
}
```

Here is
the loop
working
while
both dice
are not !=
equal

The following exercises are deliberately vague so that you have to get the code yourself. With significant hints from me.

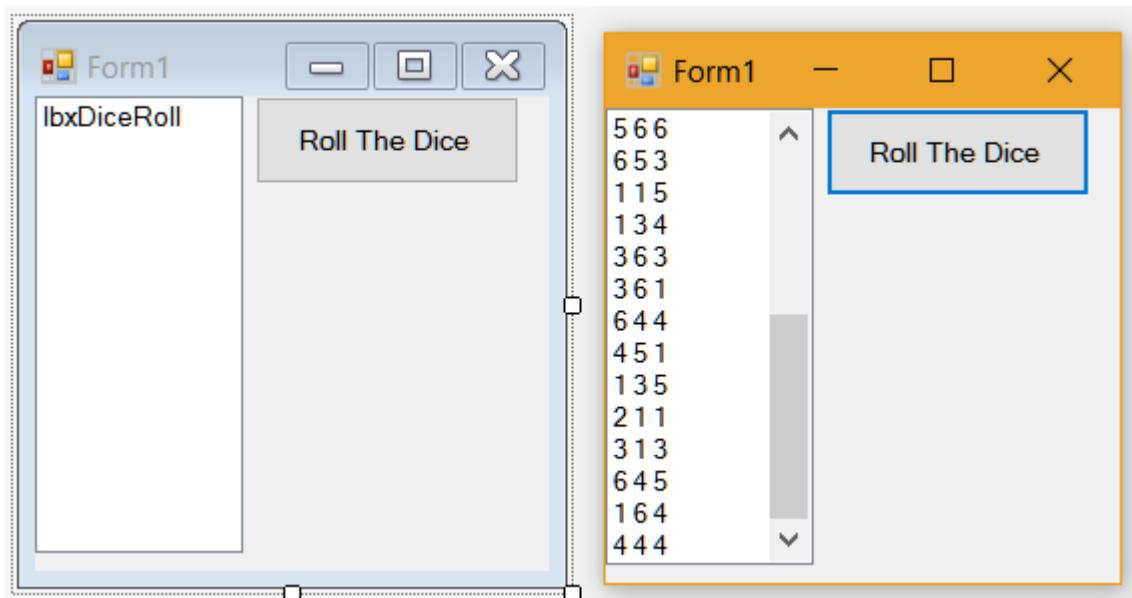
Exercise 1: Can you make this with 3 dice?

You will need to use `&&` as a concatenator between the while tests. However even this will show a logic error. What is it?

To solve this solution, you need to create an if statement to check if value1 equals value 2 and value 2 equals value 3.

You also need a Boolean to check when they all are equal. It is set to true and then becomes false when the values are equal.

```
//We need a boolean,(True, False) to tell us when the values match
bool RollDiceAgain = true;
```

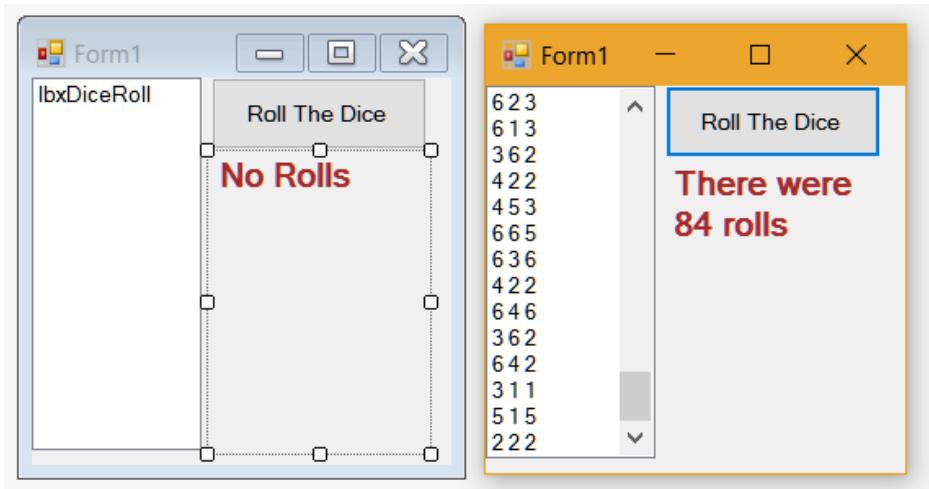


```
do //roll the dice while ....
{
    Dice1 = myrandom.Next(1, 7);
    Dice2 = myrandom.Next(1, 7);
    Dice3 = myrandom.Next(1, 7);
    lbdiceRoll.Items.Add(Dice1 + " " + Dice2 + " " + Dice3);

    if ((Dice1 == Dice2) && (Dice2 == Dice3))
    { RollDiceAgain = false; }
} while (RollDiceAgain == true);
```

Exercise 2 How many times does the program run before it gets a triple.

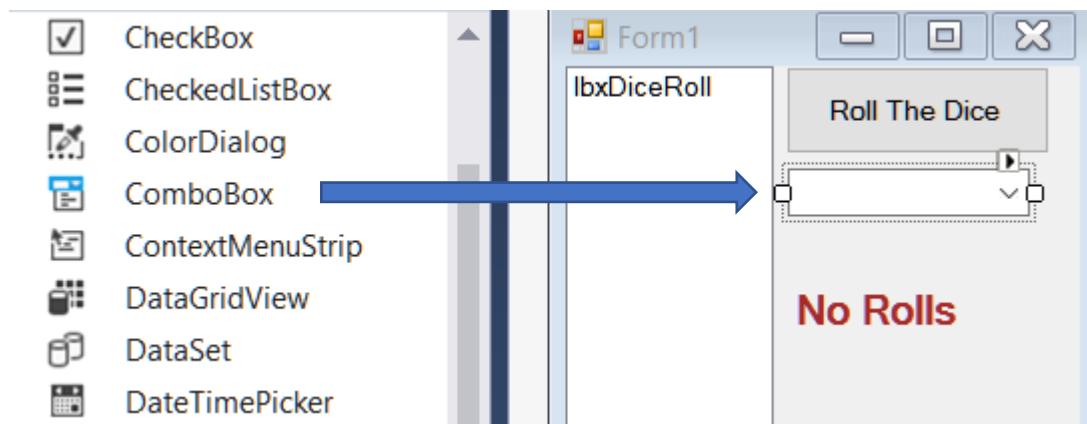
You need a counter that increases by 1 each time the loop rolls, and then outputs to a **label** when it finishes.



Exercise 3 Tell the program to run until it gets a triple number to stop on.

Instead of making the dice roll stop when each dice equals the other, what if we make a program that stops the dice roll when they all roll a 6? Or 4? Or any other number.

We want a ComboBox called **cbxStopNumber**



We then need to fill it with the numbers from 1 to 6 when the program loads.

In the Form1() constructor there is `InitializeComponent()`; this line loads the designer with all the controls. After that we can start our code as the ComboBox will have loaded.

Type in **for** and then press **Tab** and **Tab**

`InitializeComponent();`

That gives you a standard for loop to use.

`for`

```
for (int i = 0; i < UPPER; i++)
{
}
```

We need to set the start number to **1** and the upper limit to be **less than 7**, then **i** will increase from 1 to 6 and get added to the ComboBox.

```
for (int i = 1; i < 7; i++)
{
    cbxStopNumber.Items.Add(i);
}
```

All code

```
public Form1()
{
    InitializeComponent();
    for (int i = 1; i < 7; i++)
    {
        cbxStopNumber.Items.Add(i);
    }
    cbxStopNumber.SelectedIndex = 1;
}
```

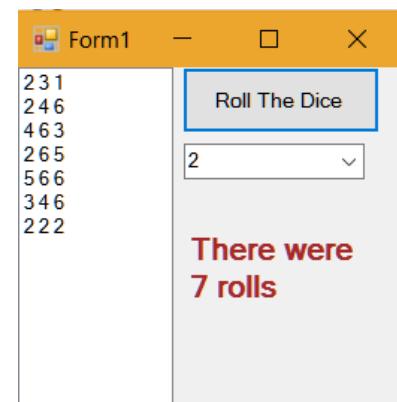
The `cbxStopNumber.SelectedIndex = 1;` part gives a start number of **2** that shows up in the combo box as a visual guide that its working.

Now We need to tell the program what the number is that we are stopping on. We can get that from `cbxStopNumber.Text` this will return the number that we select, in this case its 2.

The problem with this is that it comes back as a **string**, not and **int**. So, you will have to **convert it to an int** and then tell the program to stop when the dice all equal that number.

So, pass the ComboBox number to an Int variable called StopNumber, and add it to the code.

```
if ((value1 == value2) && (value2 == value3) && (value1 == StopNumber))
```



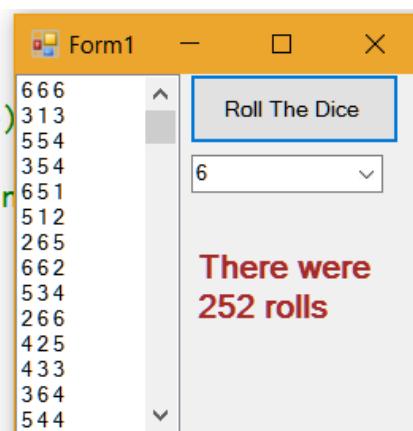
Add items to the top of the Combo Box Use Insert.

```
lbdxDiceRoll.Items.Insert(0, Dice1 + " " + Dice2 + " " + Dice3);
```

The **0** is the place in the ComboBox where you want to insert data. So, we are always inserting data to the 0 place, **at the very top**.

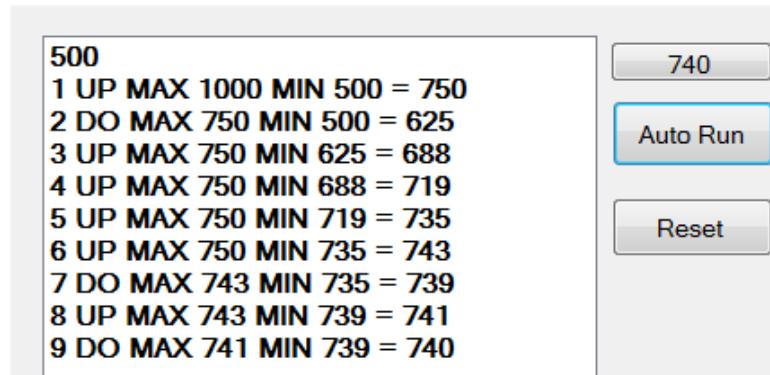
```
int Dice1, Dice2, Dice3 = 0;

//We need a boolean,(True, False)
bool RollDiceAgain = true;
//clear the listbox ready for a new roll
lbdxDiceRoll.Items.Clear();
do //roll the dice while ...
{
    Dice1 = myrandom.Next(1, 7);
    Dice2 = myrandom.Next(1, 7);
    Dice3 = myrandom.Next(1, 7);
    lbdxDiceRoll.Items.Insert(0, Dice1 + " " + Dice2 + " " + Dice3);
}
```



Guess a number program

The computer generates a number between -1 and 1000 and then has to guess what it is. All it knows is that the guess it makes is bigger or smaller than the target answer.



How does it do this? **By a process of halving its guesses.**

The best bet in this is to start the Guess at **500**.

If the Answer is **740** it is told that it's too low.

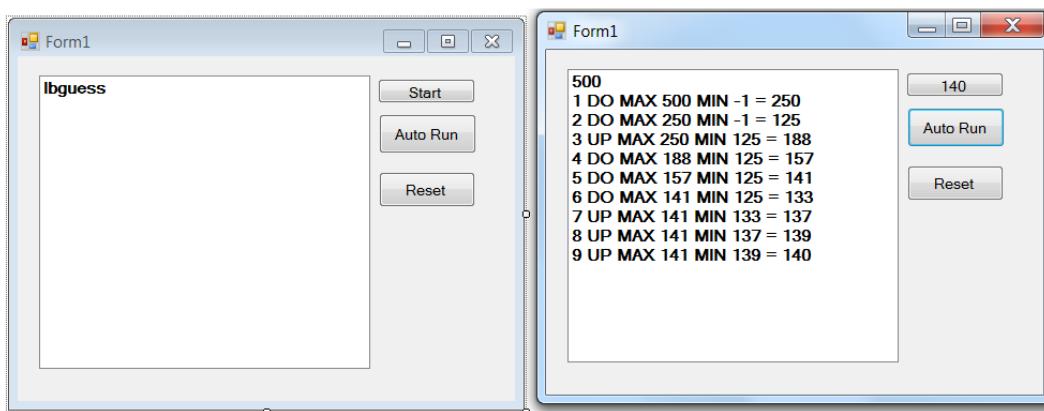
So, it knows its somewhere between 500 and 1000.

So, it finds the $\frac{1}{2}$ way point between 500 and 1000 and makes another guess (750).

Its told that 750 is too high. So, gets the $\frac{1}{2}$ way point between 750 and 500. (625)

That's too low. So, look at $\frac{1}{2}$ way between 750 and 625 (688)

Etc etc.



Work out your own code it's a good exercise.

We will do this in class (remind me)

```

if (answer > guess)
{
    min = guess;
    guess = max - ((max - min) / 2);
    counter++;
lbguess.Items.Add(counter + " UP MAX " + max + " MIN " + min + " = " + guess);

}

if (answer < guess)
{
    max = guess;
    guess = max - ((max - min) / 2);
    counter++;
lbguess.Items.Add(counter + " DO MAX " + max + " MIN " + min + " = " + guess);

```

Calculate Pi

From [here](#)

```

1  void Main()
2  {
3      var random = new SecureRandom();
4      var total = 100000000;
5      int count = 0;
6      for (int i = 0; i < total; i++)
7      {
8          var a = random.Next() & 0x7FFFFFFF;
9          var b = random.Next() & 0x7FFFFFFF;
10         if (GCD(a,b) == 1) count++;
11     }
12     double proportion = (double)count/(double)total;
13     var piEstimate = Math.Sqrt(6/proportion);
14     piEstimate.Dump();
15 }
16

```

▼ Results λ SQL IL Tree

3.1415887826029

For Loop

Possibly the most used and useful of the loops, you will start to dream about this loop in your sleep.

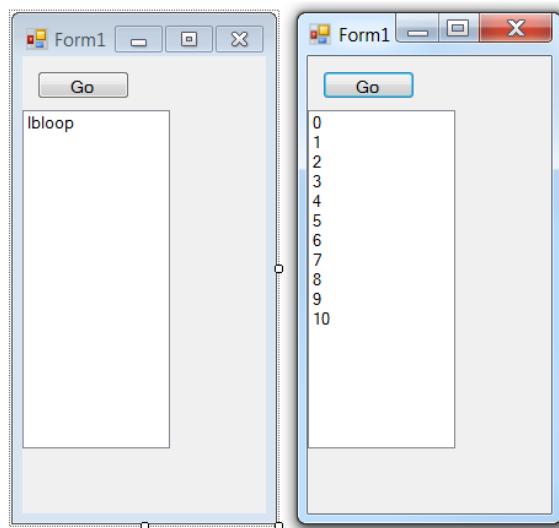
FOR(start number; to an end number, in steps of ..)

for (i = 0; i <= 10; i++)

Hint: make a FOR loop automatically by typing **for**, then pressing TAB twice.

The For statement executes the enclosed statement or statements repeatedly as follows:

- First, the initial value of the variable i is evaluated.
- Then, as long as the value of i is less than or equal to 10, the condition evaluates to true, the statement is executed and i is re-evaluated.
- When i is greater than 10, the condition becomes false and control is transferred outside the loop.



```
namespace C_sharp_for_loops
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnloop_Click(object sender, EventArgs e)
        {
            for (int i = 0; i <= 10; i++)
            {
                lbloop.Items.Add(i);
            }
        }
    }
}
```

Try these variations. Here is the For-Loop **Forward**

```
for (i = 0; i <= 10; i++)
```

Here it is **Backward** – counting down to 0 (note it's while $i > 0$)

```
for (i = 10; i > 0; i--)
```

Here it is in **steps of 2** going up

```
for (i = 0; i <= 10; i+=2)
```

Create a form with a button and a ListBox on it. Under the button click add the following For code that prints out numbers 1 to 100.

```
for(int i=0; i<100; i++)
{
    Lblist.items.add(i);
}
```

Cycling through colours

Add a Picturebox, and under the button click the name of the following method

```
private async void LoopColors()
{
    //instantiate a new color
    Color c = new Color();
    //loop through the entire color range
    for (int i = 0; i <= 255; i++)
    {
        c = Color.FromArgb(i, 255, i);
        //change the color to the new color
        pbxColorChange.BackColor = c;
        //make a 10 mill second pause
        await Task.Delay(10);
    }
}
```

From [Here](#).

Simple Array

```
//array          0           1           2           3           4           5
string[] names = { "Steve", "Thomas", "Isaac", "Sam", "Darren", "Gary" };

for (int i = 0; i < names.Length; i++)
{
    lbxLoop.Items.Add(i + " " + names[i]);
}
```

For Each loop

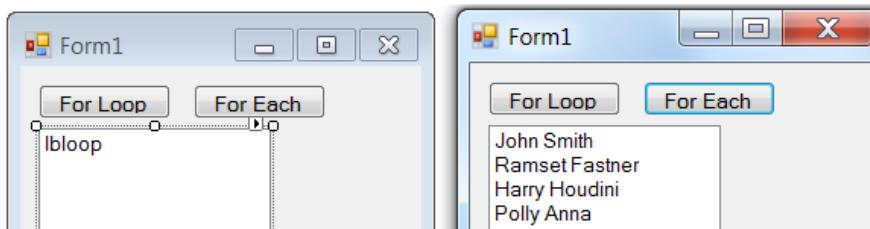
The For Each loop works mostly with arrays which we will cover later. The loop takes an item in an array and splits the array by that item.

ForEach is designed to iterate through a collection of items, setting a variable to represent each item in turn. During the loop, operations may be performed on the item.

One feature of the ForEach loop is that it is not possible to accidentally miscount and iterate over the end of the collection.

The general form of the ForEach statement is as follows:

```
foreach(type variable in collection)
statement;
```



For example,

you can

find the spaces in a sentence and split the sentence into words based on the spaces in between them. In this case, we are splitting the names by the entries in the array

Here is the array (we will cover this more later)

```
string[] people = {"John Smith", "Ramset Fastner", "Harry Houdini", "Polly Anna"};
```

Here it is being split into separate strings named **person**, and added to the ListBox.

```
foreach (string person in people)
lbloop.Items.Add(person);
```

```
private void btnforeach_Click(object sender, EventArgs e)
{
    // Use a string array to loop over names.
string[] people = {"John Smith", "Ramset Fastner", "Harry Houdini", "Polly Anna"};
    // Loop with the foreach keyword.
    foreach (string person in people)
    {
        lbloop.Items.Add(person);
    }
}
```

List exercise with people

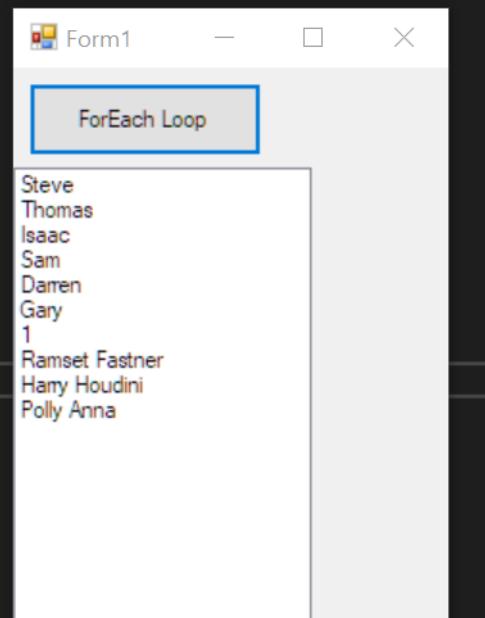
```
string[] people = { "1", "Ramset Fastner", "Harry Houdini", "Polly Anna" };

List<string> names = new List<string>();
names.Add(item: "Steve");
names.Add(item: "Thomas");
names.Add(item: "Isaac");
names.Add(item: "Sam");
names.Add(item: "Darren");
names.Add(item: "Gary");
names.Add(item: "Harald");

names.Remove(item: "Harald");

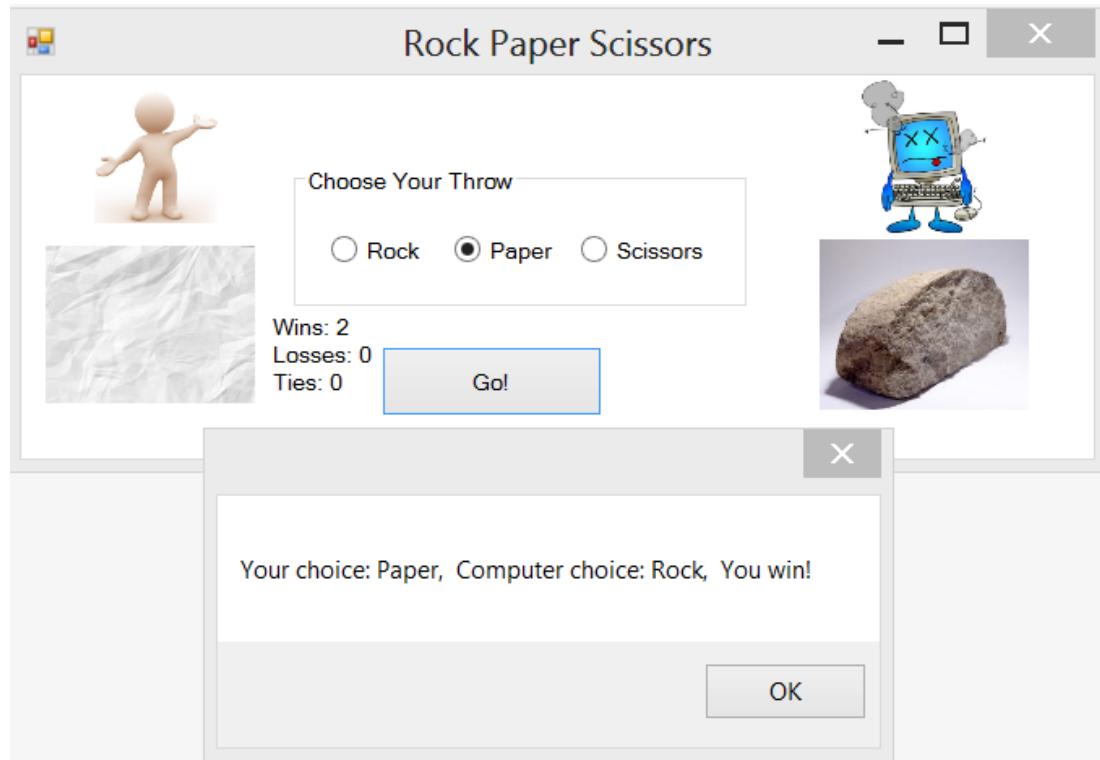
names.AddRange(people);

foreach (var person in names)
{
    lbxLoop.Items.Add(person);
}
```



Rock Paper Scissors

Create a program in which you compete against the computer in a game of Rock Scissors paper.



There are **three** parts to the program.

First is your choice, using the radio buttons you have to choose between Rock, Paper, Scissors.

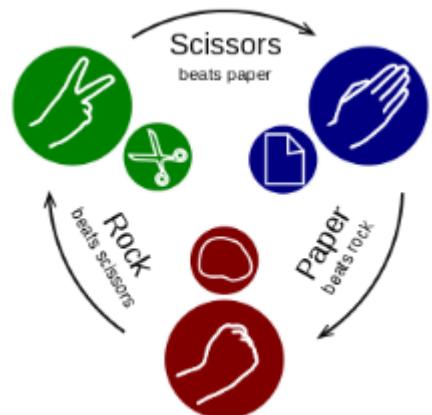
The second is the computer choice, where it chooses 1, 2 or 3, which become the words.

Finally the Third part is the comparison between the two,

How to win at [Rock Paper Scissors video](#)

Computer's side

This bit of code gives the computer side of the game. It generates 1, 2, 3. Using that we can turn it into the names.



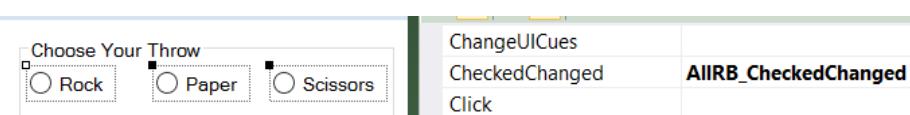
```
public int ComputerChoice() {
    //create a new instance of the Random Class
    Random CompGuess = new Random();
    //This code generates a random integer between 1 and 4, but 4 is not
    //inclusive, meaning the only possibilities are 1, 2 and 3
    //1 represents Rock, 2 represents Paper, 3 represents Scissors
    int RndComputerChoice = CompGuess.Next(1, 4);
    //Send the number back to the program
    return RndComputerChoice;
```

Here is my Unit Test for it.

```
public class UnitTest1 {
    [TestMethod]
    public void RandomNumberGeneratorTest() {
        //instantiate a new form and pass the form properties across
        Form1 f = new Form1();
        //run the method on the form and then pass the number to a variable for
        //testing
        int RndComputerChoice = f.ComputerChoice();
        //is the number greater than 0 and less than 4?
        Assert.IsTrue(RndComputerChoice > 0 && RndComputerChoice < 4);
    }
}
```

Human Side

Here is the human side. I have 3 radio buttons and **each** radio button is connected to the `AllRB_CheckedChanged` method.



Choose Your Throw
Rock Paper Scissors

Get

When any radio button is checked it runs the same method `AllRB_CheckedChanged` and sends in the button details using the sender to the fakeRB.

When you click on a radio button in a group there are two events running.

One is the check going **true**, black, `Rock` and the second is the other radio button is losing its black dot. It's becoming **false** `Paper`.

Both events happen and are sent through to the sender.

Therefore we only want to get the data from the checked radio button, so we use an If statement `if (fakeRB.Checked)` to isolate it. We could add **== True** to the condition but the default is True, so it's not needed.

We then run the `HumanChoice(fakeRB.Name)` method that takes in the name of the radio button eg: "rbnRock" and sends back the name of the choice, Rock, Paper etc. `return "Rock";`

The text is passed to a global variable (because I am trying to keep it simple) of `HumanChoiceText`.

```
private void AllRB_CheckedChanged(object sender, EventArgs e) //create a fake radio button
{
    RadioButton fakeRB = (RadioButton)sender;
    //get the radiobutton that is checked, as there are two events happening at once, one is becoming unchecked
    if (fakeRB.Checked) { //pass the name of the checked radiobutton to the method,
        HumanChoiceText = HumanChoice(fakeRB.Name);
    }
}

// RadioButton checkchanged displays the image when the button changes
private string HumanChoice(string name) {
    if (name == "rbnRock") {
        pbHumanPic.Image = Resource1.rock;
        return "Rock";
    } else if (name == "rbnPaper") {
        pbHumanPic.Image = Resource1.paper;
        return "Paper";
    } else if (name == "rbnScissors") {
        pbHumanPic.Image = Resource1.scissors;
    }
}
```

```
        return "Scissors";
    }
    return null;
}
```

Add images to your program to show the choices of the players.

Can you do this?

Programs like this are totally random, you can't predict which random number will occur. However if you use two people, then people are NOT random. If you can keep a record of what people have called in the past, you can work out their favoured calls.

What they found was that ...

"if a player wins over her opponent in one play, her probability of repeating the same action in the next play is considerably higher than her probabilities of shifting actions.

" If a player has lost two or more times, she is likely to shift her play, and more likely to shift to the play that will beat the one that has just beaten her than the same one her opponent just used to beat her.

For instance, if Megan loses by playing scissors to Casey's rock, Megan is most likely to switch to paper, which would beat Casey's rock. Per the research, this is a sound strategy, since Casey is likely to keep playing the hand that has been winning. The authors refer to this as the "win-stay, lose-shift" strategy.

Therefore, this is the best way to win at rock-paper-scissors: if you lose the first round, switch to the thing that beats the thing your opponent just played. If you win, don't keep playing the same thing, but instead switch to the thing that would beat the thing that you just played.

In other words, play the hand your losing opponent *just* played. To wit: you win a round with rock against someone else's scissors. They are about to switch to paper. You should switch to scissors. [How to win at RPS](#)

Can you sum the numbers from 1 to n?

This employer can't find people to do even this simple task. Can you? Read the article.

[Honest to god, is asking someone you're interviewing to sum the numbers from 1 to n too much to ask?](#)

It's kind of our question to weed out the people we shouldn't bother with. We ask them to write code in whatever language they want, even pseudo code to show us how to sum the numbers from 1 to n. That's it, we don't specify that they have to do it in any certain way and not even that it needs to be in the form of a function.

But almost no one we interview can do it! Many of them don't even grasp that they need a loop! I mean I knocked it out easy when I interviewed for my job here and I was astonished when I was told how many people can't manage that.

Is it too open-ended? Could we be somehow making them too nervous to figure it out? Is it that we're wasting our time picking up resumes from career fairs? Or is it just that everyone in DC lies through their teeth on their resumes?

We just had one guy who claimed 14 years of software engineering who couldn't manage this.

Find the multiples of 3 and 5

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below 1000.

<http://projecteuler.net/problem=1>

The answer is 233168. But how do you calculate it?

The FizzBuzz Project – Standard interview Question

A programming interviewer laments

After a fair bit of trial and error I've discovered that people who struggle to code don't just struggle on big problems, or even smallish problems (i.e. write a implementation of a linked list). They struggle with tiny problems.

So I set out to develop questions that can identify this kind of developer and came up with a class of questions I call "FizzBuzz Questions" named after a game children often play (or are made to play) in schools in the UK.

An example of a Fizz-Buzz question is the following:

Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

Most good programmers should be able to write out on paper a program which does this in under a couple of minutes. Want to know something scary? The majority of comp sci graduates can't. I've also seen self-proclaimed senior programmers take more than 10-15 minutes to write a solution.

FizzBuzz is a game that has gained in popularity as a programming assignment to weed out non-programmers during job interviews.

The object of the assignment is less about solving it correctly according to the below rules and more about showing the programmer understands basic, necessary tools such as if-else-statements and loops.

The rules of FizzBuzz are as follows:

For numbers 1 through 100,

- if the number is divisible by 3 print Fizz;
- if the number is divisible by 5 print Buzz;
- if the number is divisible by 3 and 5 (15) print FizzBuzz;
- else, print the number.

Try and make it without checking out these sites.

<http://c2.com/cgi/wiki?FizzBuzzTest>

<http://www.codinghorror.com/blog/2007/02/why-cant-programmers-program.html>

11. Source Control

New [Intro to Git](#)

Version control software provides many different features and solves several problems related to writing and maintaining software, especially when there is a team of developers involved rather than just one.

The repository can be thought of as storage. It's where you put your project. The first main feature that version control software provides is that it **allows you to easily share your project with other people**.

Give them a link to your repository and they can obtain a copy for themselves. This operation is usually called **cloning** the repository, or **checking it out**. This has obvious advantages for software that is worked on by several developers, especially when the developers are distributed geographically.

Now suppose you make some changes to your source code that you're happy with. You obviously want to save them, so you do that by **committing** your changes with a commit message that explains what the changes you made are for. Your changes are now saved in the repository and other people with access to your repository will be able to see them.

The interesting thing about version control software is that it **stores a history of every commit ever made**, so you can always go back to *any* previous version of a file, and you can also compare any two versions of a file.

For example, suppose you know that a bug was introduced after some point in time. You can go back to that point in time and compare the source then with the source as it is now, and you'll be able to see all of the changes that have been made since.

Furthermore, you can look back through each change individually and look at the commit messages to see what each change was for, who made the change, when it was made, what changed etc.

Commits are also **atomic**, which allows several developers to make changes to the same source code at the same time. **I can make changes to code at the same time as you, and we can each have our own copies saved to our own computers.**

Now suppose you finish editing your code and commit your changes. Later on, I finish my changes and also try to commit them. The version control software will tell me that the file I'm trying to commit has been changed, and will then give me the opportunity to **merge** my changes with the ones you've already committed.

Most version control systems have very clever tools which can automatically and intelligently merge changes together, or at least provide an easy way to manually merge changes.

Another useful feature provided by most version control systems is **branching**. A branch is a set of changes which are isolated from other branches containing other sets of changes.

This allows developers to independently work on the same source, even committing many changes, without interfering with each other. I can commit all of my changes to the code in my own branch while you commit all of your changes to code in *your* own branch, and there's no problem.

I won't see your changes and you won't see mine. Later, once we've both independently completed our features, we can **merge** the branches together. Once again, the merge tool will be able to intelligently merge our changes together.

[Git](#) is a commonly used version control system. In fact, it's a **distributed** version control system, which means that it **doesn't need any central location** on which to store the repository - every developer can have their own copy of the repository and they can **push** their changes or **pull** changes from each other (or a central location, if they really want).

Another popular one is [Mercurial](#). A non-distributed VCS that's still frequently used is [Subversion](#) (though distributed VCS is more common these days).

[Github](#) is a central location for you to host your git repositories, and provides other project management features. Another example is [Bitbucket](#).

Here is a simple example of how a development team might use branches to work on a project.

- When a developer begins work on a feature, he or she creates a new branch. The feature is developed on this branch until it's complete, after which it's tested.
- Once a feature has been tested, the branch that the feature is on can then be merged into an integration branch. It is then tested alongside other features that have also been developed and recently merged in with it. This ensures that different features work together correctly.
- Points on the integration branch which have been tested and found to be stable and working correctly are then merged into a stable branch. This is a branch which contains only complete, stable, fully tested and integrated features. This is the branch from which official releases can be made. Different points on the branch can be tagged with version numbers or release names.

That's an example of a simple workflow, but there can be more complex workflows involving more branches and merges as you add bug fixes, hot fixes, patches, refactoring, code reviews etc. into the mix.

Watch the Working with Visual Studio section to see how to operate it in VS.

<http://pluralsight.com/training/Courses/TableOfContents/intro-to-svn>

Someone's plea for help understand Git

[Git github repositories etc can someone el15/](https://www.reddit.com/r/explainlikeimfive/comments/2qjwvz/git_github_repositories_etc_can_someone_el15/)

Git: It's a program that will remember how your work looks and used to look for you. It can also compare it to other people's work if you're more people doing the same work.

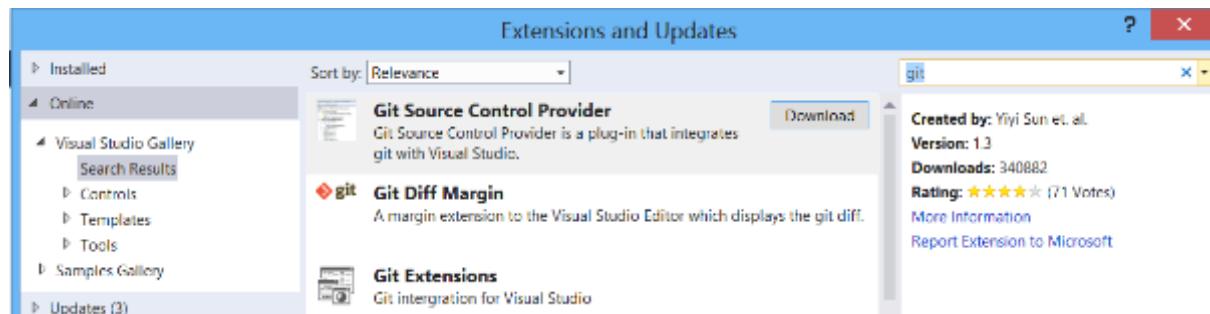
Github: A website that uses Git to let you share your work and find people to work with and stuff to work on.

Repository: A folder where you have work that Git is remembering for you.

Installing Git

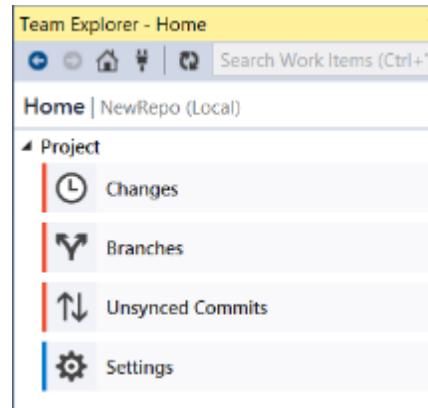
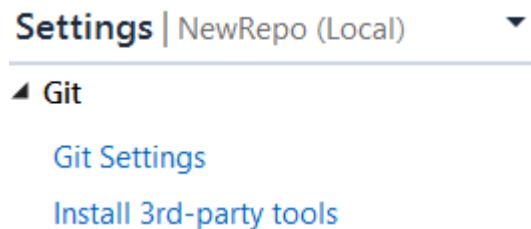
Got Tools/Extensions and Updates and do a search for Git Online.

Download the **Git Source Control Provider** and later the **Git Extensions**. Then Restart the program.



This adds the Following under your Team Explorer Tab

Click on **Home**, and go to **Settings**

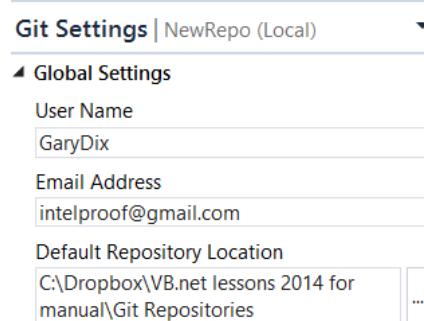


Click on Git Settings and Change the path of your saved files to where ever you normally save. I made a new folder and named it **Git Repositories**.

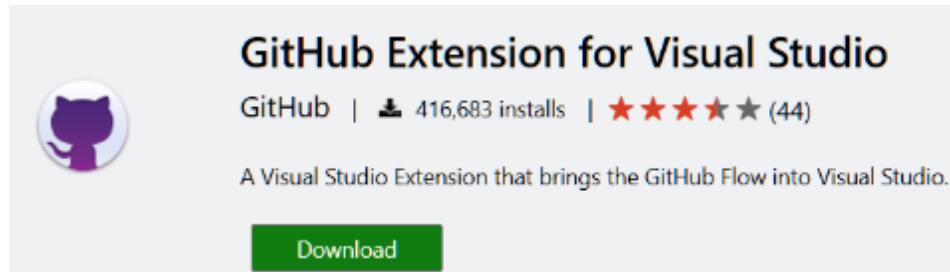
[Git on your Visual Studio – Review History](#)
[Get Started with Git and Team Services](#)

[Review history](#) – awesome tool

[Git History \(git log\)](#) – looks good



GitHub Installation



Download from [here](#), or via your Extensions Menu Option

1. **Connect** - From the Team Explorer section, click the *Connect...* button in the GitHub invitation section to login to the extension. The extension supports two-factor authentication (2fa) with GitHub and stores credentials in the Windows Credential store so that Git Operations within Visual Studio work with your GitHub repositories. The extension also supports logging into a GitHub Enterprise instance.
2. **Clone** - Once connected, click on the *Clone* button to list all repositories that you have access to on GitHub.
3. **Create** - The create dialog lets you create a repository on GitHub.com and locally that are connected together.
4. **Publish** - For a local-only repository, click on the Sync navigation item to get the GitHub publish control. This make it quick to publish your local work up to GitHub.
5. **Open in Visual Studio** - once you log-in with the extension, GitHub.com will show a new button next to repositories labeled "Open in VisualStudio." Click on the button to clone the repository to Visual Studio.
6. **Create Gist** - Create gists by using the GitHub context menu when you right-click on selected text
7. **Open/Link to GitHub** - Easily open on GitHub or share a link to the code you're working on by using the GitHub context menu.
8. **Pull Requests** - View your repository's Pull Requests and create new ones from the *Pull Requests* button in the Team Explorer Home

12. Arrays

Arrays and Best Practices

Arrays allow you to store and use large amounts of items as a group. It's basically a list of objects that's numbered off starting at 0.

Each item is called an element. C# arrays **are zero indexed**; that is, the array indexes start at **zero** and counts up from there.

When declaring an array, the square brackets [] must come after the type, int[], not the identifier.

```
int[] numbers; // not int numbers[];
```

Arrays are a **fixed size**. They **cannot be changed in size**, so they are best used when you have a known number of items that are not going to change.

[5] tells the array that it will hold 5 elements in it starting from 0 to 4

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

You can define your array at the beginning and then specify the value later in the code. This could be handy if you don't know the size of the array at first and you need to count the items before building it.

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

Yeah, I code



If you declare the array at the beginning you can leave out the number as it counts them and assumes that's how many you want.

```
int[] myInts = new int[1];
myInts[1] = 5;
```

```
int[] numbers = new int[] {1, 2, 3, 4, 5};
string[] names = new string[] {"Matt", "Joanne", "Robert"};
```

You can shorten the code to the following.

```
int[] numbers = {1, 2, 3, 4, 5};
string[] names = {"Matt", "Joanne", "Robert"};
```

You can add data to your array by calling each individual element as well

```
string[] names = new string[3];
names[0] = "John";
names[1] = "Harry";
names[2] = "Susan";
```

The two types below are the same.

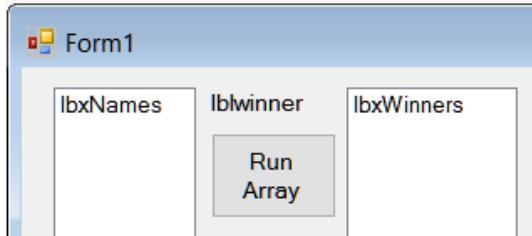
```
var colorOptions = new string[4];  
  
colorOptions[0] = "Red";  
colorOptions[1] = "Espresso";  
colorOptions[2] = "White";  
colorOptions[3] = "Navy";  
  
string[] colorOptions = new string[4] {"Red", "Espresso", "White", "Navy"};
```

Common Mistakes in Arrays

| Common Mistake | Error Description | Corrected Code |
|---|--|--|
| <code>int numbers[];</code> | The square braces for declaring an array appear after the data type, not after the variable identifier. | <code>int[] numbers;</code> |
| <code>int[] numbers; numbers = {42, 84, 168};</code> | When assigning an array after declaration, it is necessary to use the new keyword and then specify the data type. | <code>int[] numbers; numbers = new int[]{ 42, 84, 168 };</code> |
| <code>int[3] numbers = { 42, 84, 168 };</code> | It is not possible to specify the array size as part of the variable declaration. | <code>int[] numbers = { 42, 84, 168 };</code> |
| <code>int[] numbers = new int[];</code> | The array size is required at initialization time unless an array literal is provided. | <code>int[] numbers = new int[3];</code> |
| <code>int[] numbers = new int[3]{}</code> | The array size is specified as 3, but there are no elements in the array literal. The array size must match the number of elements in the array literal | <code>int[] numbers = new int[] { 42, 84, 168 };</code> |
| <code>int[] numbers = new int[3]; Console.WriteLine(numbers[3]);</code> | Array indexes start at zero. Therefore, the last item is one less than the array size. (Note that this is a runtime error, not a compile-time error.) | <code>int[] numbers = new int[3]; Console.WriteLine(numbers[2]);</code> |
| <code>int[] numbers = new int[3]; numbers[numbers.Length] = 42;</code> | 1 needs to be subtracted from the Length to access the last element. (Note that this is a runtime error, not a compile-time error.) | <code>int[] numbers = new int[3]; numbers[numbers.Length- 1] = 42;</code> |

Array with a For loop

Create a new project and add a ListBox **lbxNames** and a button **btnArray**



Firstly we create our array that holds a list of criminals names.

Note that the array elements are in red.

```
string[] names = {"Arthur", "Daniel", "Jane", "Darren", "Ethan", "Jeff", "Judah",
"Kyle", "Sonya", "Josh", "Keum"};
```

The array is called **names**. To get the array into the ListBox can be done in a number of ways, the most common is to either count them in using a **For Loop** or as each element with a **For Each Loop**.

```
for (int i = 0; i < names.Length; i++)
```

Arrays start at 0 so this gets the first element to the last element with **names.Length**. Note the command **<** less than. Because the array started at 0 it stops 1 **BEFORE** the number held in the length.

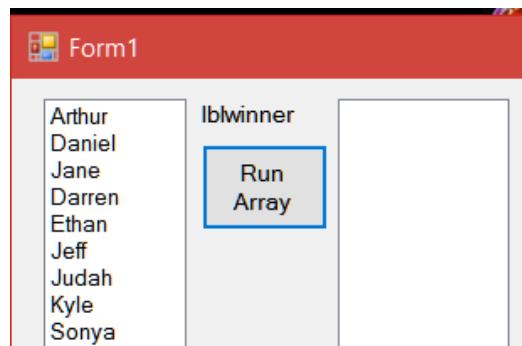
So this **For loop** adds in the names at their element storage point.

```
Names[0] = "Arthur"
Names[1] = "Daniel"
Names[2] = "Josh".
```

Here is the code.....

```
private void btnArray_Click(object sender, EventArgs e)
{
    //Clear the listbox
    lbxNames.Items.Clear();
    // Create an array of names
    string[] names = {"Arthur", "Daniel", "Jane", "Darren", "Ethan", "Jeff",
"Judah", "Kyle", "Sonya", "Josh", "Keum" };

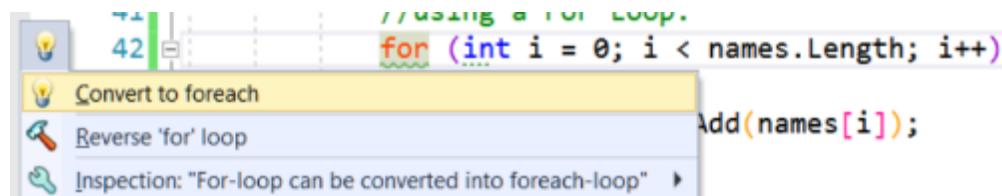
    //using a For Loop.
    for (int i = 0; i < names.Length; i++)
    {
        lbxNames.Items.Add(names[i]);
    }
}
```



Even though you can use a For loop, **even Visual Studio doesn't like you doing that!**

```
//using a For Loop.  
for (int i = 0; i < names.Length; i++)  
    Add(names[i]);  
}
```

ForEach is made to deal with collections.



Array with For Each Loop

We can use a **For Each** loop to extract the elements from the array as well.

Note that this creates a new variable called `name`, that is a **string**. Each element is passed to the `name` variable and that is passed on to the `ListBox`.

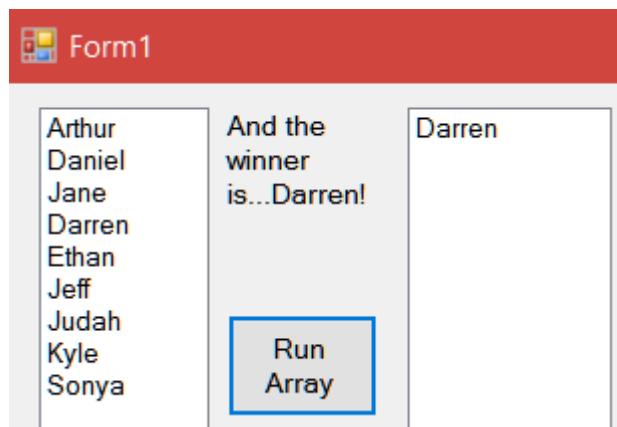
```
//load them into the listbox - I love ForEach
foreach (string name in names)
{
    lbxNames.Items.Add(name);
}
```

The ForEach actually is converted into a For loop at runtime, we just don't see it.

Unlike a For loop where you can change the elements such as `names[0] = "Josh"`,
ForEach's are read-only, you can't change `person = "Josh"` in the loop".

Create a random number generator that holds the number of the people in the array.

Because we clear out the `lbxNames` each time you click the button you can just add it to the button click.



```
//create a random number generator
Random myrandom = new Random();
//run a random number and pass it to winner note Names.Length has -1 on it.
int winner = myrandom.Next(0, names.Length - 1);
//put the winner number into the array to get back the name.
lblWinner.Text = "And the winner is..." + names[winner] + "!";
//Add to listbox
lbxWinners.Items.Add(names[winner]);
```

Create a `lblWinner` and when you click on the button a random winner is selected from the array and shown on the label with the extra text around it.

Arrays can also be searched for elements with **.Contains..**

```
if (names.Contains("Daniel"))
{
    this.Text = "Daniel is in the list";
}
```

There is an **Array class** that can be used as well with a range of useful methods. Add this before your For Each statement.

```
Array.Sort(names);
Array.Reverse(names);
```

You can also set the place of the elements using the **SetValue** method.

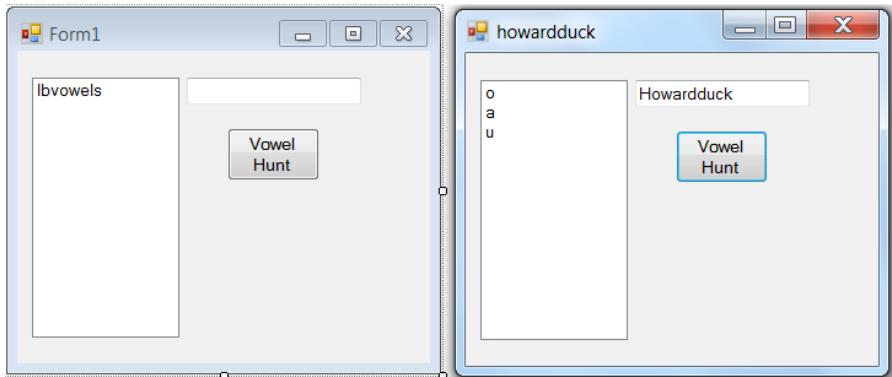
```
names.SetValue("Keum", 0);
names.SetValue("Steve", 1);
```

Put these before your **For** statement.

Char Array (Character Array) Find Vowels

A **char array** stores string data. Not only can string characters be accessed individually using the array, but it is also possible to retrieve the entire string as an **array of characters** using the string's **ToCharArray()** method.

This little exercise takes a word, chops it up into each letter and passes it to a Char array. Then it compares each letter in the word with each letter in the Vowels array when there is a match it prints them out to the list box.



```

private void btncheck_Click(object sender, EventArgs e)
{
    //An array of vowels
    char[] vowels = { 'a', 'e', 'i', 'o', 'u' };

    //input the text from the text box and make all the letters lower case
    string word = txtword.Text.ToLower();

    //just a check to see its working
    this.Text = word;

    //pass the word to a char array called letters
    //Note that the length of this array is the length of the word being passed in
    // - we don't have to specify it.
    char[] letters = word.ToCharArray();

    //run 2 loops comparing each letter with each other letter in the arrays.
    foreach (char letter in letters)
    {
        foreach (char vowel in vowels)
        {
            if (vowel == letter)
            {
                lbvowels.Items.Add(vowel);
            }
        }
    }
}

```

Ok this is easy but how will you be able to count how many vowels are in a word. ie: There are 2 i's, 3 e's etc.

You will need to use the index, as you need to get each element out at a place.

Hint, use a **For loop** and another array **vowelcount** to hold how many times it occurs.

```
var vowelcount = new int[5];
```

In your IF statement have it add 1 to the array at the place where the letter is

```
if (letters[i] == vowels[j])
{
    //add 1 each time to the vowelcount array
    vowelcount[j]++;
}
```

See how each letter in the char array is converted into a number, and that number is what is compared between chars, not the letter itself. →

When you get that working, count the total number of vowels and the total number of consonants.

Now using the following code snippet you can change the vowels in the word, as it checks each letter in the char array to uppercase

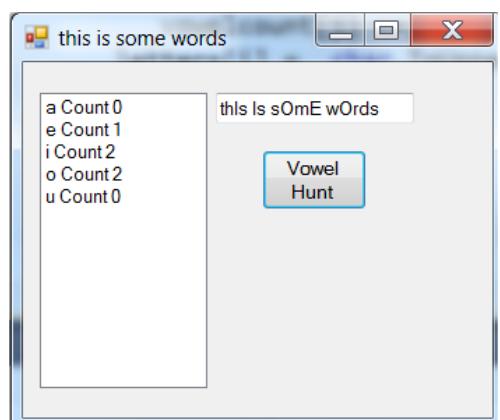
```
letters[i] = char.ToUpper(letters[i]);
```

For this to work we are going to have to rebuild the char array back into a word and redisplay it in the text box.

```
for (int i = 0; i < letters.Length; i++)
{
    for (int j = 0; j < vowels.Length; j++)
    {
        if (letters[i] == vowels[j])
        {
            //add 1 each time to the vowelcount array
            vowelcount[j]++;
        }
        //set the letter to uppercase
        letters[i] = char.ToUpper(letters[i]);
    }
}
//show the output in the listbox
for (int i = 0; i < vowels.Length; i++)
{
```

| Locals | | |
|--------------|----------------------|--------|
| Name | Value | Type |
| i | 0 | int |
| vowels | {char[5]} | char[] |
| [0] | 97 'a' | char |
| [1] | 101 'e' | char |
| [2] | 105 'i' | char |
| [3] | 111 'o' | char |
| [4] | 117 'u' | char |
| + vowelcount | {int[5]} | int[] |
| word | "this is some words" | string |
| letters | {char[18]} | char[] |
| [0] | 116 't' | char |
| [1] | 104 'h' | char |
| [2] | 105 'i' | char |
| [3] | 115 's' | char |
| [4] | 32 '' | char |
| [5] | 105 'i' | char |
| [6] | 115 's' | char |
| [7] | 32 '' | char |
| [8] | 115 's' | char |
| [9] | 111 'o' | char |
| [10] | 109 'm' | char |
| [11] | 101 'e' | char |
| [12] | 32 '' | char |
| [13] | 119 'w' | char |
| [14] | 111 'o' | char |

Autos Locals Watch 1



```
        lbvowels.Items.Add((vowels[i] + " Count " + vowelcount[i]));
    }
    string newword = null;
//rebuild the letters array back into a word and display it in the box
    foreach (var letter in letters)
    {
        newword += letter;
    }
    txtword.Text = newword;
}
```

Char Array Palindrome

A **char array** stores string data. Not only can string characters be accessed individually using the array, but it is also possible to retrieve the entire string as an **array of characters** using the string's **ToCharArray()** method.

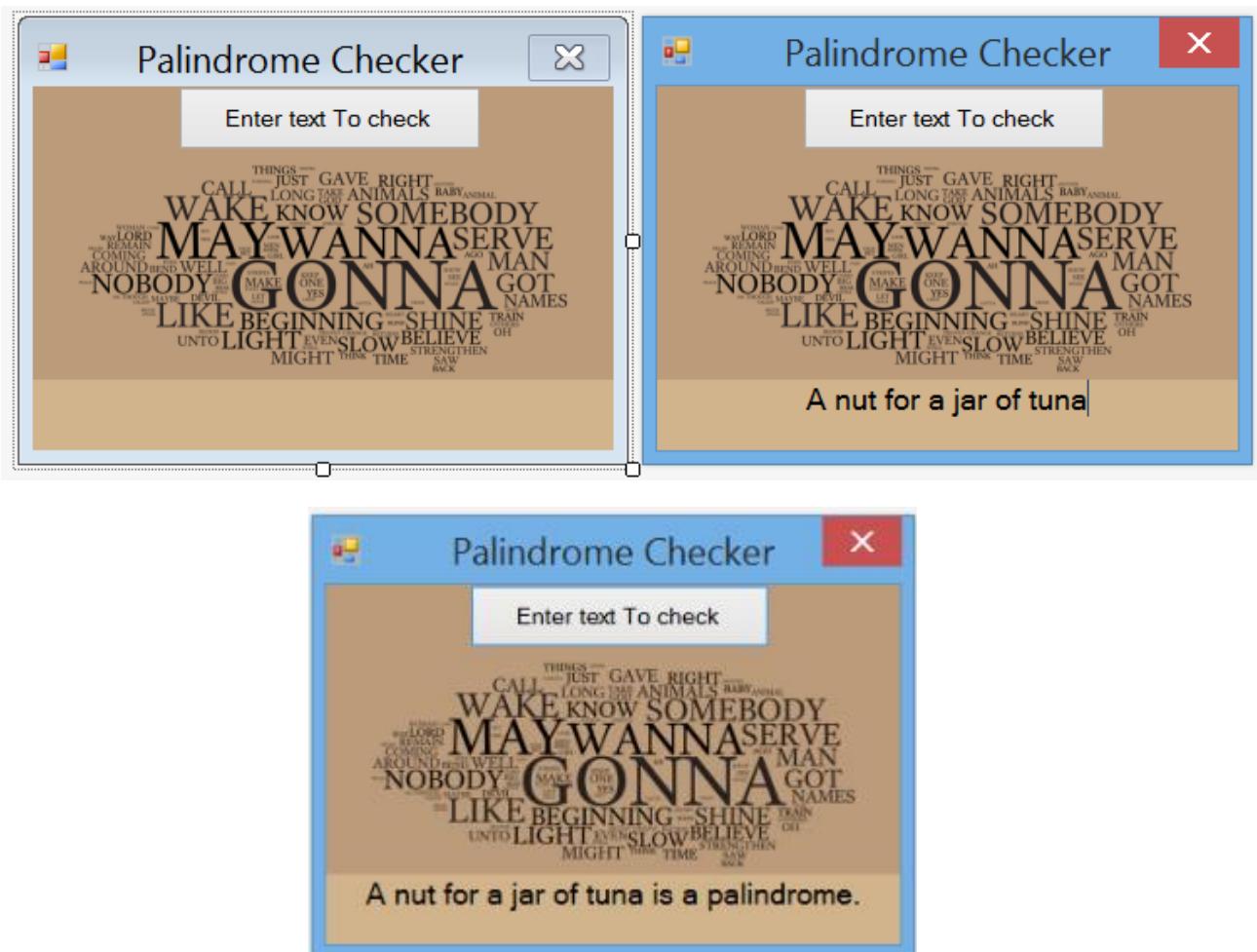
Using this method, you could reverse the string using the **System.Array.Reverse()** method which Reverses the sequence of the elements in the entire one-dimensional Array..

This example uses the **new** keyword **new string(text)**) which creates a new string from the reversed array of characters.

A **palindrome** is a word, phrase, number, or other sequence of symbols or elements that reads the same forward or reversed, with general allowances for adjustments to punctuation and word dividers.

Famous examples include "Amore, Roma", "A man, a plan, a canal: Panama" and "No 'x' in 'Nixon'" . <http://en.wikipedia.org/wiki/Palindrome>

Some examples <http://www.palindromelist.net/>



Note that I have used `/// <summary>` at the beginning of the **CheckPalindrome** method. (just type `///` and it generates).

Also I pass data in and out of the method, and only have calls to the objects under the button code.

The method takes in a string and returns another string

```
textToCheck.Text = CheckPalindrome(textToCheck.Text);
```

```
private string CheckPalindrome(string textIn)
```

```
private void btnarray_Click(object sender, EventArgs e) {
    textToCheck.Text = CheckPalindrome(textToCheck.Text);
}

/// <summary>
/// Checks if the text entered is a palindrome returns a true false string
/// </summary>

private string CheckPalindrome(string textIn) {
    //keep a copy of the text that is entered to compare
    string oldtext = textIn;

    // Remove spaces and convert to lowercase
    textIn = textIn.Replace(" ", "");
    textIn = textIn.ToLower();

    // Convert to an array
    char[] text = textIn.ToCharArray();

    // Reverse the text array
    Array.Reverse(text);

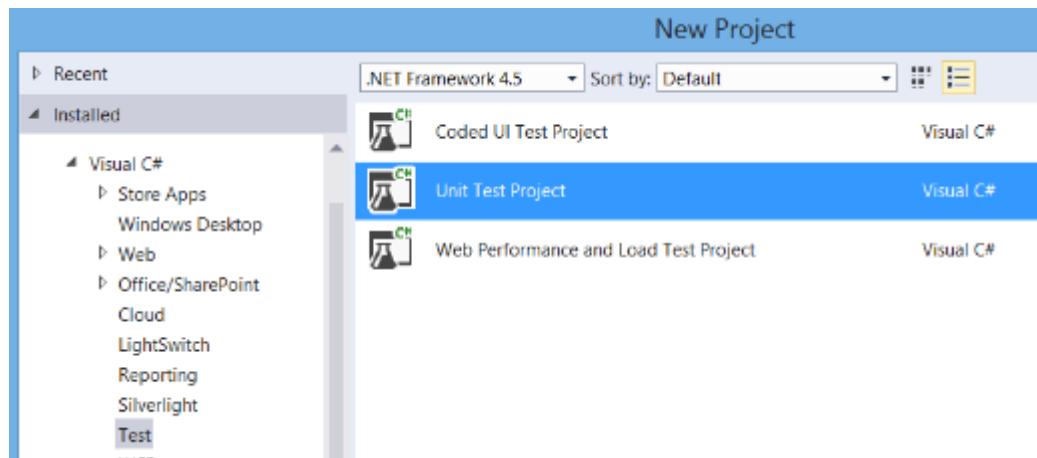
    // Convert the array back to a string and check if reverse string is the same.
    if (textIn == new string(text)) {
        return oldtext + " is a palindrome.";
    } else {
        return oldtext + " is not a palindrome.";
    }
}
```

Unit Testing the Palindrome Code

Go Add New Project

Under Visual C# click on Test

Choose Unit Test Project



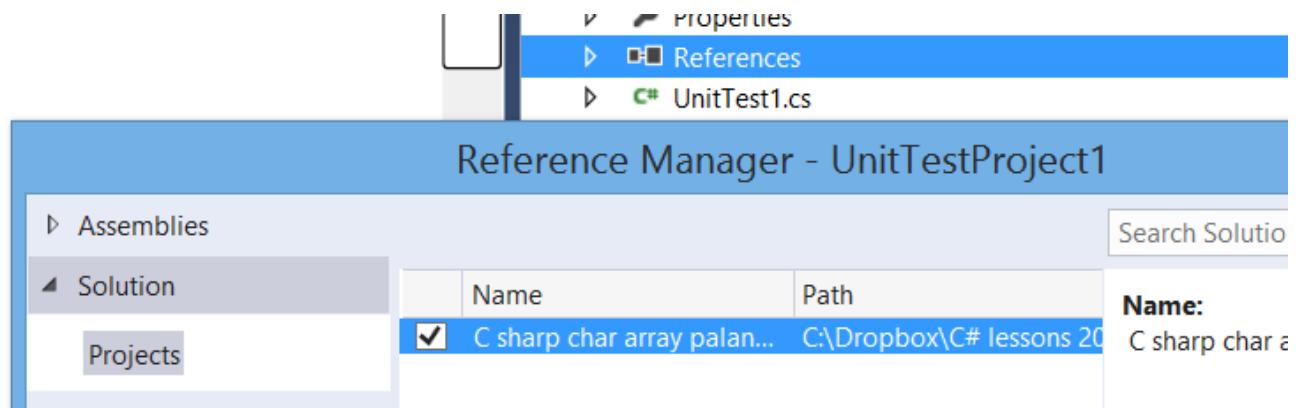
(At this stage the **Solution** at the top disappeared, I turned it off and on and it reappeared but there is a setting that you can change to make it show all the time)

This will give you two projects

We need to make a reference in the Unit test project to the Palindrome project so that they are joined together.

Right click on the Unit Test Project **Reference** and choose **Add Reference**.

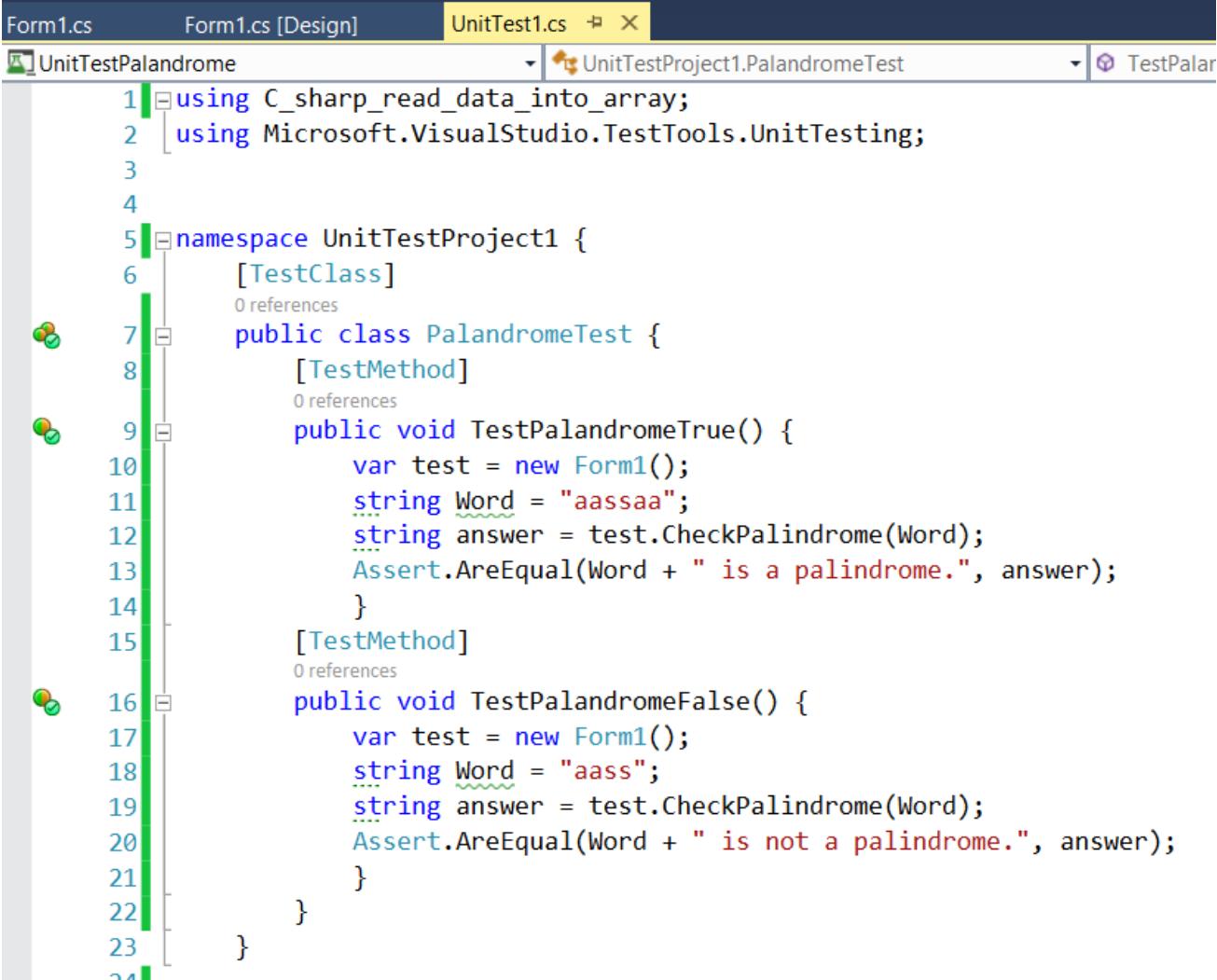
Then tick the reference to your project.



The reference will be added in

The two tests used

- References
 - C sharp char array palandromes
 - Microsoft.VisualStudio.QualityToc
 - System

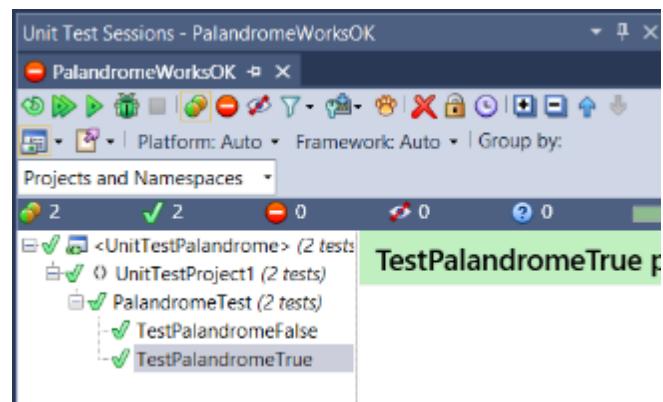


Unit Test Project - UnitTestProject1

```

1  using C_sharp_read_data_into_array;
2  using Microsoft.VisualStudio.TestTools.UnitTesting;
3
4
5  namespace UnitTestProject1 {
6      [TestClass]
7      public class PalindromeTest {
8          [TestMethod]
9          public void TestPalindromeTrue() {
10             var test = new Form1();
11             string Word = "aassaa";
12             string answer = test.CheckPalindrome(Word);
13             Assert.AreEqual(Word + " is a palindrome.", answer);
14         }
15         [TestMethod]
16         public void TestPalindromeFalse() {
17             var test = new Form1();
18             string Word = "aass";
19             string answer = test.CheckPalindrome(Word);
20             Assert.AreEqual(Word + " is not a palindrome.", answer);
21         }
22     }
23 }

```



Array Exercises and Problems – Include Job interview Q's

(Don't skip this part, its where real learning takes place)

Write a program that stores 100 random values in an int array and then displays those values in a ListBox.

Have a Sort button that, when clicked, sorts the values and redisplays them in the ListBox.

How many duplicates are there in these two arrays?

```
int[] a = new int[] {1, 2, 3, 4};  
int[] b = new int[] { 5, 6, 1, 2, 7, 8 };
```

Create an array between 0 and 100. Fill it automatically using a loop.

Modify the program to allow the user to enter a number n in the range from 1 to 10 and have the program print every nth number between 0 and 100. For example, if the user enters 5, then ``0, 5, 10, 15, 20 95, 100`` is printed.

Your fame as a programmer is now beginning to spread far and wide. The next person to come and see you is the chap in charge of the local cricket team.

He would like to you write a program for him to help him analyse the performance of his players.

When a game of cricket is played each member of the team will score a particular number of runs.

What the customer wants is quite simple; given a set of player scores from a game he wants a list of those scores in ascending order.

Create an array to hold the scores and a way to enter them into the program.

Get an Array of words such as `string[] Words = {"a","b","a","b"};`

Copy it to a **new** array.

Reverse the new array.

Compare the two arrays and return **True** if the reversed array is the same as the first array

Watch this for help [csharp-collections-fundamentals](#)

Problems from Programming Interviews Exposed 3rd Edition, Chapter 6.

These are the sort of questions you can expect at a job interview.

PROBLEM Write an efficient function to find the first nonrepeated character in a string.

For instance, the first nonrepeated character in “total” is 'o' and the first nonrepeated character in “teeter” is 'r'.

This might use a couple of nested loops.

What about “The quick brown fox jumped over the lazy dog”

PROBLEM Write an efficient function that deletes characters from an ASCII string. Use the prototype `string removeChars(string str, string remove);` where any character existing in remove must be deleted from str.

For example, given a str of “Battle of the Vowels: Hawaii vs. Grozny” and a remove of “aeiou”, the function should transform str to “Bttl f th Vwl: Hw vs. Grzny”.

This interesting exercise would use two arrays, one to hold the vowels, and the other to hold the string. Then loop through the arrays and remove matching letters.

PROBLEM Write a function that reverses the order of the words in a string.

For example, your function should transform the string “Do or do not, there is no try.” to “try. no is there not, do or Do”.

Assume that all words are space delimited and treat punctuation the same as letters.

Ideas ... You need to find each word, as identified by the space, then pass each word to an array, then return the array in a reverse order. However you also have to keep the punctuation in the correct place.

The most obvious approach is to use to identify words, write these words into a temporary buffer, and then copy the buffer back over the original string. To reverse the order of the words, you must either scan the string backward to identify the words in reverse order or write the words into the buffer in reverse order (starting at the end of the buffer).

Write a function that takes an array of consecutive numbers (1,2,3,4 … n) and then shuffles them to produce a random selection.

Shuffle functions are easy to write, fast to implement, and are really handy to know. Use it in the next exercise.

Problem. I have an array stockPricesYesterday where:

The indices are the time, as a number of minutes past trade opening time, which was 9:30am local time.

The values are the price of Apple stock at that time, in dollars.

For example, the stock cost \$500 at 10:30am, so `stockPricesYesterday[60] = 500.`

Write an efficient algorithm for computing the best profit I could have made from 1 purchase and 1 sale of 1 Apple stock yesterday. For this problem, we won't allow "shorting"—you must buy before you sell.

Answers on the link

<https://www.interviewcake.com/>

Change calculator using Arrays and Loops

This program contains two arrays and three loops.

The first array holds the values of the notes/coins used to calculate the change = { 20, 10, 5, 1, 0.5, 0.1, .05, .01 }.

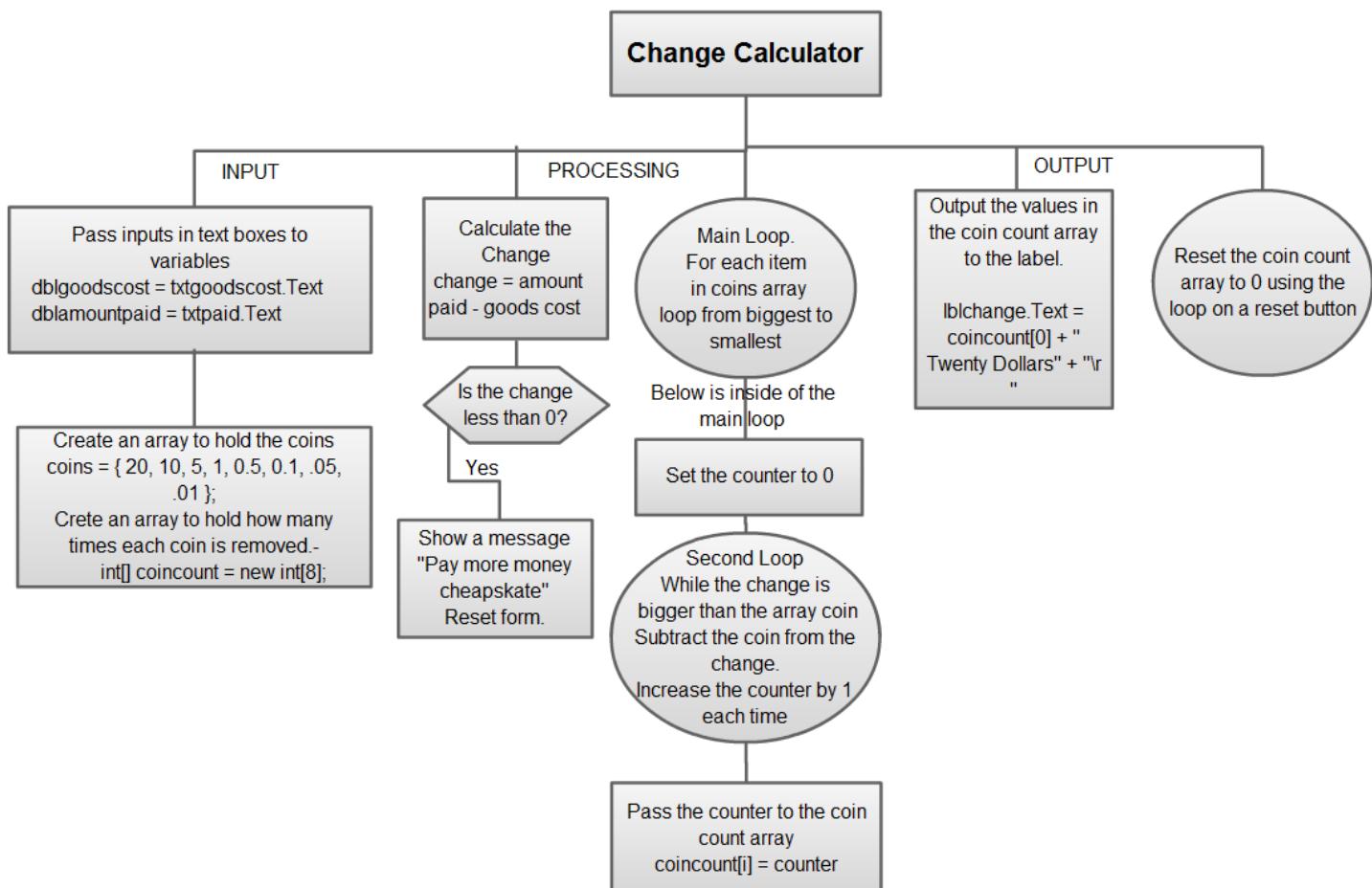
The second array is empty and holds the count of how many times that coin goes into the change `coincount = new int[8];`. The elements of this array are outputted to the label to show the change to be given.

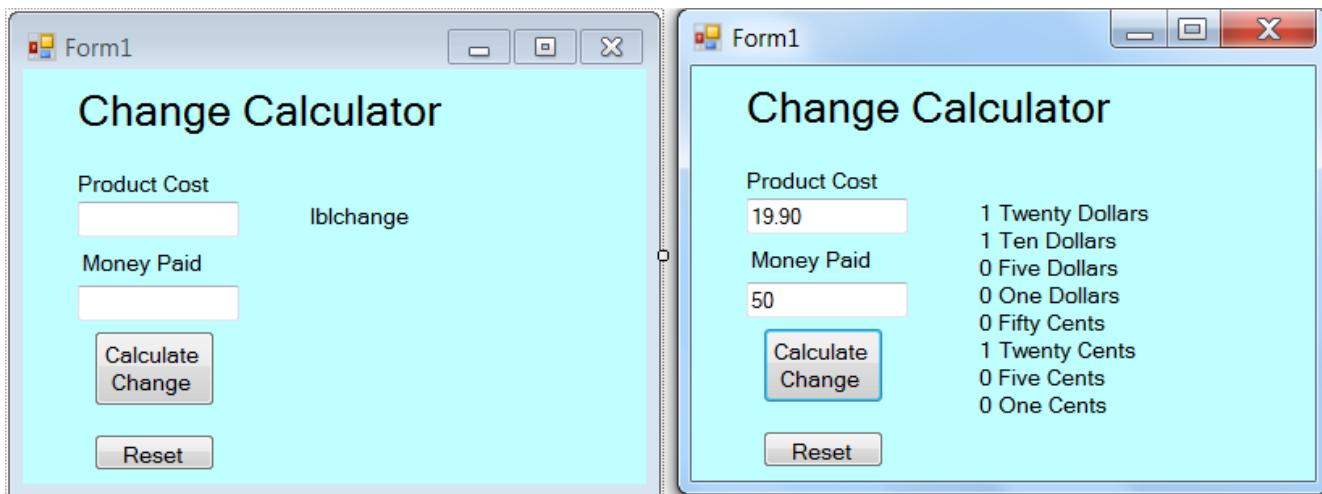
The **first loop** goes through each of the coins in the array. The **second loop** is **inside the first loop** and counts how many times the coin can be subtracted from the change.

Each time it loops it subtracts the coin from the change and the counter increases by one.

The output passes the second array to a label. `Coincount[0]`, `Coincount[1]` etc.

Finally we create another loop with the reset button, as well as resetting everything else, that resets the `coincount` array elements back to 0.





```
public partial class Form1 : Form
{
    double dblgoodscost, dblamountpaid, dblchange = 0;
//coin value to subtract - 8 coins
    double[] coins = { 20, 10, 5, 1, 0.5, 0.1, .05, .01 };
//how many times each coin is removed.- 8 places array is 0 to 7
    int[] coincount = new int[8];
```

```
private void btncalculate_Click(object sender, EventArgs e)
{
    dblgoodscost = Convert.ToSingle(txtgoodscost.Text);
    dblamountpaid = Convert.ToSingle(txtpaid.Text);
    dblchange = dblamountpaid - dblgoodscost;

    if (dblchange < 0)
    {
        MessageBox.Show("Pay more money");
        return;
    }
//counter to count through the coins array so a value can be put into the
coincount
    for (int i = 0; i < coins.Length; i++)
    {
        //counter to show how many time the foreach loop runs
        int counter = 0;
//loop while there is still more change than the coin
        while (dblchange >= coins[i])
        {
            //subtract the coin value from the change
            dblchange = dblchange - coins[i];
            //increase the counter by one each time
            counter +=1;
        }
        //after that loop
        coincount[i] = counter;
    }
}
```

```
//print out on the label
lblchange.Text = coincount[0] + " Twenty Dollars" + "\r\n" +
coincount[1] + " Ten Dollars" + "\r\n" + coincount[2] + " Five Dollars" + "\r\n" +
coincount[3] + " One Dollars" + "\r\n" + coincount[4] + " Fifty Cents" + "\r\n" +
coincount[5] + " Twenty Cents" + "\r\n" + coincount[6] + " Five Cents" + "\r\n" +
coincount[7] + " One Cents";
}

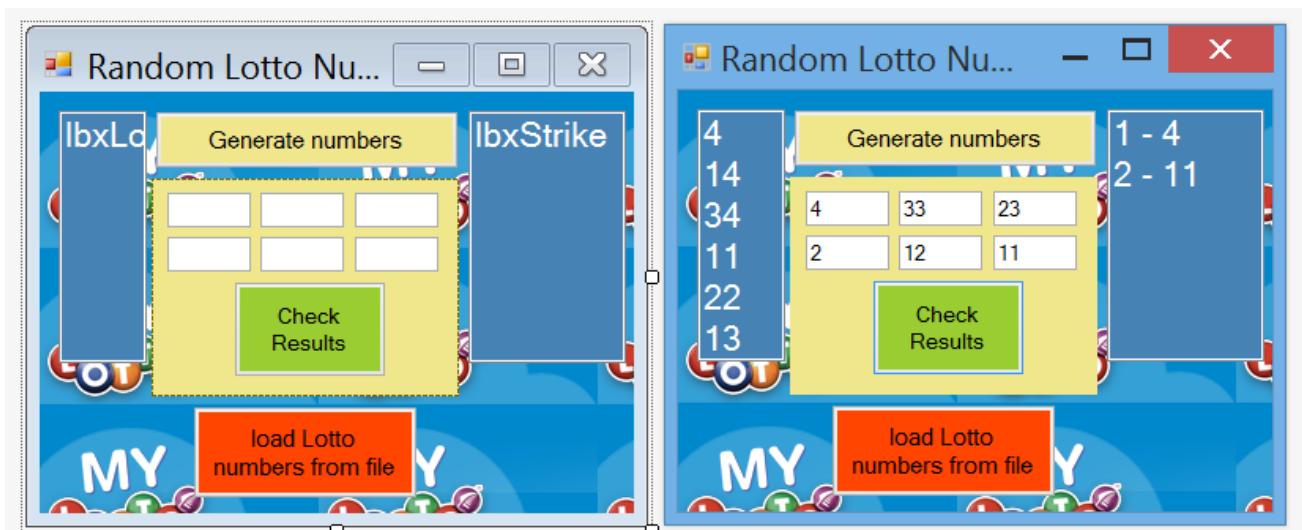
private void btnreset_Click(object sender, EventArgs e)
{
    for (int i = 0; i < coincount.Length; i++)
    {
        coincount[i] = 0;
    }
    txtgoodscost.Text = "";
    txtpaid.Text = "";
    lblchange.Text = "";
}
```

Lotto number Generator and Checker – Shuffle function

This is a simple program but has some good coding ideas to use such as the shuffle array and good coding practice.

This program has two parts,

1. Create a **random number generator** that creates 6 numbers between 1 and 45. Use a shuffle function.
2. Check your own six numbers in the text boxes against those random numbers
3. record how many when they match.
4. No code under the button click apart from the method name to the method



<https://mylotto.co.nz/lotto/results/>

The final section, which we can do in the future is to load in lotto numbers from a file, and check them against the generated ones.

Here is the complete code without the load button. **Only use it if you are stuck.**

```
public partial class Form1 : Form
{
    //define an array
    int[] lotto = new int[6];
    int[] luckynumbers = new int[6];
    Random myrandom = new Random();

    public Form1()
    {
        InitializeComponent();
    }

    //generate random number button
    private void button1_Click(object sender, EventArgs e)
    {
        lboutput.Items.Clear();

        //loop to create the random numbers.
        for (int i = 0; i < 6; i++)
        {
            int answer = myrandom.Next(1, 41);
            lotto[i] = answer;
            lboutput.Items.Add(answer);
        }

        //adding numbers to array with tryparse to check that they are numbers.
        int.TryParse(txt1.Text, out luckynumbers[0]);
        int.TryParse(txt2.Text, out luckynumbers[1]);
        int.TryParse(txt3.Text, out luckynumbers[2]);
        int.TryParse(txt4.Text, out luckynumbers[3]);
        int.TryParse(txt5.Text, out luckynumbers[4]);
        int.TryParse(txt6.Text, out luckynumbers[5]);
    }

    //check results button
    private void button2_Click(object sender, EventArgs e)
    {
        int strike = 0;
        lboutput.Items.Clear();

        foreach (int item in lotto)
        {
            foreach (int number in luckynumbers)
            {
                if (item == number)
                {
                    strike++;
                    lboutput.Items.Add("Strike! " + item);
                    btncheck.Text = Convert.ToString(strike);
                }
            }
        }
    }
}
```

```
        }
    }
}
}
```

The problem with the above code is that you can have multiple numbers. The way to fix this is quite easy, but means that we lose our nice nested loops, for a simple Do –While loop.

```
//generate random number button
private void button1_Click(object sender, EventArgs e)
{
    lboutput.Items.Clear();

    //loop to create the random numbers.
    int counter = 0;

    do //loop until there are six numbers
    {
        //generate a random number
        int answer = myrandom.Next(1, 41);

//set a boolean to false that checks if the number is increasing
        bool isexisting = false;

        //loop through the array of existing numbers
        foreach (var number in lotto)
        {
            //if the number is already in the array
            if (answer == number)
            {
                //set the existing to true - we don't add that number in
                isexisting = true;
            }
        }
        //if the number is NOT in the array
        if (isexisting == false)
        {
            //add the number in and increment the counter by 1
            lotto[counter] = answer;
            lboutput.Items.Add(answer);
            counter++;
        }
        //do this loop while the counter is less than 6
    } while (counter < 6);
}
```

C#7 Using Tuples to return data from Array

This line `min = (n < min) ? n : min;`. Is a cut down version of an IF statement, you can see it in its original form [here](#)

The [conditional operator](#) (`?:`) returns one of two values depending on the value of a Boolean expression. Following is the syntax for the conditional operator.

condition ? first_expression : second_expression;

The condition must evaluate to true or false. If condition is **true**, first_expression is evaluated and becomes the result. If condition is **false**, second_expression is evaluated and becomes the result. **Only one of the two expressions is evaluated.**

You can express calculations that might otherwise require an **if-else** construction more concisely by using the conditional operator.

```
// Using Tuples to extract out max and Min
private (int Max, int Min) Range(int[] numbers)
{
    int min = int.MaxValue;
    int max = int.MinValue;
    foreach (var n in numbers)
    {
        // Below are cut down IF statements
        // IF n< min then do whats after the ? in this case pass n to min min = n, otherwise pass
        // the existing min to the min.
        min = (n < min) ? n : min;
        max = (n > max) ? n : max;
    }
    return (max, min);
}

private void button1_Click(object sender, EventArgs e)
{
    var numbers = new int[] { 1, 2, 3, 100, 180, 2250 };
    var range = Range(numbers);

    this.Text = $"The numbers supplied range from {range.Min} to {range.Max}.";
}
```

13. Multidimensional Arrays

Multidimensional arrays are for table like data, where you use more than one index to access elements.

Two indices

First index

Second index

```
double[ , ] heights = new double[50, 100];
```

Or for the next exercise `var timetable = new int[12, 12];`

Sometimes we want to hold more than just a row. Sometimes we want a grid.

We can do this by creating a two dimensional array. For example, to hold the board for a game of noughts and crosses (tic tac toe) we could use:

```
int [,] board = new int [3,3];
board [1,1] = 1;  (below see 1,1 on the grid)
```

| | | 0 | 1 | 2 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | |
| 1 | 0 | 1 | 0 | |
| 2 | 0 | 0 | 0 | |

This looks very like our one dimensional array, but there are some important differences. The `[,]` now has a comma. The presence of a comma implies something each side of it.

This means that the array now has two dimensions, rather than just one. So when we give the size of the board we must supply two dimensions rather than just one.

Then, when we want to specify an element we have to give two subscript values. You can think of a 2D array as a grid:

The first subscript will specify which **row** and the second which **column**. (X,Y)

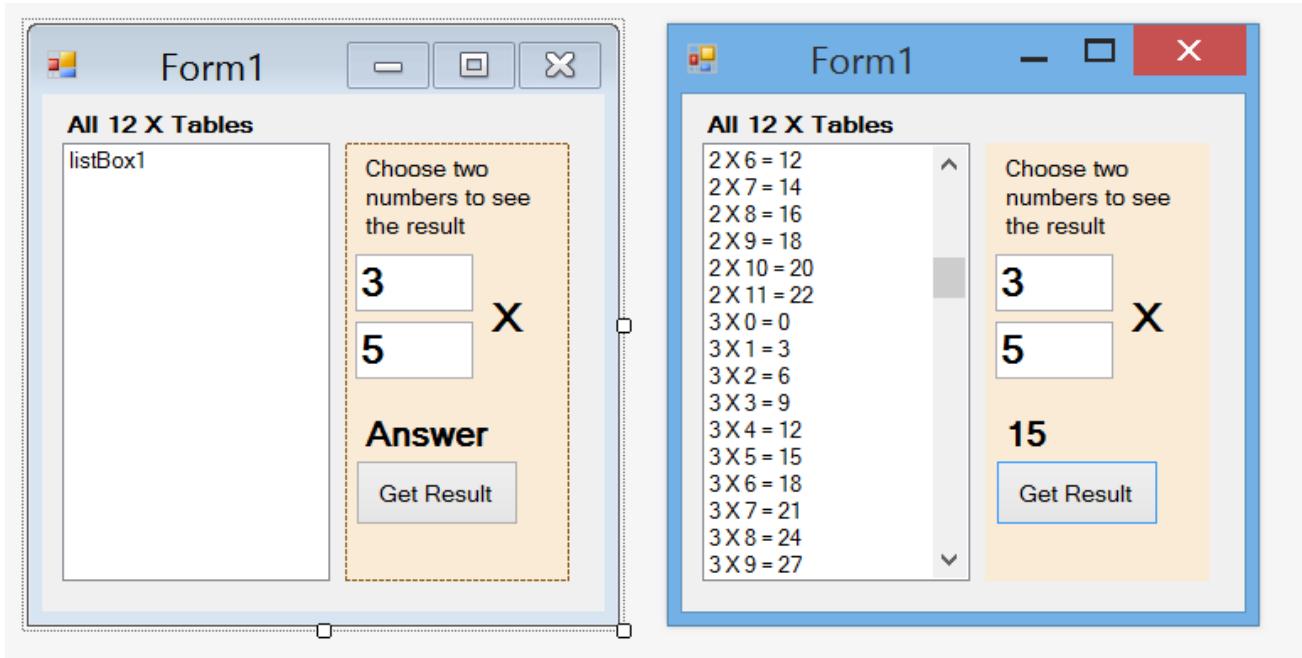
In the example above the array is square (i.e. the same dimension across as up). We can change this if we like:

`int [,] board = new int [3,10];` to make a rectangle with 3 rows and 10 columns.

Two dimensional array - Times table program

This two dimensional array acts as a times table. Each element in the array is specified by column and row, values, i.e.: 2,3. If you multiply those **xy** cords together you get 6, and that number is placed at that point in the array.

The **Get Result** button runs the array and the two text boxes add in X Y cords to get an array element giving us a calculator!



The heart of this program is creating a new two dimensional array.

```
var timetable = new int[12, 12];
```

This holds a 12 x 12 grid of elements, each accessible by its own address for example 4,5 and 4,6. Into these Index positions we add in the element we want to store, in this case the multiplication of the two indexes.

The double **For** loop counts through every combination of the two arrays, does the calculation and adds in as an element.

All 12 X Tables

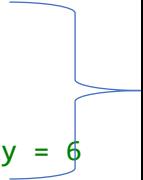
| |
|-------------|
| 4 X 5 = 20 |
| 4 X 6 = 24 |
| 4 X 7 = 28 |
| 4 X 8 = 32 |
| 4 X 9 = 36 |
| 4 X 10 = 40 |
| 4 X 11 = 44 |
| 5 X 0 = 0 |
| 5 X 1 = 5 |
| 5 X 2 = 10 |
| 5 X 3 = 15 |
| 5 X 4 = 20 |
| 5 X 5 = 25 |
| 5 X 6 = 30 |
| 5 X 7 = 35 |
| 5 X 8 = 40 |

```

private void btmultiarray_Click(object sender, EventArgs e)
{
    lbxTimestable.Items.Clear();
        //int[Column,Row];
    var timetable = new int[12, 12];
        //load data in based on elements multiplied by each other this gets
the length of the first row 0 and the second if gets the second row 1
    for (int x = 0; x < timetable.GetLongLength(0); x++)
    {
        for (int y = 0; y < timetable.GetLongLength(1); y++)
        {
            //this puts the sum of xy in the array at that positon ie: x=2, y=3 xy = 6
            timetable[x, y] = x * y;// [3,2] = 6
            //X plus Y equals the value held at X,Y
        lbxTimestable.Items.Add(x + " X " + y + " = " + timetable[x, y]);
        }
    }

    //here we enter in to the program your own numbers to find the
element at the xy position.
    int Xin = Convert.ToInt16(txtX.Text);
    int Yin = Convert.ToInt16(txtY.Text);
        //this shows the answer in the label that is returned
    lblanswer.Text = Convert.ToString(timetable[Xin, Yin]);
}

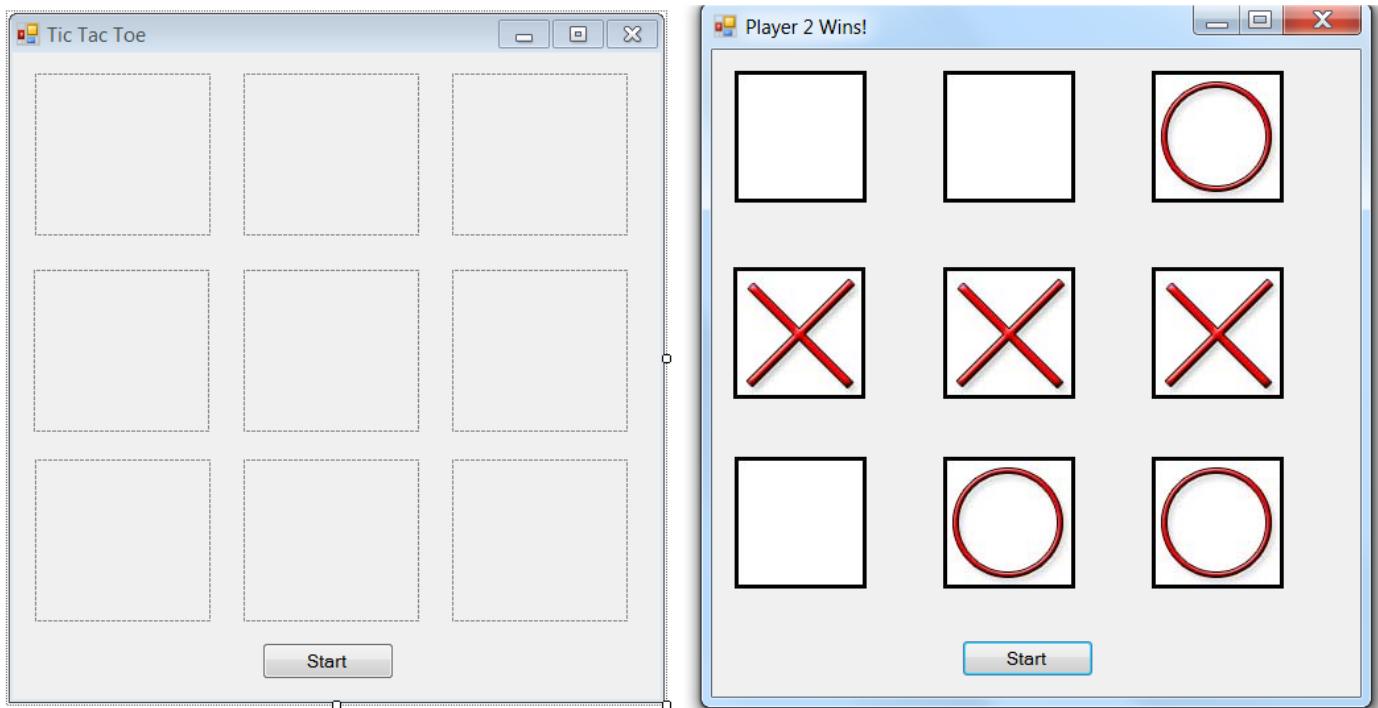
```



With this basic code change the program so that you can have your own array of any size you like

Watch the **C# Collections Fundamentals** video for ideas.

Tic Tac Toe game – 2 Dimensional Array of Pictureboxes



This exercise involves a little thinking and makes it a good beginner task.



Steps.

1. Load in the title images to a resource file (I called mine Tiles)
2. Make 9 picture boxes, set their background image to the Blank Tile
3. Name them pictureBox1, pictureBox2, pictureBox3, etc. These will be held in an array of Pictureboxes as seen over the page.

Read about TicTacToe [here](#)

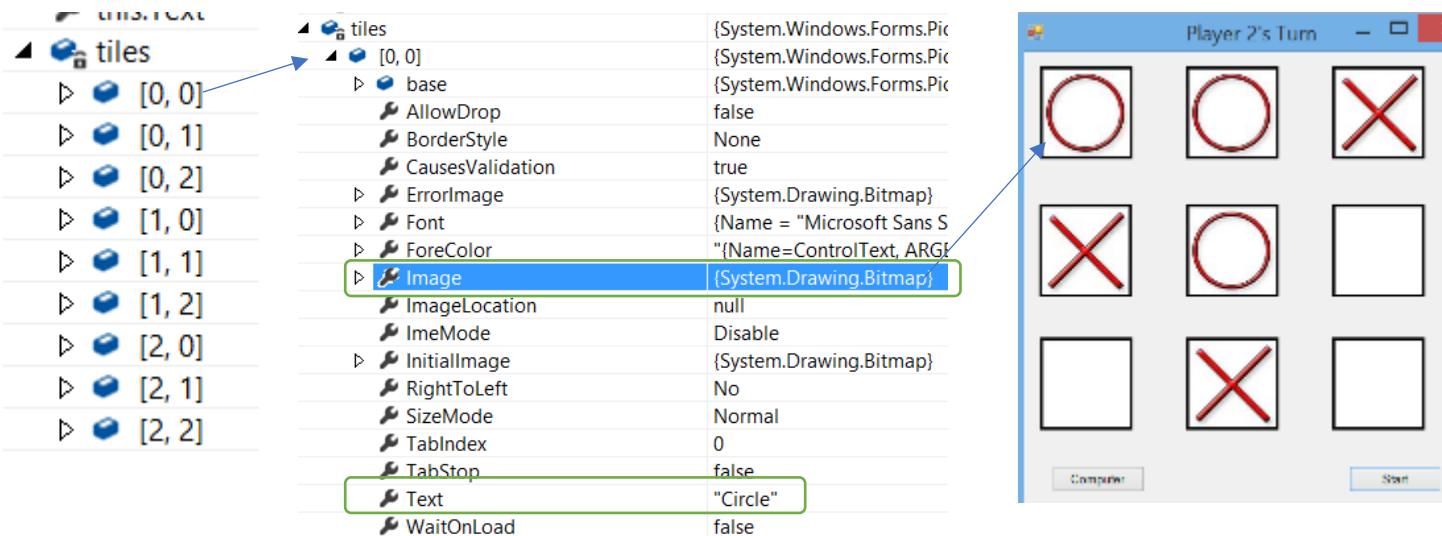
This exercise is based on **a two dimensional array of Pictureboxes**. (cool isn't it). We know it's an array of Pictureboxes because of the `Picturebox[,]` with the comma in the brackets showing its two dimensional.

```
public partial class Form1 : Form {
    private PictureBox[,] tiles; //this is declared globally, so you can use it all
    through the Form1 code.
```

```
public Form1() {
    InitializeComponent();
    tiles = new PictureBox[,]
    //3 x 3 array instantiated in the constructor of the form to give it values.
    { { pictureBox1, pictureBox2, pictureBox3 },
      { pictureBox4, pictureBox5, pictureBox6 },
      { pictureBox7, pictureBox8, pictureBox9 } };
```

Each picturebox in the array is the name of the picturebox on the form.

That way all the properties of the picturebox can be held in the array such as the image and any text or tags.



What events are we going to have to code?

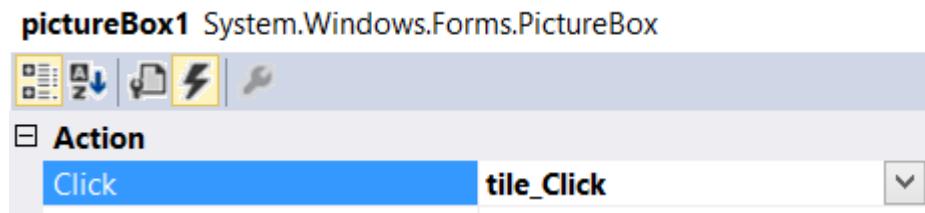
Clicking on a tile, Clicking on any tile triggers the same event method. This creates a fake tile and using sender passes across all the properties of the picturebox you clicked on.

That gets passed to the CheckTile method.

```
private void tile_Click(object sender, EventArgs e) {

    //using sender means that any tile clicked on will run this code
    PictureBox faketile = (PictureBox)sender;
    CheckTile(faketile);
}
```

Make sure you attach the tile_Click to the click event of each picturebox.



Which player is current (Player 1 or Player 2),

If it player 1 then it gets a Circle, or if its 2 it gets a Cross. Also the Tag property of the tile gets the words Circle or Cross added. It's the Tag property that we check to see if we have 3 in a row.

Although I think it's easier to use the tag property to check which tile has which image, it could introduce unneeded complexity as you can also check the tile.image as well.

```
switch (ThisPlayer.CurrentPlayer) {
    case 1:
        tile.Image = Resource1.o_tile;
        tile.Text = "Circle";
        break;
    case 2:
        tile.Image = Resource1.x_tile;
        tile.Text = "Cross";
        break;
}
```

Checking the board to see if a move has won the game. This is probably the biggest section. Fundamentally you need to check all the files vertically, horizontally and diagonally every time a player puts a tile down to look for 3 in a row.

However those three must not be blank tiles, only Cross or Circle.

To get the diagonal tiles you need to hard code in the tiles.

If the tile Text at [0,0] is not equal to Blank, and if it is the same as tile[1,1], and if tile[1,1] is the same as tile[2,2] then the person wins.

```
tiles[0, 0].Text.ToString() != "Blank" && tiles[0, 0].Text.Equals(tiles[1, 1].Text)
&& tiles[1, 1].Text.Equals(tiles[2, 2].Text)
```

You can only check 2 tiles at a time for equality, so some repetition comes into play. Note && for Equals, and || for Or.

To check vertical and Horizontal, either hard code the tiles or much more nicer, use a for loop that loops through the check. .

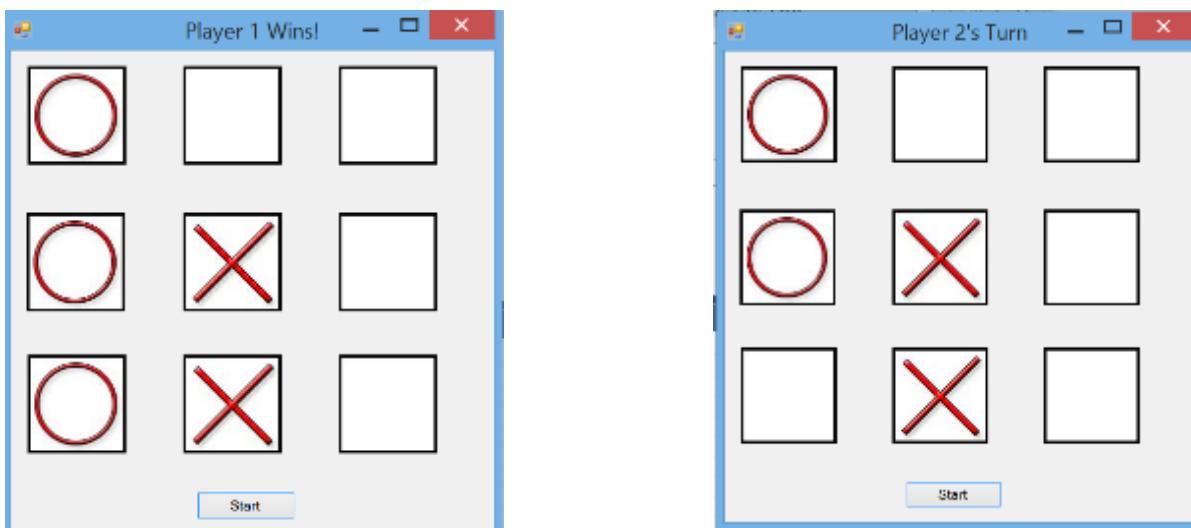
Checking the board to see if there are any spaces left free.

Create a method to work out the current player, then code it so a click on the picturebox swaps Player 1 with Player 2. (an iF / Else statement works well here)

We want an array of objects, in this case Pictureboxes.

```
private PictureBox[,] tiles;
tiles = new PictureBox[,] //3 x 3 array
    { { pictureBox1, pictureBox2, pictureBox3 },
      { pictureBox4, pictureBox5, pictureBox6 },
      { pictureBox7, pictureBox8, pictureBox9 } };
```

Create a method to clean it up for the next game. Note the Title text that tells who's turn it is and who has won.



```
public partial class Form1 : Form
{
    int currentPlayer = 1;
    private int winner = 0;
    private int turnCounter = 0;
    private PictureBox[,] tiles;

    public Form1()
    {
        InitializeComponent();
    tiles = new PictureBox[3, 3] {
{ pictureBox1, pictureBox2, pictureBox3 },
{ pictureBox4, pictureBox5, pictureBox6 },
{ pictureBox7, pictureBox8, pictureBox9 } };
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        ResetSquares();
    }

    private void buttonStart_Click(object sender, EventArgs e)
    {
        ResetSquares();
    }

    private void tile_Click(object sender, EventArgs e)
    {
        PictureBox Faketile = (PictureBox)sender;

        if (Faketile.Text != "Blank" || winner != 0)
            return;
        //what is interesting to note is that the faketile can automatically send
        changes to its properties back to the real tile I the array to update.
        if (currentPlayer == 1)
        { //player 1 is always O
            Faketile.Image = Resource1.o_tile;
            Faketile.Text = "Circle";
            this.Text = "Player 2's Turn";
        }
        else
        { // player 2 is always X
            Faketile.Image = Resource1.x_tile;
            Faketile.Text = "Cross";
            this.Text = "Player 1's Turn";
        }
        turnCounter++;
        winner = DetermineWinner();
    }
}
```

```
if (winner != 0)
    this.Text = String.Format("Player {0} Wins!", winner);
else if (winner == 0 && turnCounter == 9)
    this.Text = "Draw!";
else
    currentPlayer = currentPlayer == 1 ? 2 : 1;
}

private void ResetSquares()
{
    foreach (PictureBox tile in tiles)
    {
        tile.Image = Resource1.blank_tile;
        tile.Text = "Blank";
    }

    currentPlayer = 1;
    winner = 0;
    turnCounter = 0;
    this.Text = "Player 1's Turn";
}

private int DetermineWinner()
{
    if (
        //Check for diagonals
        ( tiles[0, 0].Text != "Blank" &&
            tiles[0, 0].Text.Equals(tiles[1, 1].Text) &&
            tiles[1, 1].Text.Equals(tiles[2, 2].Text)) ||
        ( tiles[0, 2].Text != "Blank" &&
            tiles[0, 2].Text.Equals(tiles[1, 1].Text) &&
            tiles[1, 1].Text.Equals(tiles[2, 0].Text)) ||
        //Check for horizontals
        ( tiles[0, 0].Text != "Blank" &&
            tiles[0, 0].Text.Equals(tiles[0, 1].Text) &&
            tiles[0, 1].Text.Equals(tiles[0, 2].Text)) ||
        ( tiles[1, 0].Text != "Blank" &&
            tiles[1, 0].Text.Equals(tiles[1, 1].Text) &&
            tiles[1, 1].Text.Equals(tiles[1, 2].Text)) ||
        ( tiles[2, 0].Text != "Blank" &&
            tiles[2, 0].Text.Equals(tiles[2, 1].Text) &&
            tiles[2, 1].Text.Equals(tiles[2, 2].Text)) ||
        //Check for verticals
        ( tiles[0, 0].Text != "Blank" &&
            tiles[0, 0].Text.Equals(tiles[1, 0].Text) &&
            tiles[1, 0].Text.Equals(tiles[2, 0].Text)) ||
        ( tiles[0, 1].Text != "Blank" &&
```

```

        tiles[0, 1].Text.Equals(tiles[1, 1].Text) &&
        tiles[1, 1].Text.Equals(tiles[2, 1].Text)) ||
    ( tiles[1, 2].Text != "Blank" &&
        tiles[0, 2].Text.Equals(tiles[1, 2].Text) &&
        tiles[1, 2].Text.Equals(tiles[2, 2].Text))
    ) //close if condition
{
    return currentPlayer;
}
else
    return 0;
}
}
}

```

Here is a better way using a For loop to determine the winner that I used in part 2.

```

public int DetermineWinner(PictureBox[,] tiles) {
    for (int i = 0; i < 3; i++) {
        //horizontal and vertical
        if (tiles[i, 0].Text.ToString() != "Blank" && tiles[i, 0].Text == tiles[i, 1].Text
        && tiles[i, 1].Text == tiles[i, 2].Text
            ||
        tiles[0, i].Text.ToString() != "Blank" && tiles[0, i].Text == tiles[1, i].Text &&
        tiles[1, i].Text == tiles[2, i].Text
            ) {
            return currentPlayer; //Check for diagonals
        }
    }

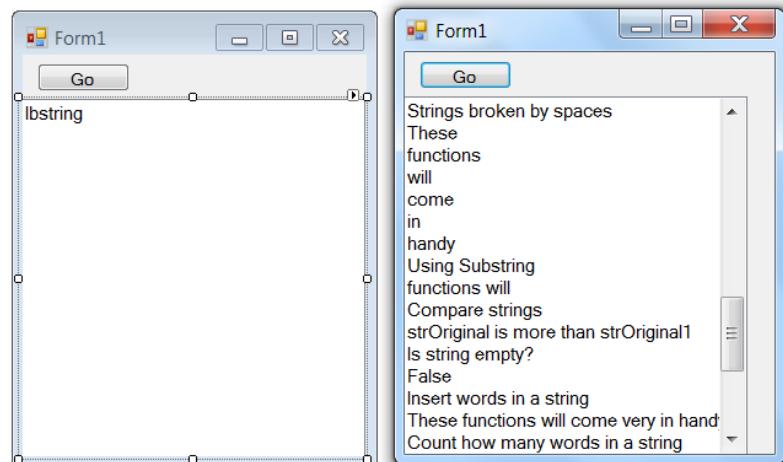
        if ( //Check for diagonals
        tiles[0, 0].Text.ToString() != "Blank" && tiles[0, 0].Text.Equals(tiles[1, 1].Text)
        && tiles[1, 1].Text.Equals(tiles[2, 2].Text)
            ||
        tiles[0, 2].Text.ToString() != "Blank" && tiles[0, 2].Text.Equals(tiles[1, 1].Text)
        && tiles[1, 1].Text.Equals(tiles[2, 0].Text)) {
            return currentPlayer;
        }
    return 0; //the default if no match is to return 0
}

```

14. String Operations

Create these practice exercises on using strings. We will incorporate these into future exercises. **Otherwise skip this and go to the exercises and use it as a reference.**

- String to Characters
- Split String
- Substring
- Compare string
- Is empty string
- Insert into string
- Count words
- Replace words
- Remove words
- Create time and date
- Is string a number
- Does string start with ...



Create a button, btnGo and a ListBox lbxString.

Place these two strings at the top of the code

```

public partial class Form1 : Form
{
    string strOriginal = "These functions will come in handy";
    string strModified = String.Empty;

    public Form1()
    
```

Underneath the Go button add all the different string functions below. You might like to comment out lbxString.Items.Add(c.ToString()); when you get it to run as it takes up more space.

```

private void btngo_Click(object sender, EventArgs e)
    {
    
```

Breaking a word down to individual letters (characters)

Char is character, such as a letter or symbol.

```

//=====
lbString.Items.Add("Strings to characters");
//this breaks the words down to a character and displays each one
foreach (char c in strOriginal)
{
    lbString.Items.Add(c.ToString());
}

```

Breaking sentences down into words

```

string[] words = strOriginal.Split(' ');

```

```
strOriginal = strOriginal.Trim();
int counter = 0;

foreach (string word in words)
{
    lbstring.Items.Add(word);
}
```

Split a string looking for the space.

The Char this time holds the space, the gap between the words. So delim is just the variable name for the char which is holding a space.

```
//=====
lbstring.Items.Add("Strings broken by spaces");
char[] delim = { ' ' };
string[] strArr = strOriginal.Split(delim);
foreach (string s in strArr)
{
    //Comment out the messagebox when it bugs you too much
    //MessageBox.Show(s);
    lbstring.Items.Add(s);
}
```

Using substring

```
//=====
lbstring.Items.Add("Using Substring");
// Substring. Starting position and length of string to be extracted specified
strModified = strOriginal.Substring(6, 15);
    //    MessageBox.Show(strModified);
lbstring.Items.Add(strModified);
```

Compare two strings

```
//=====
lbstring.Items.Add("Compare strings");
//compare two strings and show if they are less than, the same, or more than.
if ((string.Compare(strOriginal, strModified, false)) < 0)
{
    lbstring.Items.Add("strOriginal is less than strOriginal1");
}
else if ((string.Compare(strOriginal, strModified, false)) > 0)
{
    lbstring.Items.Add("strOriginal is more than strOriginal1");
}
else if ((string.Compare(strOriginal, strModified, false)) == 0)
{
    lbstring.Items.Add("Both strings are equal");
}
```

Check for empty string

```
//=====
//Is there an empty string?
lbstring.Items.Add("Is string empty?");
bool check = String.IsNullOrEmpty(strOriginal);
lbstring.Items.Add(check);
```

Add words to a string at a set place

```
//=====
//Add words to a string
lbstring.Items.Add("Insert words in a string");
strModified = strOriginal.Insert(30, "very ");
lbstring.Items.Add(strModified);
```

Count words in a string

```
//=====
//Count words in a string. - using Index OF a very handy tool
lbstring.Items.Add("Count how many words in a string");

// Using IndexOf
int start = 0;
int count = -1;
int idx = -1;

strOriginal = "She sells sea shells on the sea shore";
string srchString = "sea";
while (start != -1)
{
    start = strOriginal.IndexOf(srchString, idx + 1);
    count += 1;
    idx = start;
}
lbstring.Items.Add(srchString + " occurs " + count + " times");
```

Replace words in a string

```
//=====
//Replace words in a string
lbstring.Items.Add("Replace words in a string");
strModified = strOriginal.Replace("come handy", "be useful");
lbstring.Items.Add(strModified);
```

Remove words from a string

```
//=====
//Remove words from a string
lbstring.Items.Add("Remove words in a string");
    // Removes everything beginning at index 25
strModified = strOriginal.Remove(25);
lbstring.Items.Add(strModified);
```

Create time and date from a string

```
//=====
//Creating time and date from a string
lbstring.Items.Add("Creating date and Time from a string");
```

```
string strdate = "08/02/2008";
DateTime dt = Convert.ToDateTime(strdate);
lbstring.Items.Add("day " + dt.Day + " month " + dt.Month + " year " + dt.Year);
```

Check if string is a number

```
//=====
lbstring.Items.Add("Is the string a number?");
int i = 0;
strOriginal = "234";
bool b = Int32.TryParse(strOriginal, out i);
lbstring.Items.Add(strOriginal + " " + b);
```

Check if string starts with certain words

```
//=====
lbstring.Items.Add("Does the string start with These?");
strOriginal = "these are the words";
if (strOriginal.StartsWith("These", StringComparison.CurrentCultureIgnoreCase))
    lbstring.Items.Add(strOriginal + " true");
}
```

The String.Format and String Interpolation Methods

Use String.Format if you need to insert the value of an object, variable, or expression into another string. For example, you can insert the value of a Decimal value into a string to display it to the user as a single string:

```
Decimal pricePerOunce = 17.36m;  
String s = String.Format("The current price is {0} per ounce.", pricePerOunce);  
// Result: The current price is 17.36 per ounce.
```

See the following Examples

<https://msdn.microsoft.com/en-us/library/system.string.format%28v=vs.110%29.aspx?cs-save-lang=1&cs-lang=csharp#code-snippet-1>

String.Format and its cousins are very versatile and useful, but their use is a little clunky and error prone.

Particularly unfortunate is the use of {0} etc. placeholders in the format string, which must line up with arguments supplied separately:

```
var s = String.Format("{0} is {1} year{{s}} old", Name, Age);
```

C# 6 String Interpolated

An interpolated string expression looks like a template string that contains expressions. An interpolated string expression creates a string by replacing the contained expressions with the ToString representations of the expressions' results.

String interpolation lets you put the expressions right in their place, by having “holes” directly in the string literal:

```
var s = $"{Name} is {Age} year{{s}} old";
```

Just as with String.Format, optional alignment and format specifiers can be given:

```
var s = $"{Name,20} is {Age:D3} year{{s}} old";
```

The contents of the holes can be pretty much any expression, including even other strings:

```
var s = $"{Name} is {Age} year{(Age == 1 ? "" : "s")} old";
```

Notice that the conditional expression is parenthesized, so that the : "s" doesn't get confused with a format specifier.

<https://github.com/dotnet/roslyn/wiki/New-Language-Features-in-C%23-6>

<https://msdn.microsoft.com/en-us/library/dn961160%28v=vs.140%29.aspx>

String Performance Tips

As strings are immutable, which means you can't change them, then every time you concatenate the string you are creating another string.

Here are two examples, Var S and Var T.

```
public static void Run()
{
    var s = "This" + " is" + " a" + " string";           I
    Console.WriteLine(s);

    var v = 15;
    var t = "This is " + v.ToString() + " a value";
    Console.WriteLine(t);
```

So the **var s** example above looks like this to the compiler

1. This
2. is
3. This is
4. a
5. This is a
6. string
7. This is a string
8. s = This is a string

As a result there are 8 strings that the garbage collector has to clean up. The compiler can optimise this example for you but the second example will not.

The solution is to use **string.concat** that will be a highly efficient operation that will not produce the 8 strings.

For the second example `String.Format` allows you to use parameters and you don't need the `.ToString()` function.

`v.ToString()` is determined at run time so this won't be optimised.

```
public static void Run()
{
    var s = string.Concat("This", " is", " a", " string");
    Console.WriteLine(s);

    var v = 15;
    var t = string.Format("This is {0} a value", v);
    Console.WriteLine(t);

}
```

Optimise Line breaks

If you are trying to generate line breaks by making each line add more text like this...

```
var x = "This is the start.";
x += "Some more text";
x += "And even more";
```

Then use the `@` symbol that turns the text into literal characters and means that line breaks are automatically included.

```
var y = @"
This is the start.
Hello world.
Hi there.";
```

The best way if you are building up a large amount of text is to use `StringBuilder` that will add a line break at the end of `.AppendLine()`:

```
StringBuilder bldr = new StringBuilder();
bldr.Append("This is the start");
bldr.AppendFormat("Foo bar {0}", 1);
bldr.AppendLine();

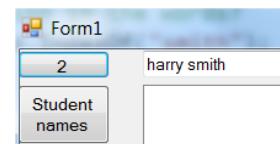
var result = bldr.ToString();
```

The string is only built once, at the `.ToString()`

String (text) exercises

These exercises use the string examples before and also exercises that we have done in earlier lessons. They build up one on another just as you would when coding.

How many words are there in a string? Show the answer on the front of your Q1 button(btn1.text). You will need to make a counter. (int count;)



Create a new button btn2 text = Student names. We are going enter students names.

Check if the name smith is entered into program. If it is then make a MessageBox say "Smith is banned". Hint, try using IndexOf

If Smith is banned then don't let the ListBox have his name, otherwise have the ListBox put the person's name in it.



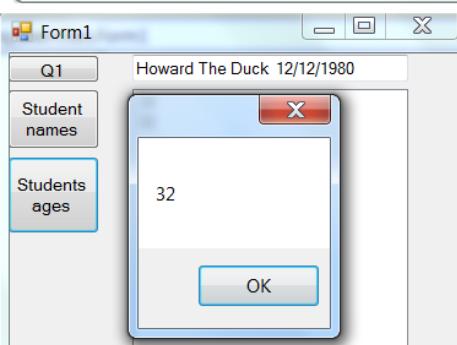
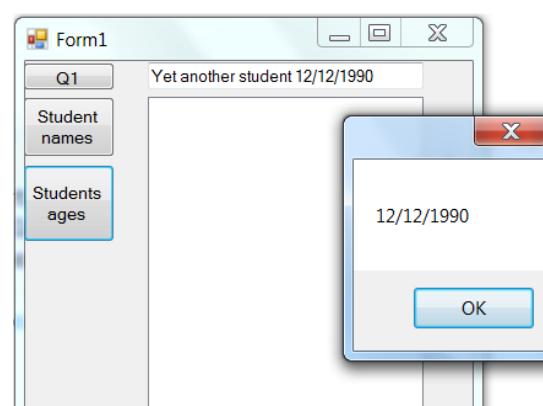
We are going to enter the student name and their birthdate with a new button Btn3 (just to make coding easier) text = Student Ages.

The format will always be Firstname Surname Birthdate. Birthdate will always be formatted 00/00/0000 (Day/Month or Month/Day is not important at this stage (whatever your computer shows will do)).

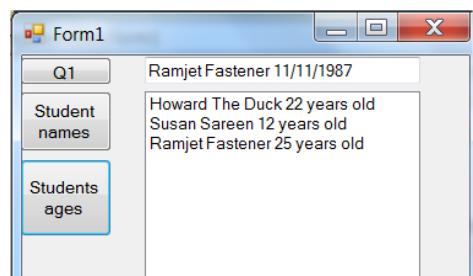
So its "Howard the Duck 12/12/2000".

Using **Substring(start, length)** and **LastIndexOf** and find a way to show in a MessageBox the birthdate of the student. Remember that although the format of the date will always be the same, the length of their name will always change.

Now we want to calculate the age of the person from their birthdate and show it in a MessageBox. Use `DateTime now = DateTime.Today;` to get the current date. And subtract one from the other with `birthdate.Year`



Finally fill the ListBox with the person's name and their age.



```
private void btn1_Click(object sender, EventArgs e)
{
}
```

```
//How many words are in the string?  
string strwords = txtinput.Text;  
char[] space = { ' ' };  
string[] strArr = strwords.Split(space);  
int count = 0;  
lbxoutput.Items.Clear();  
foreach (string s in strArr)  
{  
    count++;  
}  
btn1.Text = Convert.ToString(count);  
  
}  
  
private void btn2_Click(object sender, EventArgs e)  
{  
    string strwords = txtinput.Text;  
    int findsmith;  
        //is smith in the words?  
    findsmit = strwords.IndexOf("smith");  
    if (findsmith >0)  
    {  
        MessageBox.Show("Smith is Banned");  
    }  
    else  
    {  
        lbxoutput.Items.Add(strwords);  
    }  
  
}  
  
private void btn3_Click(object sender, EventArgs e)  
{  
    string strwords = txtinput.Text;  
    string date;  
    int wordlength = strwords.LastIndexOf(" ");  
    int endword = strwords.Length - wordlength;  
    date = strwords.Substring(wordlength, endword);  
  
    DateTime birthdate = Convert.ToDateTime(date);  
    DateTime now = DateTime.Today;  
    int age = now.Year - birthdate.Year;  
  
    string nameonly = strwords.Substring(0, wordlength);  
    lbxoutput.Items.Add(nameonly + " " + age + " years old");  
}
```

String Exercises - Do this

Write a program that finds how many times a substring is contained in a given text (perform case insensitive search). Example:

The target substring is "in".

The text is as follows: We are living in an yellow submarine. We don't have anything else. Inside the submarine is very tight. So we are drinking all the day. We will move out of it in 5 days.

The result is: 9.

You are given a text. Write a program that changes the text in all regions surrounded by the tags <uppercase> and </uppercase> to uppercase. The tags cannot be nested.

Example: We are living in a yellow submarine. We don't have anything else.

The expected result: We are living in a YELLOW SUBMARINE. We don't have ANYTHING else.

Write a program that extracts from a given text all sentences containing given word. Consider that the sentences are separated by "." and the words – by non-letter symbols.

Example: The word is "in".

The text is: We are living in a yellow submarine. We don't have anything else. Inside the submarine is very tight. So we are drinking all the day. We will move out of it in 5 days.

The expected result is: We are living in a yellow submarine. We will move out of it in 5 days.

We are given a string containing a list of forbidden words and a text containing some of these words. **Write a program that replaces the forbidden words with asterisks.**

Example: Microsoft announced its next generation PHP compiler today. It is based on .NET Framework 4 and is implemented as a dynamic language in CLR.

Words: "PHP, CLR, Microsoft"

The expected result: ***** announced its next generation *** compiler today. It is based on .NET Framework 4.0 and is implemented as a dynamic language in ***.

These next two need For loops , Foreach, and Arrays.

Set up an **array** to hold the following string values:

Beyonce (f)
David Bowie (m)
Elvis Costello (m)
Madonna (f)
Elton John (m)
Charles Aznavour (m)

Write a program to count how many are male vocalists and how many are female. Display your answer in a textbox, ListBox or console.

Set up an array of the following musical instruments:

cello
guitar
violin
double bass

Loop and remove the vowels. Display the new words in a ListBox

15. Generic Collections

Watch this video series for the next part of the course, it's awesome.

<http://pluralsight.com/training/Courses/TableOfContents/csharp-collections>

Arrays do not resize dynamically. The List type in the C# language does that's the core difference. List is the most widely used of all collection classes

This means a list is *mutable*. With List, you do not need to manage the size on your own, you can add and delete from your list on the fly. List gives the same high performance Index based actions as an Array as well as the ability to add and remove element.

Lists contain **generic type**, which means almost the opposite of what you think.

Only one type of object can be held in a list, be it string, int bool etc., this prevents errors occurring when processing. When you declare a list the T (List<T>) refers to the Type you use (List<String>).

You can also access the elements in a list by their **position**, so in a list of fruit below you can get the **apple** by going **Fruit[1]** because the list starts at 0.

List<T> Exercise

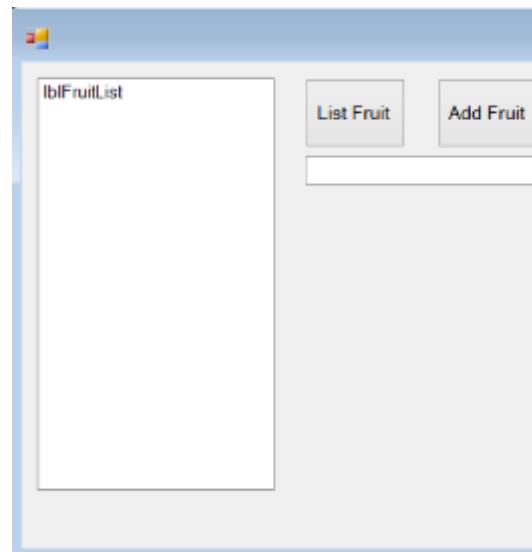
Create the following form with a ListBox, 2 buttons and a text box. Give them appropriate names.

The most common way to iterate through a list is with a ForEach, however you can use For loop and count through each index number, this allows you to put in limits and even count down.

You can instantiate your list with

```
Var FruitList = new List<string>();  
Instead of List<string> FruitList = new List<string>();
```

ReSharper likes var, as it makes the code smaller and easier to read. Since you tell the program what the type is on the right side, you are only repeating it on the left.



```
private void btnlistFruit_Click(object sender, EventArgs e)  
{  
    List<string> FruitList = new List<string>();  
  
    FruitList.Add("orange");  
    FruitList.Add("apple");  
    FruitList.Add("banana");  
  
    FruitList.Sort(); //sort Alphabetically  
    btnListFruit.Text = Convert.ToString(FruitList.Count); //count how many fruit  
    and show on the button
```

```
//This loops through each of the fruit and shows them in the list. It  
is the most common way to iterate through a list.  
foreach (string fruit in FruitList) // Loop through List with foreach  
{  
    lblFruitList.Items.Add("For each " + fruit);  
}  
//this also loops through each of the fruit by counting up the index of each  
one.  
for (int i = 0; i < FruitList.Count; i++) // Loop through List with for  
{  
    lblFruitList.Items.Add("For " + FruitList[i]);  
}  
}
```

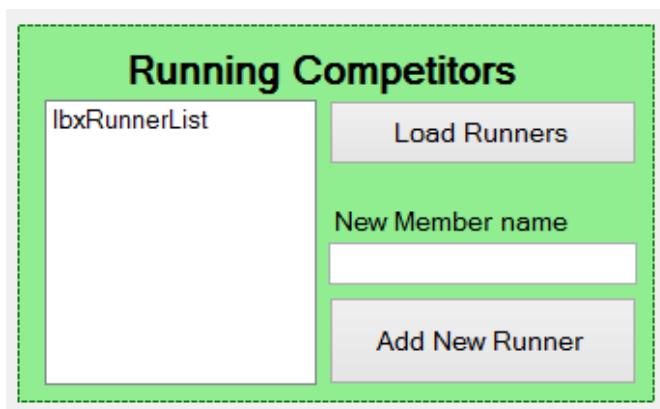
Using a SortedList Exercise (Key, Value)

The **SortedList** class introduces a new feature that brings it close to our ultimate destination of Dictionary. That feature is that it holds two values, one is the **KEY** and the other is the **VALUE**. It's very quick to look up values as the list is already sorted.

The **key** field is unique, a numbered sequence, unique words, etc. The value filled can be any data you want, of course it's restricted as its Generic, so it's either all String, or all Int etc.

This simple exercise will give an overview of the **SortedList** followed by a project

Create a new project with a **ListBox**, 2 buttons and a **Text box** to enter the runners names.



Create a Global variable named **Runners List**, it has to be global as it runs across a number of methods (of course you could make it local and pass it through as a parameter)

```
private SortedList<int, string> RunnerList = new SortedList<int, string>();
```

Under the Load Runners button click event add the 4 runners to the **SortedList**.

`loadLBxRunners();` is a method we make next to load the data into the listbox

```
private void btnLoadRunners_Click(object sender, EventArgs e)
{
    // this is a Key Value pair
    RunnerList.Add(1, "Howard");
    RunnerList.Add(2, "Jones");
    RunnerList.Add(3, "James");
    RunnerList.Add(4, "Speedy");
    //run the method that loads the list into the listbox
    loadLBxRunners();
}
```

The **ForEach** in the method here will generate automatically, just select the options you want and give it a name **Runner**.

```

foreach (var VARIABLE in COLLECTION)
{
}

foreach (var VARIABLE in RunnerList)
{
    KeyValuePair<int, string> Struct System.Collections.Generic.KeyValuePair<TKey, TValue>
    var
        Defines a key/value pair that can be set or retrieved.
}

```

Often programmers just use the **pair** text it generates. So you get **pair.key** and **pair.value**.

```

foreach (KeyValuePair<int, string> Runner in RunnerList)
{
    No suggestions
}

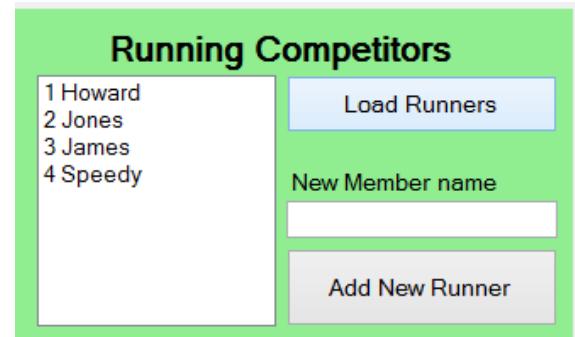
```

Runner now holds the **Key** and **Value** for each entry in the SortedList, you just choose which you need to use, in this case both.

```

foreach (KeyValuePair<
{
    Runner.|
}
//reload
foreach (K
    1bxRunn
        Key
        ToString
        Value

```



```

private void loadLBxRunners()
{
    //clear the listbox
    1bxRunnerList.Items.Clear();

    //reload the listbox by adding in each entry in the sortedlist
    foreach (KeyValuePair<int, string> Runner in RunnerList)
    {
        1bxRunnerList.Items.Add(Runner.Key + " " + Runner.Value);
    }
}

```

The Add new Runner Button simply adds a new runner to the list, and refreshes the ListBox. However the problem is that you can't add a new person that repeats a key as keys must be unique. So you ned to get the last key used and then add 1 to it.

Things get complicated when you delete out runners later and then add a new one in.



```
private void btnAddRunners_Click(object sender, EventArgs e)
{
    //add a new member
    int value = 0;
    //loop through each entry, the last one is the max of the loop
    foreach (var pair in RunnerList)
    {
        value = pair.Key; //just keep the last entry added
    }

    //loop until there is a spare number. if Runnerlist contains the
    value then add 1 to it and try again ==true is redundant in the code
    //We could loop from 0 with a For loop looking for an empty value as well.
    while (RunnerList.ContainsKey(value))
    {
        //add 1 to value
        value += 1;
    }
    //add the new entry to the list.
    RunnerList.Add(value, txtRunnerName.Text);
    //empty the text box and reset the focus to it.
    txtRunnerName.Text = "";
    txtRunnerName.Focus();

    loadLBxRunners();
}
```

Delete out Runners.

Instead of using a delete button we can use the `SelectedIndexChanged` and delete the selected item by clicking on it. That gives us back a string with the value of "1 Howard".

Double click on the `ListBox` to generate the method.

We only want the Key, (1) so need to split the string at the space using the `split` function get the number and remove it from the `RunnersList` at that key. Then reload the `ListBox`.

```
private void lbRunnerList_SelectedIndexChanged(object sender, EventArgs e)
{
    //get the item in the listbox clicked on
    string SelectedRunner = lbxRunnerList.SelectedItem.ToString();
    string[] data = SelectedRunner.Split(Convert.ToChar(" "));

    RunnerList.Remove(Convert.ToInt32(data[0]));
    loadLBxRunners();
}
```



Now what happens when you add a new Runner?

Use a For loop, or something, to replace the deleted entry with a new entry at that number.

Generics - Dictionaries

A dictionary or a hash table is a collection in which each element is a **key/value** pair.

Dictionaries are most commonly used for lookups and sorted lists. The Dictionary class is one of the most commonly used collections (along with the List<T> collection). It uses a Hashtable data structure to store keys and values, and it is fast and efficient.

A dictionary is really fast for working with arrays. Normally if you want to find an element in an array you have to loop all through the array, from beginning to end, with a big array this can take time.

However a Hashtable / Dictionary will go directly to the item, regardless of the size of the array or any other elements in it.

There are a whole bunch of Dictionary types, each has a slightly different purpose, and some are faster than others.

As you can see from the table below both Dictionary and Hashtable have the same underlying structure, with Dictionary slightly faster.

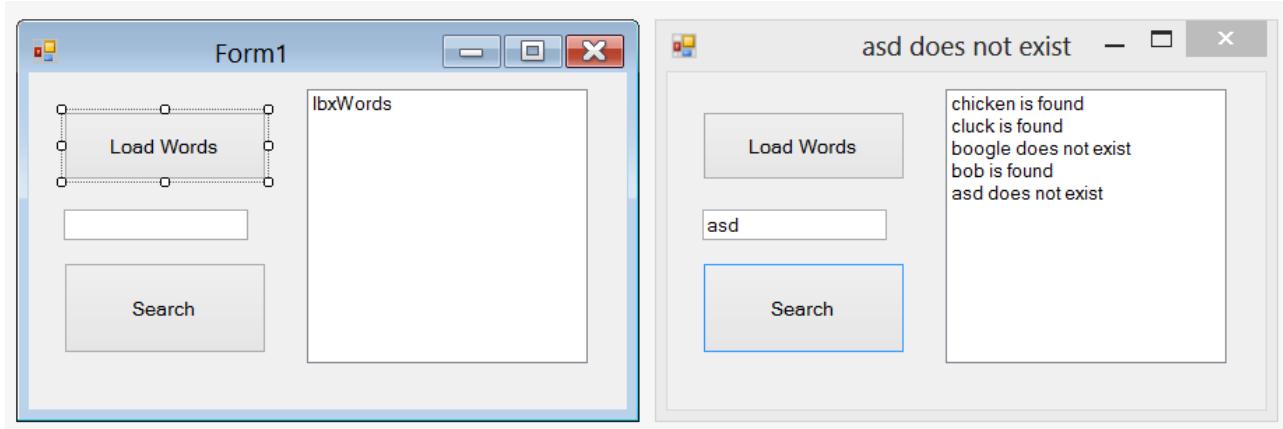
| Type | Internal structure | Retrieve by index? | Memory overhead (avg. bytes per item) | Speed: random insertion | Speed: sequential insertion | Speed: retrieval by key |
|------------------------|--------------------|--------------------|---------------------------------------|-------------------------|-----------------------------|-------------------------|
| Unsorted | | | | | | |
| Dictionary <K,V> | Hashtable | No | 22 | 30 | 30 | 20 |
| Hashtable | Hashtable | No | 38 | 50 | 50 | 30 |
| ListDictionary | Linked list | No | 36 | 50,000 | 50,000 | 50,000 |
| OrderedDictionary | Hashtable + array | Yes | 59 | 70 | 70 | 40 |
| Sorted | | | | | | |
| SortedDictionary <K,V> | Red/black tree | No | 20 | 130 | 100 | 120 |
| SortedList <K,V> | 2xArray | Yes | 2 | 3,300 | 30 | 40 |
| SortedList | 2xArray | Yes | 27 | 4,500 | 100 | 180 |

Dictionaries are fun to use and really practical. Here are some popular methods

```
DictionaryTest.Add("name", "value");
DictionaryTest.Clear();
DictionaryTest.ContainsKey("name");
DictionaryTest.ContainsValue("value"); // not recommended, very slow
DictionaryTest.TryGetValue("name"); // not recommended, very slow
```

Use a real Dictionary in a Dictionary

Dictionaries are great because they are fast. In this case we will see how fast it is, by loading 170,000 words into a dictionary and then searching it for known words.



In the example below we made a dictionary **DictionaryWords** and load the words from the text file into it.

Get the **dict.txt** file and load it into your C drive, `@"c:\\dict1.txt"` or in the bin folder next to your program if you don't want to specify the path. The `@` symbol treats everything between the `" "` as text and doesn't operate on it.

```
public partial class Form1 : Form
{
    Dictionary<string, string> DictionaryWords = new Dictionary<string, string>();
    public Form1()
    {
        InitializeComponent();
    }

    private void btnLoadWords_Click(object sender, EventArgs e)
    {
        //make a counter that shows how many words are loaded (really just something to look at)
        int count = 0;
        //a great way of loading each line of a file into your program
        foreach (string line in File.ReadLines(@"c:\\dict1.txt"))
        {
            //create the counter, show it on the top of the form
            count++;
            this.Text = count.ToString();
            //if the dictionary doesn't have the word then load it. Otherwise it crashes when it finds a conflict with an existing word
            if (DictionaryWords.ContainsKey(line) == false)
            {
                //add key / value with both the same word
                DictionaryWords.Add(line, line);
            }
        }
    }
}
```

```
        }
    } //shows when the words are loaded
    this.Text = "Words Loaded";
}

private void btnSearch_Click(object sender, EventArgs e)
{
    //input the search word
    string SearchWord = txtInputWord.Text;
    //if the word is in the dictionary
    if (DictionaryWords.ContainsKey(SearchWord))
    {
        this.Text = SearchWord + " is found";
        lbxWords.Items.Add(SearchWord + " is found");
    }

    else
    { //the word isn't in the dictionary
        this.Text = SearchWord + " does not exist";
        lbxWords.Items.Add(SearchWord + " does not exist");
    }
}
}
```

Can You?

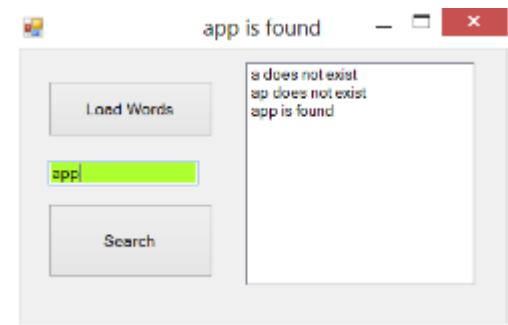
Create a way that the text box background is red when there is no word and green when there is one.

Can you get it to give suggestions as you type the word in? You might want to change the types to char.

Create a way to add words to your dictionary. (This may require writing back to the file, or just in the current words)

What **real** words are in this list of 2,151,220 unique ASCII passwords?

<https://dazzlepod.com/lulzsec/final/>



Oxford English Dictionary

Look in the Goodies, or google for the Oxford English Dictionary text file



OxfordDicWithDesc.txt

It contains an entire dictionary. Load it into a Dictionary with Key / Value pairs, and then create a program so you can type in a word and get the definition. We will use this next term as an android app as well.

How?

Take each line

```
Abandon -v. 1 give up. 2 forsake, desert. 3 (often foll. By to; often refl.) Yield  
to a passion, another's control, etc. -n. Freedom from inhibitions. ↗ abandonment  
n. [french: related to *ad-, *ban]
```

Find the first space in the string which will occur after the first word.

Pass the word before the space to the Key and what's after the space to the Value.

Use the .split() command – read examples here <https://www.dotnetperls.com/split>

```
Abandon -v. 1 give up. 2 forsake, desert. 3 (often foll. By to; often refl.) Yield  
to a passion, another's control, etc. -n. Freedom from inhibitions. ↗ abandonment  
n. [french: related to *ad-, *ban]
```

Abandoned adj. 1 deserted, forsaken. 2 unrestrained, profligate.

```
Abase v. (-sing) (also refl.) Humiliate, degrade. ↗ abasement n. [french: related  
to *ad-, *base2]
```

Abashed predic. Adj. Embarrassed, disconcerted. [french es- *ex-1, bair astound]

```
Abate v. (-ting) make or become less strong etc.; diminish. ↗ abatement n. [french  
abatre from latin batt(u)o beat]
```

Abattoir n. Slaughterhouse. [french abatre fell, as *abate]

Count how many letters are in a sentence

Let's work on this earlier problem with a Dictionary. Can we count how many letters are in a sentence?

```
private void button2_Click(object sender, EventArgs e)
{
    var Letters = new Dictionary<char, int>();
    string word = "if word is the word and bird is the word
then word is the bird";
    // for each letter in the word
    foreach (char a in word)
    {
        //if there is no entry in the dictionary then add a new entry
        //with 1 as the counter
        if (Letters.ContainsKey(a) == false)
        {
            Letters.Add(a, 1);
        }
        else // if its already in the dictionary increase the number
        (value) by one.
        {
            //finds the key and adds 1 to the value, basically counts up
            Letters[a]++;
        }
    }
    //loop through the dictionary and get out the key/value pairs

    foreach (KeyValuePair<char, int> pair in Letters)
    {
        listBox1.Items.Add(pair.Key + " --> " + pair.Value);
    }
}
```

button2

| | |
|---|-------|
| i | -> 6 |
| f | -> 1 |
| - | -> 14 |
| w | -> 4 |
| o | -> 4 |
| r | -> 6 |
| d | -> 7 |
| s | -> 3 |
| t | -> 4 |
| h | -> 4 |
| e | -> 4 |
| a | -> 1 |
| n | -> 2 |
| b | -> 2 |

button2

| | |
|-------|------|
| -> 14 | |
| a | -> 1 |
| b | -> 2 |
| d | -> 7 |
| e | -> 4 |
| f | -> 1 |
| h | -> 4 |
| i | -> 6 |
| n | -> 2 |
| o | -> 4 |
| r | -> 6 |
| s | -> 3 |
| t | -> 4 |
| w | -> 4 |

Change the Dictionary to a Sorted Dictionary to see it sorted →

```
var Letters = new SortedDictionary<char, int>();
```

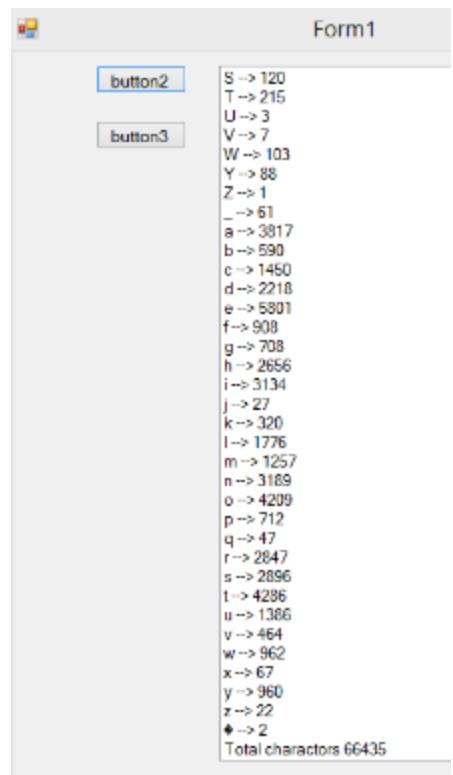
Import a REAL file such as Rodger Zelazney's book in the goodies folder Zelazney.txt.

Add it to your Bin folder and rename it **Z.txt** (easy to debug then)

```
string word = File.ReadAllText("Z.txt");
```

Once you have it working, get it to count how many characters are in the book.

What surprised me was how FAST the program ran.



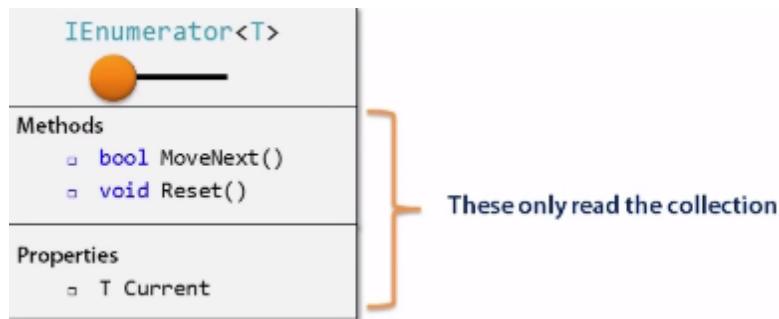
Can you?

Make it count all the words in the book and return the top ones?

Having got the program working can you now [chart the data?](#)

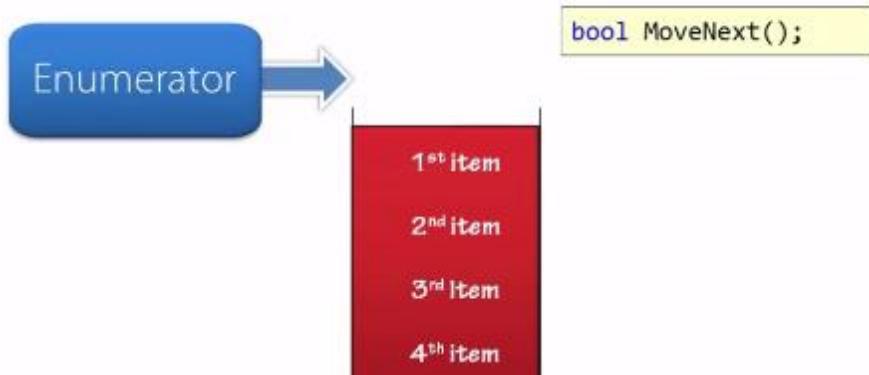
Enumerators – under the hood theory

Underneath every ForEach and arrays is the part that does the counting through each element, This is called the Enumerator. You can use it in your code by itself , although its limited, but even if you just use the items that derive from it it's well worth knowing about it and understanding what it is. This comes from the **C# Collections Fundamentals** video



In your collection the enumerator is like a cursor. Its starts by not pointing to any element in the collection.

You then call **MoveNext()**; and it moves you to the next element in the collection.



Then you call a property **T Current** which returns that item.



Then you call **MoveNext()**; again and **T Current** for the next item in the collection. Rinse and Repeat till the end.

At the end **MoveNext()**; will return **False** to show there are no more items in the collection.

Note that you can only move to the next item and retrieve the item, you can't look ahead to another item, not count items, nor go back!

Therefore you can use it when streaming in data from an outside source because it doesn't need any more information. It also works on Strings, which is just a list of Char.

You will find enumerators almost everywhere. Anywhere that you can use a ForEach loop is using `IEnumerable<T>`.

foreach Loops

When the compiler sees this...

```
foreach (T item in collection)
{
    // do something
}
```

ForEach on a Collection<T>
gives this underlying code

...it replaces it with something equivalent to this...

```
using (IEnumerator<T> enumerator = collection.GetEnumerator())
{
    bool moreItems = enumerator.MoveNext();
    while (moreItems)
    {
        T item = enumerator.Current;
        // do something
        moreItems = enumerator.MoveNext();
    }
}
```

This works for all collections **except Arrays**. When it detects an Array it converts the ForEach into a For loop, because it's a much more efficient way of iterating through arrays.

```
foreach (T item in collection)
{
    // etc.
}
```

However ForEach on
an Array gives a For
Loop code

If this is an array type
 – the compiler will use a **for** loop
 – NOT an Enumerator

You can't change or modify the collection while enumerating a collection unless it's an Array

So if you have a ForEach loop and inside it you try to change the collection it will not accept it.

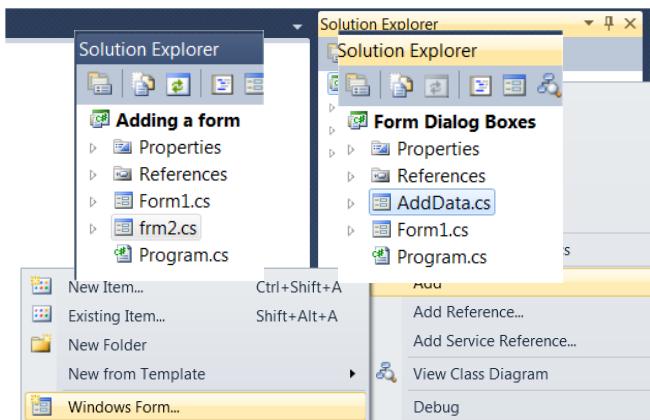
However, if you use an Array, the ForEach compiles as a For Loop and you CAN change it as you iterate through it.

16. Multiple forms and passing data using a dialog box

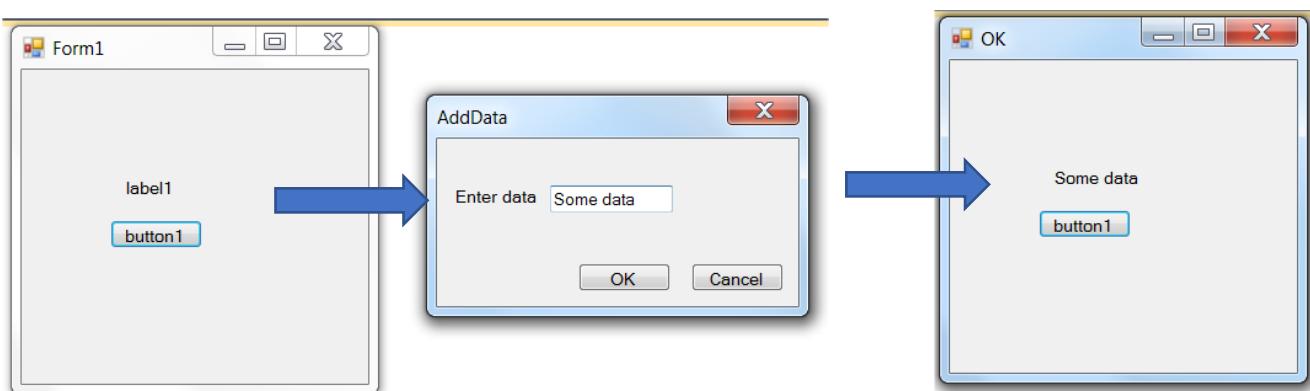
We are going to create a Dialog form to add data to your main form on. We are going to create two forms, one is a dialog box that will return back the data entered into it.

To add another form to a project go **Add New Item** and then Choose **Windows Form**.

Then rename the new form to **AddData**

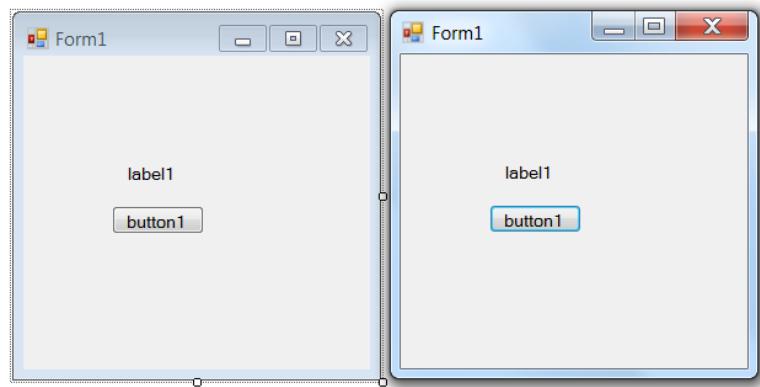


In the Example below clicking on Button1 opens the Adddata form then you enter the data into the text box then click OK.



The main form

On your main form add a button and a label.



Create the code below after you have made the Adddata form. Otherwise the code won't work.

Comment out this line to stop having to click the OK button twice ...

```
if (adddataform.ShowDialog() == DialogResult.OK)
```

Main Form Code

```
public partial class FrmMain : Form
{
    public FrmMain()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        //Create an instance of the adddata form in your code
        AddData adddataform = new AddData();
        //Show the form as a dialog
        adddataform.ShowDialog();

        // if you have clicked on the OK button .... Then
        if (adddataform.ShowDialog() == DialogResult.OK)
            {//do something when they click OK once back on this form
                string data = string.Format(adddataform.txtdata.Text);
                lbldata.Text = data;
            }
        //Check with the form title that it worked
        this.Text = "OK";
    }
}
```

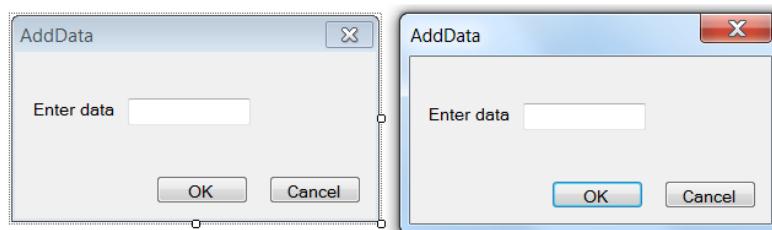
Things to add in

```
//copy across backcolor
    this.BackColor = adddataform.BackColor;
```

Set your textboxes to public by changing their **Modifiers** to Public.

Add data Form

Set up your data form properties as shown right, this creates a dialog form. Name it AddData.



```
public partial class AddData : Form
{
    public AddData()
    {
        InitializeComponent();
    }

    private void btnOK_Click(object sender, EventArgs e)
    {
        this.Close();
    }
}
```

| txtform1 System.Windows.Forms.TextBox | |
|---------------------------------------|----------------|
| GenerateMember | True |
| HideSelection | True |
| ImeMode | NoControl |
| Lines | String[] Array |
| Location | 58, 62 |
| Locked | False |
| Margin | 3, 3, 3, 3 |
| MaximumSize | 0, 0 |
| MaxLength | 32767 |
| MinimumSize | 0, 0 |
| Modifiers | Public |

| Properties | |
|-----------------------------------|---|
| AddData System.Windows.Forms.Form | |
| Font | Microsoft Sans Serif, <input checked="" type="button"/> |
| ForeColor | ControlText <input checked="" type="button"/> |
| FormBorderStyle | FixedDialog <input checked="" type="button"/> |
| HelpButton | False |
| Icon | <input checked="" type="button"/> (Icon) |
| ImeMode | NoControl |
| IsMdiContainer | False |
| KeyPreview | False |
| Language | (Default) |
| Localizable | False |
| Location | 0, 0 |
| Locked | False |
| MainMenuStrip | (none) |
| MaximizeBox | <input checked="" type="button"/> False |
| MaximumSize | 0, 0 |
| MinimizeBox | <input checked="" type="button"/> False |
| MinimumSize | 0, 0 |
| Opacity | 100% |
| Padding | 0, 0, 0, 0 |
| RightToLeft | No |
| RightToLeftLayout | False |
| ShowIcon | True |
| ShowInTaskbar | <input checked="" type="button"/> False |
| Size | 300, 169 |
| SizeGripStyle | Auto |
| StartPosition | CenterParent |
| Tag | AddData |
| Text | <input checked="" type="button"/> AddData |
| TopMost | False |

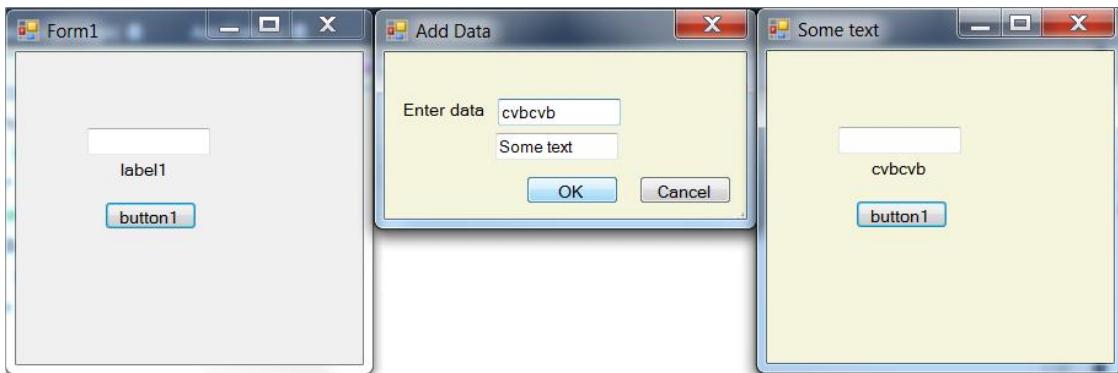
On the OK button set the Properties of the DialogResult to OK.

| Properties | |
|-----------------------------------|--------------------------------------|
| btnOK System.Windows.Forms.Button | |
| CausesValidation | True |
| ContextMenuStrip | (none) |
| Cursor | Default |
| DialogResult | OK <input checked="" type="button"/> |
| Dock | None |
| Enabled | True |

Passing data with a Public Method

On the second form create a public method and a text box txtbox1. Put some text in the text field of it. I want my public method to return what I enter into the Textbox. My method looks something like

```
public string getTextBoxValue()
{
    return TextBox.Text;
}
```



Or a better and more C# friendly way

```
public string getTextBoxValue
{
    get { return txtbox2.Text; }
}
```

Now we want to set the title bar of Form1 to whatever we enter in the Textbox on Form2.
Type this code below the 3 lines above.

```
this.Text = adddataform.getTextBoxValue;
```

Your code for the event will now look something like this

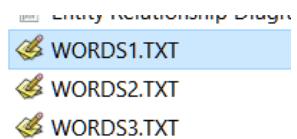
```
private void button1_Click(object sender, EventArgs e)
{
    //instantiate a new version
    AddData adddataform = new AddData();
    adddataform.ShowDialog();
    //copy across backcolor
    this.BackColor = adddataform.BackColor;
    //new text from second form
    this.Text = adddataform.getTextBoxValue;
    string data = string.Format(adddataform.txtdata.Text);
    lbldata.Text = data;
}
```

And there you have it. You can now pass data between 2 forms.

17. Opening files and saving with File

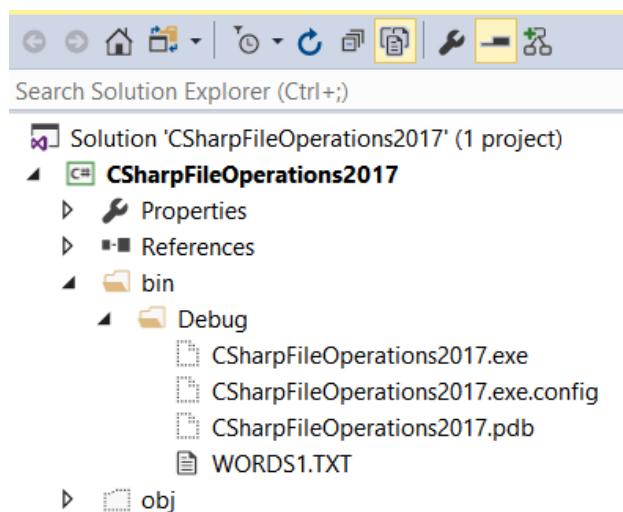
First copy files to your Debug Folder.

Turn on **Show All Files**



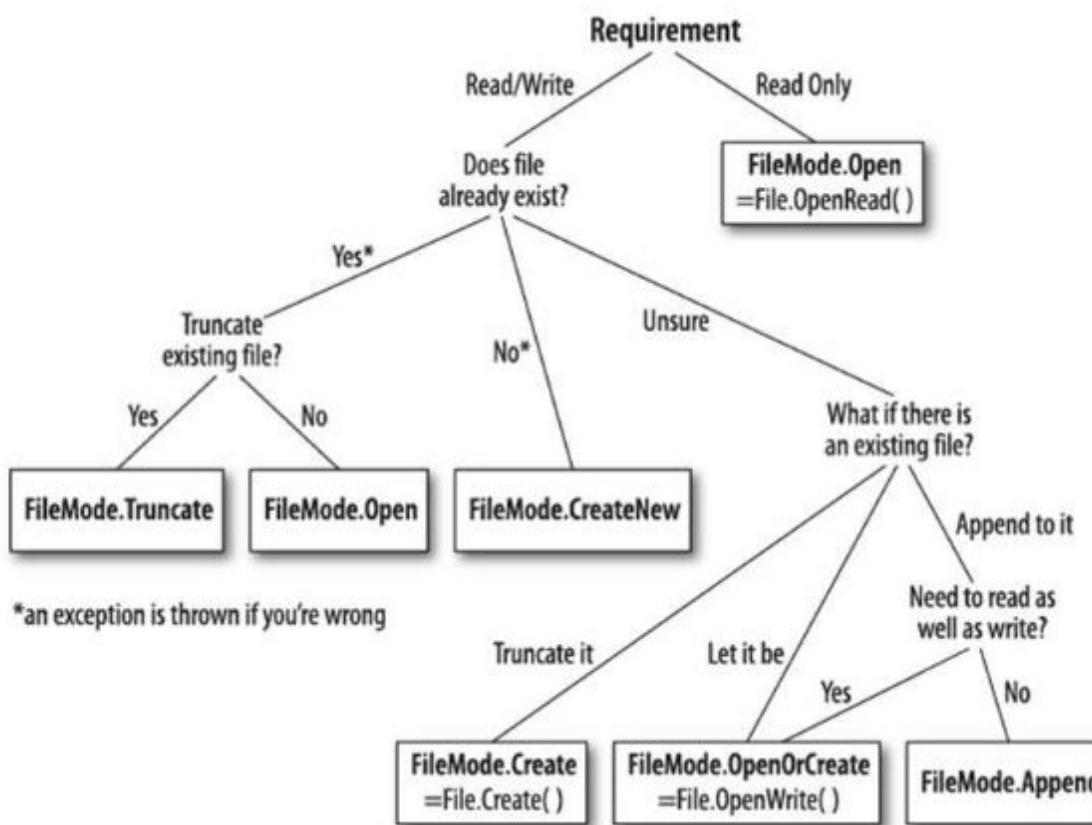
Find a Words text files in the Goodies folder,

Drag it into the Debug folder



Choosing a File Mode.

From <http://apprize.info/c/nutshell/15.html>



Here are some simple and quick tools that you can use to open and read files into your program using File.

The @ symbol at the beginning of a path is a Verbatim String Character

I had trouble finding an explanation of the @ in paths such as `string path1 = @"c:\temp\MyTest.txt";` It is, in fact, a **Verbatim String Character**

A verbatim string literal consists of an @ character followed by a double-quote character, zero or more characters, and a closing double-quote character.

A simple example is `@"hello"`. In a verbatim string literal, the characters between the delimiters are interpreted verbatim, the only exception being a quote-escape-sequence.

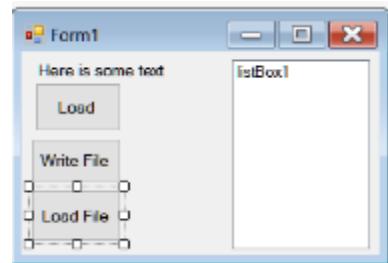
In particular, simple escape sequences and hexadecimal and Unicode escape sequences are not processed in verbatim string literals. A verbatim string literal may span multiple lines.

File.ReadLines – read in line by line

File.ReadLines reads lines as you process them. It reads all the lines in from a file in a ForEach loop, but only as needed. With the File.ReadLines method, you can read in another line before each iteration of your ForEach loop.

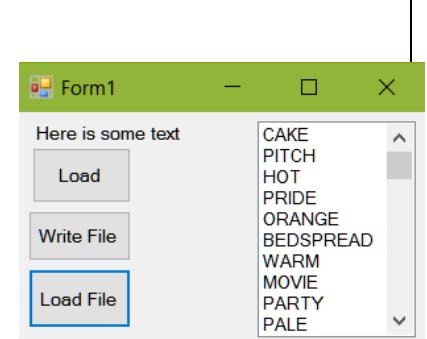
Be sure that you have **System.IO.File**

```
// Read in lines from file.
foreach (string line in File.ReadLines(@"words1.txt"))
{
    listBox1.Items.Add(line);
}
```



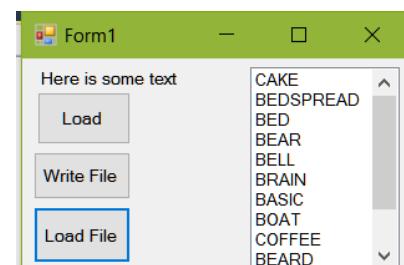
Wrap it in a check in case the file is missing

```
private void btnLoadFile_Click(object sender, EventArgs e)
{
    //If the file exists then load it
    if (File.Exists("words1.txt"))
    {
        // Read in lines from file.
        foreach (string line in File.ReadLines(@"words1.txt"))
        {
            listBox1.Items.Add(line);
        }
    }
    else
    {
        MessageBox.Show("There is no file");
    }
}
```



Here is a good use for it, looking at each line for key words.

```
if (line.StartsWith("C") || line.StartsWith("B"))
{
    listBox1.Items.Add(line);
}
```



File.ReadAllLines – read all lines in at once

This reads the data in line by line to an **array** and allows you to operate on each line with a **ForEach**.

ReadAllLines reads the entire file in at once to the array while **File.ReadLines** reads it in just line by line.

```
private void btnLoadAllLines_Click(object sender, EventArgs e)
{
    string[] lines = File.ReadAllLines("words1.txt");
    //do something with the array
}
```

| Autos | |
|-------|---------------|
| Name | Value |
| lines | {string[100]} |
| [0] | "CAKE" |
| [1] | "PITCH" |
| [2] | "HOT" |
| [3] | "PRIDE" |
| [4] | "ORANGE" |
| [5] | "BEDSPREAD" |
| [6] | "WARM" |
| [7] | "MOVIE" |
| [8] | "PARTY" |

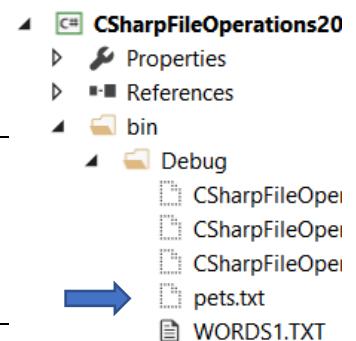
When you read in a file with **File.ReadAllLines**, many strings are allocated and put into an array in a single method call.

With **StreamReader**, you can allocate each string as you pass over the file by calling **ReadLine**. This makes **StreamReader** more efficient unless you need all the file data in memory at once.

File.WriteAllLines – save an array

This easily writes from an array to a file. Very handy

```
private void btnWriteFile_Click(object sender, EventArgs e)
{
    string[] pets = { "cat", "dog", "Walrus" };
    File.WriteAllLines("pets.txt", pets);
}
```

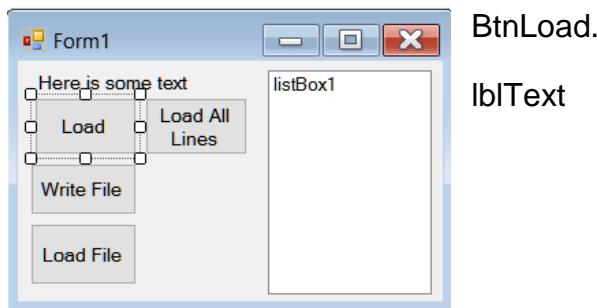
*File.WriteAllText – save all text*

This quick and simple code writes text to a text file, the second parameter can also hold a string variable.

```
File.WriteAllText("pets.txt", "Budgies rule all!");
```

Read and Write from a Resource File

Magic numbers and text are not liked in programming, if you can create a resource file and store all the text and numbers in that. Then you can change the language, or edit the text easily.



BtnLoad.

lblText

In a new file, create a Resource file, and add in the following.

The text in the resource folder is only Read Only.

| | | Name | Value | Comment |
|---|--|----------|--------------------------|--------------|
| | | lblText | Loaded From the Resource | Default text |
| | | lbltext2 | Another message Loaded | #2 |
| ▶ | | lbltext3 | Final message loaded | #3 |

Add the code.

```
public partial class Form1 : Form
{
    //Make a counter
    private int count = 1;
    public Form1()
    {
        InitializeComponent();
    }

    private void btnLoad_Click(object sender, EventArgs e)
    {//loop through counter with switch
        switch (count)
        {
            case 1:
                lblText.Text = Resource1.lblText;
                break;
            case 2:
                lblText.Text = Resource1.lbltext2;
                break;
            case 3:
                lblText.Text = Resource1.lbltext3;
                count = 1;
                break;
        }
    }
}
```

```
    }
    count++;
}
}
```

Address Book – 2D Array with file loading

This Program inputs comma delimited data from a text file, and passes it to a multidimensional array. (change it to a Dictionary at the end as a further exercise).

The array is 20 rows, and 3 elements so only holds 20 people

```
string[,] people = new string[20, 3];
```

| | people | (string[20, 3]) |
|--------|--------|---------------------|
| [0, 0] | | "Howard The Duck" |
| [0, 1] | | "565678" |
| [0, 2] | | "The Lake Pond" |
| [1, 0] | | "John Smith" |
| [1, 1] | | "8786678" |
| [1, 2] | | "12 Smith Street" |
| [2, 0] | | "Tarzan" |
| [2, 1] | | "6565655" |
| [2, 2] | | "The local hangout" |
| [3, 0] | | "Harvey Wallbanger" |
| [3, 1] | | "454567" |
| [3, 2] | | "A Street" |

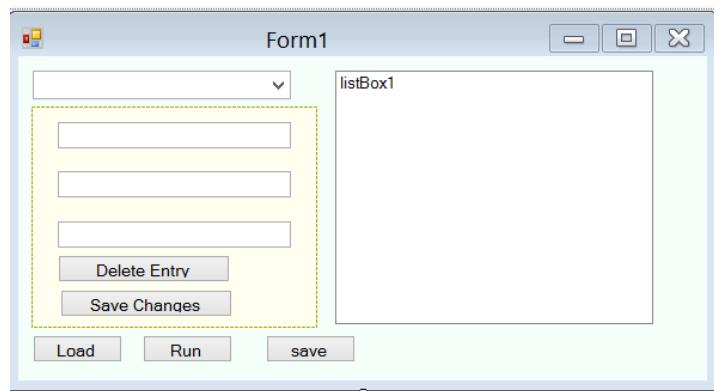
The program uses one of my favourite string functions **Split**, that takes a string, and splits it into chunks by the comma..

```
string[] data = line.Split(',') ;
```

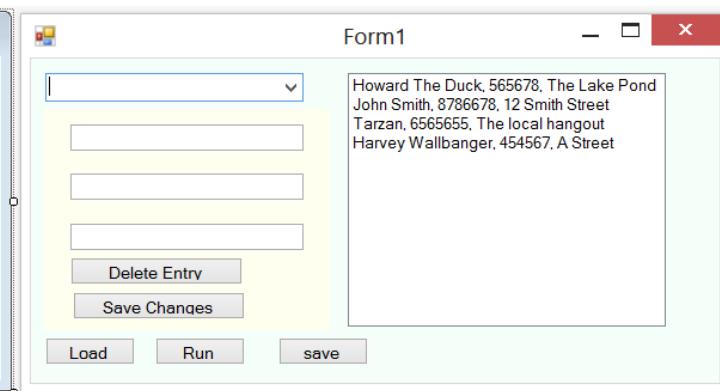
You can see it working on the first line below taking the entry and chopping it up then passing it to a local array named data.

| | | |
|---|------|--|
| ▲ | data | {string[3]} |
| | [0] | "Howard The Duck" |
| | [1] | " 565678" |
| | [2] | " The Lake Pond" |
| | line | "Howard The Duck, 565678, The Lake Pond" |

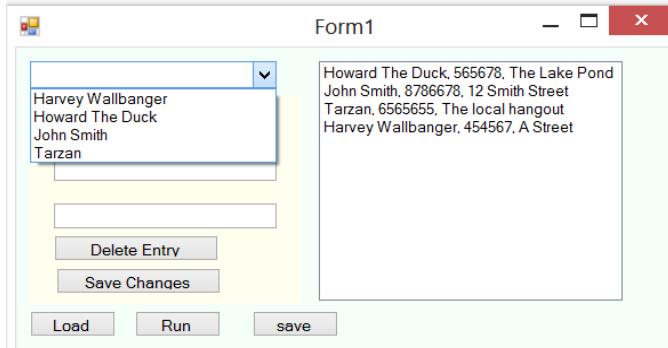
Design



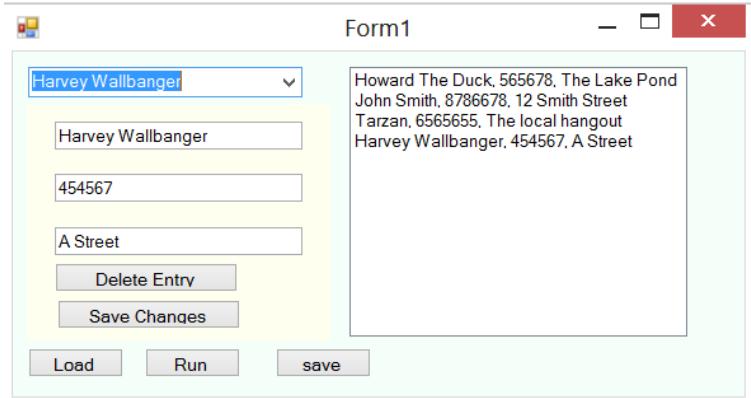
Load the Array to the ListBox



Load the Array to the ComboBox



ComboBox fills the Text boxes



Create a text file with 4 or more comma delimited names, addresses and phone numbers and save it into your debug folder of your application.

C# lessons 2013 for manual > C# Address Book - 2013 > addressbook > addressbook > bin > Debug

| Name | Date modified | Type | Size |
|---------------------------------|-------------------|--------------------|-------|
| addressbook.exe | 15-Aug-13 9:16 AM | Application | 13 KB |
| addressbook.pdb | 15-Aug-13 9:16 AM | Program Debug D... | 46 KB |
| addressbook.txt | 30-Jul-13 8:07 PM | Text Document | 1 KB |
| addressbook.vshost.exe | 15-Aug-13 9:40 AM | Application | 23 KB |
| addressbook.vshost.exe.manifest | 03-Jun-12 2:34 AM | MANIFEST File | 1 KB |

John Smith, 8786678, 12 Smith Street
Howard The Duck, 565678, The Lake Pond
Tarzan, 6565655, The local hangout
Harvey Wallbanger, 454567, A Street

First create an array. The **comboboxrowcount** is used later.

```
public partial class Form1 : Form
{
    //20 rows, 1 column, 3 elements
    string[,] people = new string[20, 3];
    int comboboxrowcount;
```

The code imports the data line by line from the text file, chops it up by the comma and passes the data to the three variables, name, phone and address.

Those three variables pass the data to the three data elements.

This neat split function **line.Split(',')** splits up at the comma and passes it to an array that will automatically work out how many fields it needs to hold.

Trim data[0].Trim(); deletes the white space around each entry.

The **loadarray()**; method below runs automatically when the form loads

```
public Form1()
{
    InitializeComponent();
//load the array from the text file when the program
starts
    LoadArray();
}
```

```
private void LoadArray()
{
    int rowcount = 0;
    foreach (string line in File.ReadLines("addressbook.txt"))
    {
//split the line using the comma and pass it to the string array
        string[] data = line.Split(',');
//pass each data field, name, address, phone to the array
//names[row, element]
        people[rowcount, 0] = data[0].Trim();
        people[rowcount, 1] = data[1].Trim();
        people[rowcount, 2] = data[2].Trim();
//make the counter increase by 1
        rowcount++;
    }
}
```

| | people | {string[20, 3]} |
|----------|--------|---------------------|
| • [0, 0] | [0, 0] | "Howard The Duck" |
| • [0, 1] | [0, 1] | "565678" |
| • [0, 2] | [0, 2] | "The Lake Pond" |
| • [1, 0] | [1, 0] | "John Smith" |
| • [1, 1] | [1, 1] | "8786678" |
| • [1, 2] | [1, 2] | "12 Smith Street" |
| • [2, 0] | [2, 0] | "Tarzan" |
| • [2, 1] | [2, 1] | "6565655" |
| • [2, 2] | [2, 2] | "The local hangout" |
| • [3, 0] | [3, 0] | "Harvey Wallbanger" |
| • [3, 1] | [3, 1] | "454567" |
| • [3, 2] | [3, 2] | "A Street" |

This is the first line of data coming in from the text file. See how the whole entry is entered as Line, then split by the comma and passed to the 3 array elements.

| | | |
|---|------|--|
| ▲ | data | {string[3]} |
| • | [0] | "Howard The Duck" |
| • | [1] | " 565678" |
| • | [2] | " The Lake Pond" |
| • | line | "Howard The Duck, 565678, The Lake Pond" |

Here is the code to load the ListBox from the text file.

```
private void btnload_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();
    //just loads data to listbox to show how simple it is
    foreach (string line in File.ReadLines("addressbook.txt"))
    {
        listBox1.Items.Add(line);
    }
}
```

If you are interested in learning [lambda](#) expressions here is the **For Each** code from above in a single line.

```
File.ReadLines("addressbook.txt").ToList().ForEach(line =>
listBox1.Items.Add(line));
```

BtnRun fills the ComboBox

```
private void FillTheCombobox()
{
    //clear the combobox first
    cbname.Items.Clear();
    int comboboxrowcount = 0;

    //need to stop when there are no more people otherwise it crashes
    while (people[comboboxrowcount, 0] != null)
    {
        //will add just the name
        cbname.Items.Add(people[comboboxrowcount, 0]);
        comboboxrowcount++;
        cbname.Sorted = true;
    }
}
```

This is the change event of the ComboBox

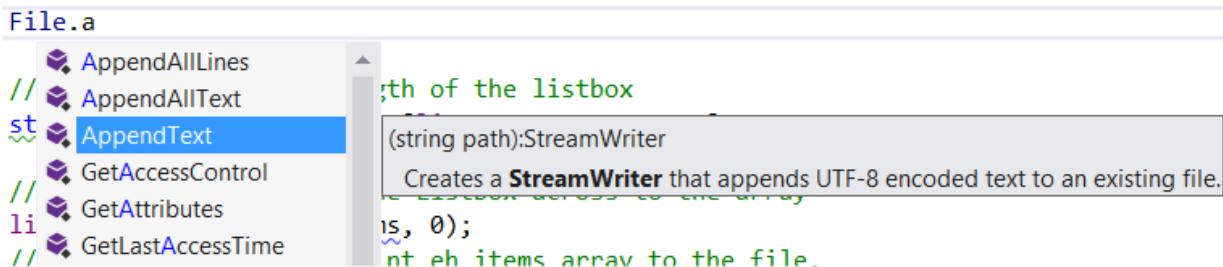
```
private void cbname_SelectedIndexChanged(object sender, EventArgs e)
{
    //copy over the name in the combobox to string name
    string name = Convert.ToString(cbname.SelectedItem);
    comboboxrowcount = 0;

    while (people[comboboxrowcount, 0] != null)
    {
        //loop through the people array looking for the one with the same name
        if (name == Convert.ToString(people[comboboxrowcount, 0]))
        {
            //copy phone and address to other text boxes
            txtname.Text = people[comboboxrowcount, 0];
            txtphone.Text = people[comboboxrowcount, 1];
            txtaddress.Text = people[comboboxrowcount, 2];
        }
        comboboxrowcount++;
    }
}
```

Part 2: Save, Delete, Save changes exercise

Create your own **Save**, **Delete**, and **New** (not shown) Buttons

For **Save** investigate using Appendtext



For **Delete**, you might replace the items in the array with blank lines and then remove them when you write back to the file.

Make sure the ListBox and the ComboBox are **updated** to show the changes.

Change the **Array** into a **Dictionary**. Use the Name as the key, and the Phone and address as the Value. It will make your code much more easier to use and fun to write.

StreamReader and StreamWriter



C# makes use of a thing called a **stream** to allow programs to work with files.

A stream is a link between your program and a data resource. Data can flow up or down your stream, so that streams can be used to read and write to files.

C# has a range of different stream types which you use depending on what you want to do. All of the streams are used in exactly the same way. In fact you are already familiar with how streams are used, since the Console class, which connects a C# program to the user, is implemented as a stream.

The **ReadLine** and **WriteLine** methods are commands you can give any stream that will ask it to read and write data.

We are going to consider two stream types which let programs use files; these are the **StreamWriter** and **StreamReader** types.

When you use the file handing classes you will need to add the following statement at the very top of your program:

```
using System.IO;
```

IO is the input/output namespace that holds the commands for reading and writing to files.

StreamWriter open the stream

You create a stream object just like you would create any other one, by using new. When the stream is created it can be passed the name of the file that is to be opened.

```
StreamWriter writer ;  
writer = new StreamWriter("test.txt");
```

The variable **writer** will be made to refer to the stream that you want to write into. When the new StreamWriter is created the program will open a file called test.txt for output and connect the stream to it. If this process fails for any reason, perhaps your operating system is not able/allowed to write to the file or the name is invalid, then the action will fail with an appropriate exception.

Note however that this code does not have a problem if the file test.txt already exists. All that happens is that a brand new, empty, file is created in place of what was there. This is potentially dangerous. It means that you could use the two statements above to completely destroy the contents of an existing file, which would be bad. Most useful programs ask the

user whether or not an existing file should be overwritten, you will find out later how to do this.

Write to the stream

Once the stream has been created it can be written to by calling the write methods it provides.

```
writer.WriteLine("hello world");
```

The above statement calls the WriteLine method on the stream to make it write the text —hello world into the file test.txt.

Each time you write a line to the file it is added onto the end of the lines that have already been written.

Closing the stream

When your program has finished writing to a stream it is very important that the stream is explicitly closed using the Close method:

```
writer.Close();
```

When the Close method is called the stream will write out any text to the file that is waiting to be written and disconnect the program from the file. Any further attempts to write to the stream will fail with an exception. Once a file has been closed it can then be accessed by other programs on the computer, i.e. once the close has been performed you can use the Notepad program to open test.txt and take a look at what is inside it. Forgetting to close a file is bad for a number of reasons.

StreamReader opening and reading from a file

Reading from a file is very similar to writing, in that the program will create a stream to do the actual work. In this case the stream that is used is a StreamReader.

Note: There are lots of different ways to read data from a file. You use the one that works for you at the time.

Textreader is the base class for StreamReader and you will see it mentioned in the code.

These are the core members of the Textreader/ StreamReader class.

Peek Returns the next available character but does not consume it. A value of -1 indicates you have reached the end of the stream.

Read() Reads the next character from the input stream and advances the character position by one character. (Overrides TextReader.Read().)

Read(Char[], Int32, Int32) Reads a specified maximum of characters from the current stream into a buffer, beginning at the specified index. (Overrides TextReader.Read(Char[], Int32, Int32).)

ReadBlock Reads a maximum of *count* characters from the current stream, and writes the data to buffer, beginning at a specified index. (Inherited from TextReader.)

ReadLine Reads a line of characters from the current stream and returns the data as a string. (Overrides TextReader.ReadLine().)

ReadToEnd Reads the stream from the current position to the end of the stream. (Overrides TextReader.ReadToEnd().)

```
using (StreamReader sr = new StreamReader("words1.txt"))
{
    string line;
    // Read and display lines from the file until the end of the file is reached.
    while ((line = sr.ReadLine()) != null)
    {
        //do stuff with the text such as read it to a text box or split it by comma to an array
        //    string[] parts = line.Split(',');
        listBox1.Items.Add(line);
    }
}
```

The above program connects a stream to the file, reads the line from the file, and then close the stream.

Detecting the End of an Input File

Repeated calls of **ReadLine** will return successive lines of a file. However, if your program reaches the end of the file the **ReadLine** method will return an empty string each time it is called.

Fortunately the StreamReader object provides a property called **EndOfStream** that a program can use to determine when the end of the file has been reached. When the property becomes true the end of the file has been reached.

```
StreamReader reader;
reader = new StreamReader("Test.txt");
while (reader.EndOfStream == false)
{
    string line = reader.ReadLine();
    listbox.items.add(line);
}
reader.Close();
```

The above program will open up the file test.txt and display every line in the file on the console. The while loop will stop the program when the end of the file is reached.

```
using System;
using System.IO;
class Test
```

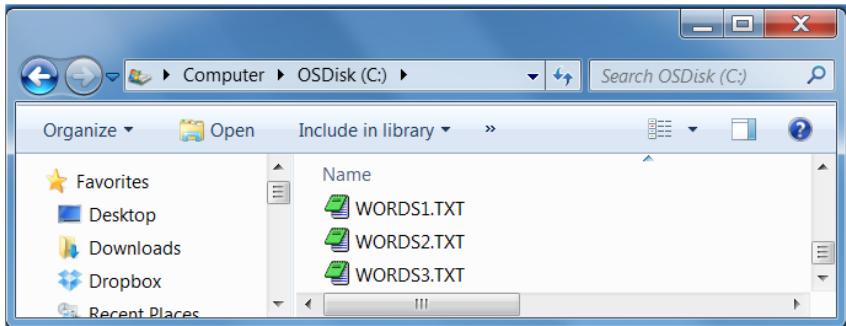
```
{  
    public static void Main()  
    {  
        try  
        {  
            // Create an instance of StreamReader to read from a file. The using statement  
            // also closes the StreamReader.  
            using (StreamReader sr = new StreamReader("TestFile.txt"))  
            {  
                String line;  
                // Read and display lines from the file until the end of the file is reached.  
                while ((line = sr.ReadLine()) != null)  
                {  
                    ListBox.Items.Add(line);  
                }  
            }  
            catch (Exception e)  
            {  
                // Let the user know what went wrong.  
            }  
        }  
    }  
}
```

Here is another example to read in words in a list into a ListBox

```
string fileName = "";  
string buffer = "";  
lbwords.Items.Clear();  
fileName = "c:\\WORDDS1.txt";  
StreamReader rdr = new StreamReader(fileName);  
while (!rdr.EndOfStream)  
{  
    buffer = rdr.ReadLine();  
    lbwords.Items.Add(buffer);  
} rdr.Close();
```

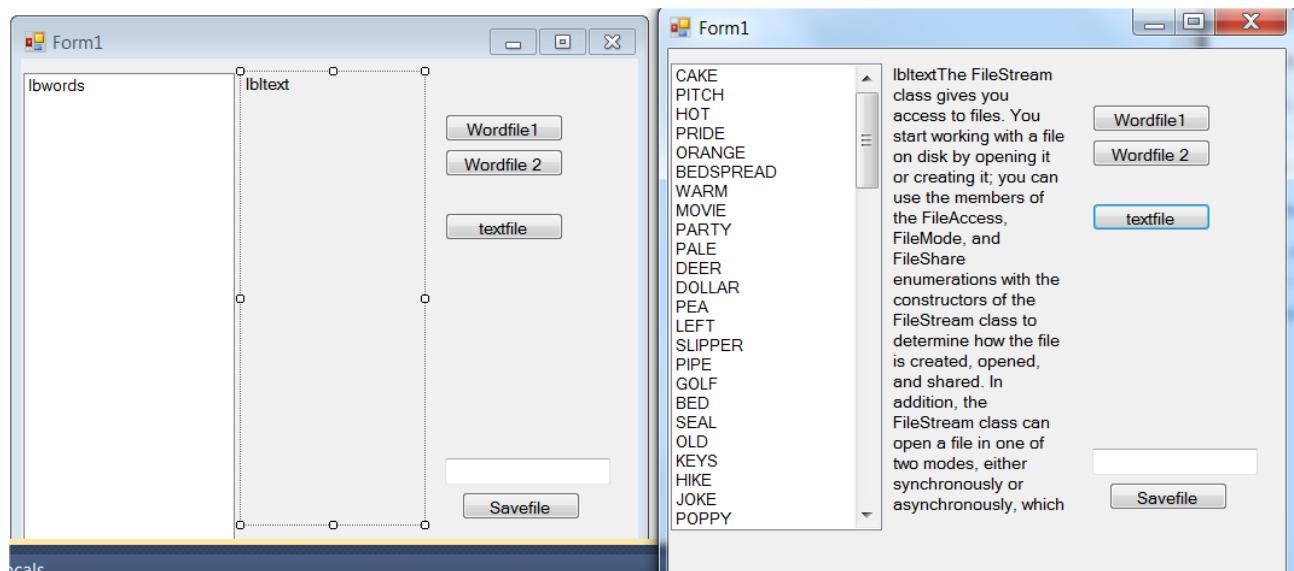
Using StreamReader to load files

Get the three word files from the Goodies folder and pop them in your C drive root. (you can use any files if you wish, these files are just a list of words, one on each line.)



Create the following form. Buttons Wordfile 1 and Wordfile 2 open the words.txt files from above. Button Textfile opens a text file that we make, or get later. At the bottom we have a textbox to put the filename of the saved file in.

To the left are a ListBox (lbwords) and a label(lbltext) with the autosize turned off.



Open a list file with StreamReader

Button 1 opens the Words1 text file and imports the list line by line into a ListBox with the While Statement.

```
using System;
using System.Windows.Forms;
using System.IO; //don't forget this namespace
namespace C_Sharp_streamreader
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void btn1_Click(object sender, EventArgs e)
        {
            string text, fileName = "";
            fileName = "c:\\\\Words1.txt";
            lbwords.Items.Clear(); // clear the list of the last entries
//Open a streamreader
            StreamReader reader = new StreamReader(fileName);
//run until you reach the end of the stream
            while (reader.EndOfStream == false)
            {
                text = reader.ReadLine(); // add data in line by line
                lbwords.Items.Add(text);
            }
            reader.Close(); //don't forget to close the reader
        }
    }
}
```

The Using Statement – Make sure you are using ‘Using’

Button 2 uses the **USING** statement to produce shorter code. Note the difference in the **WHILE** statement as well,

This example below uses **USING** `using (StreamReader sr = new StreamReader("TestFile.txt"))` which also closes the StreamReader when it finishes.

The **using** statement allows the programmer to specify when objects that use resources should release them. In this case it means that `reader.Close();` isn't needed. When Using is finished, it then disposes of the resource.

A **using** statement can be exited either when the end of the **using** statement is reached or if an exception is thrown and control leaves the statement block before the end of the statement.

```
using (something that takes resources)
{
}
```

```
private void btn2_Click(object sender, EventArgs e)
{
    lbwords.Items.Clear();
    // Create an instance of StreamReader to read from a file. The using statement
    // also closes the StreamReader.
    using (StreamReader reader = new StreamReader("c:\\\\Words2.txt"))
    {
        String line = "";
        // Read and display lines from the file until the end of the file is reached.
        While its not null.
        while ((line = reader.ReadLine()) != null)
        {
            lbwords.Items.Add(line);
        }
    }
}
```

[StreamWriter to save a list to a file.](#)

In this case we are taking the list from the ListBox, as a result we have to count out the entries in the ListBox and then use a for loop to write them to the file.

We also have a textbox holding the name of the file that you want to save to.

```
private void btnsave_Click(object sender, EventArgs e)
{
    string fileName, word = "";
    //count how many items are in the listbox
    int total = lbwords.Items.Count;

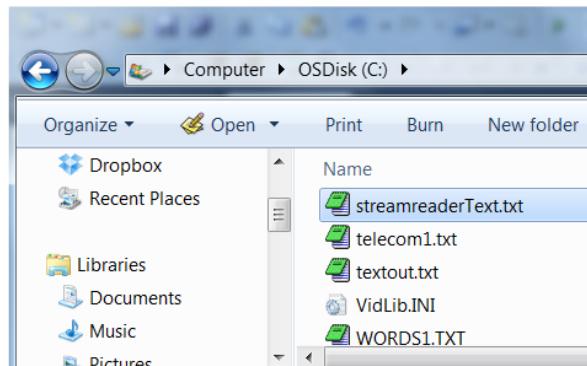
    //if you have written something in the text filename box
    if (txtfilename.Text != "")
    {
        //append it to the c: drive prefix
        fileName = "C:\\\" + txtfilename.Text;
        StreamWriter writer = new StreamWriter(fileName);

        //count through all the items on the list
        for (int j = 0; j < total; j++)
        {
            //pass the item to the word variable
            word = (string)lbwords.Items[j];
            //write the word to the file
            writer.WriteLine(word);
        }
        //close the streamwriter when finished
        writer.Close();
        MessageBox.Show(fileName + " saved");
    }
}
```

Open a text file with StreamReader

This is easier than opening a list and follows the same principles.

Create a text file with some paragraphs of text and save it to your C Drive as streamreadertext.txt



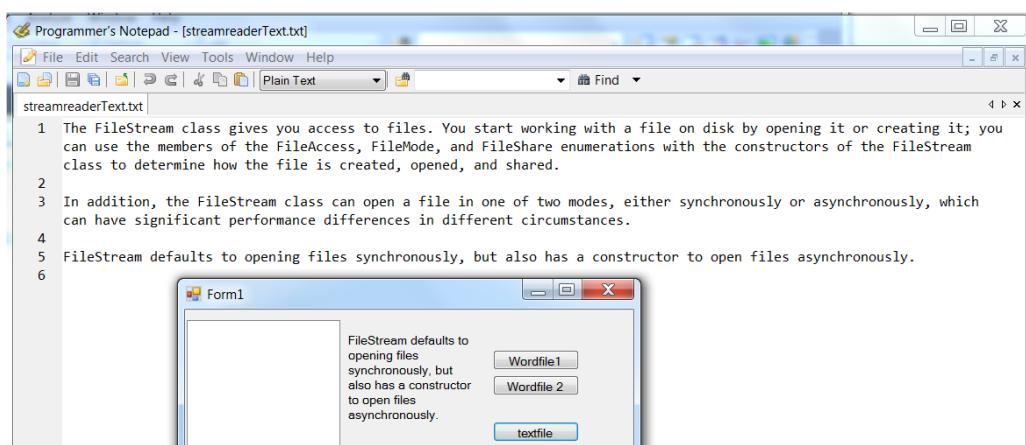
Data is read in line by line regardless of what the format of the text is.

Before we used a list, with hard returns on each line, now it's just text with a return at the end of the paragraph. Each paragraph counts as a line then.

Otherwise the code is similar to the earlier one

```
private void btntextfile_Click(object sender, EventArgs e)
{
    string line = "";
    using (StreamReader reader = new StreamReader("c:\\streamreaderText.txt"))
    {
        while ((line = reader.ReadLine()) != null)
        {
            lbltext.Text = line;
        }
    }
}
```

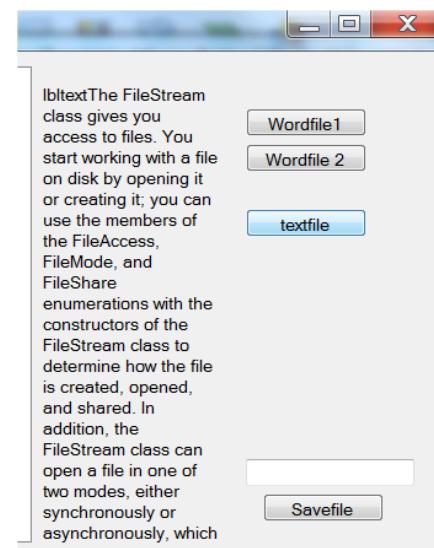
But wait! We have a problem look what happens when we run the program. Only the last line shows in the label. Why?



All lines are passed to the label but each line is overwritten by the line after it. So the last line in the text is the only line left.

If we change the code to append or join each line to the existing text in the text box it suddenly works.

```
lbltext.Text += line;
```

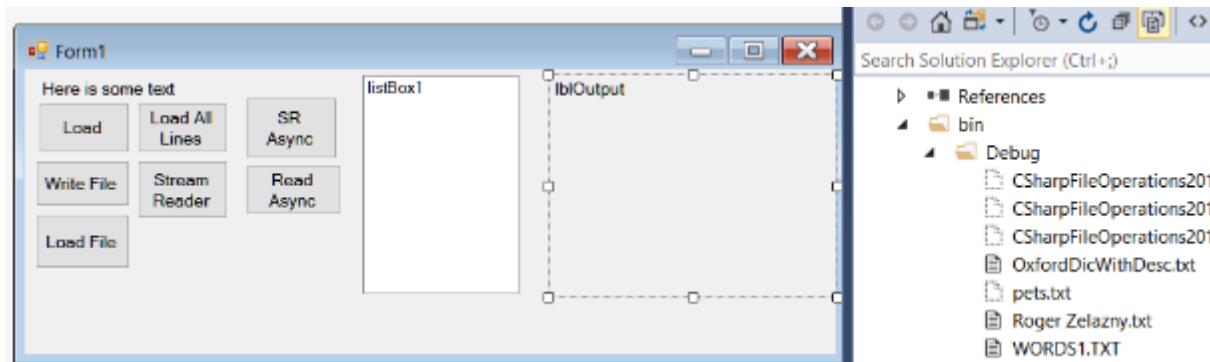


Now that you can open the file, use the StreamWriter to save the file to a filename.

Using Async to load files.

When loading really big files you don't want the program to be inoperable until the process is finished do load large files async.

Add a couple more buttons on the form **btnSRasync** and **btnReadAsync**, a label called **lblOutput**, add in the Roger Zelazny text file or any big text file.



<https://msdn.microsoft.com/en-ca/library/hh137813.aspx>

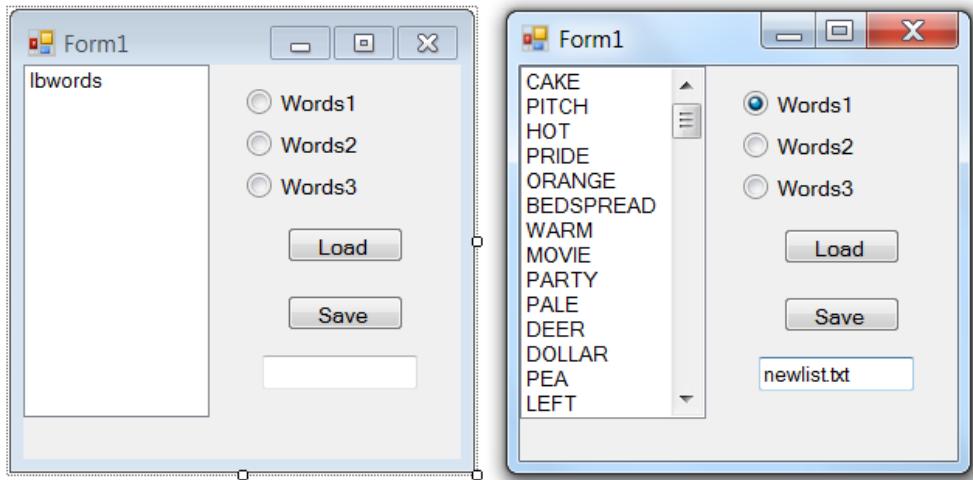
```
private async void btnSRasync_Click(object sender, EventArgs e)
{
    byte[] buffer;
    using (FileStream SourceStream = File.Open("Roger Zelazny.txt", FileMode.Open))
    {
        buffer = new byte[SourceStream.Length];
        await SourceStream.ReadAsync(buffer, 0, (int)SourceStream.Length);
    }
    lblOutput.Text = System.Text.Encoding.ASCII.GetString(buffer);
}
```

<http://stackoverflow.com/questions/13167934/how-to-async-files-readalllines-and-await-for-results>

```
private async void btnReadAsync_Click(object sender, EventArgs e)
{
    using (var reader = File.OpenText("Roger Zelazny.txt"))
    {
        lblOutput.Text = await reader.ReadToEndAsync();
    }
}
```

StreamReader and Sender Exercise

This opens one of three words files with StreamReader, the files are selected using the Sender.. The resulting list generated is then saved using StreamWriter.



```
public partial class Form1 : Form
{
    string fileName = "";
    public Form1()
    {
        InitializeComponent();
    }
```

```
private void btnload_Click(object sender, EventArgs e)
{
    //clear the listbox
    lbwords.Items.Clear();

    //read in the file
    StreamReader reader = new StreamReader(fileName);
    while (!reader.EndOfStream)
    {
        lbwords.Items.Add(reader.ReadLine());
    }
    reader.Close();
}
```

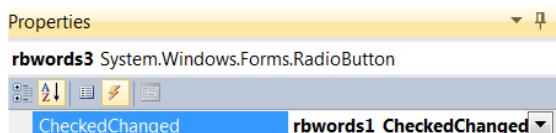
```
private void rbwords1_CheckedChanged(object sender, EventArgs e)
{
```

```

//create a fake radiobutton named FakeRB and pass all its properties over
using sender from the RB clicked
RadioButton fakeRB = (RadioButton)sender;
//look for the name of the button clicked and open the file
switch (fakeRB.Text)
{
    case "Words1":
        fileName = "c:\\Words1.txt";
        break;
    case "Words2":
        fileName = "c:\\Words2.txt";
        break;
    case "Words3":
        fileName = "c:\\Words3.txt";
        break;
}
}

```

The private void rbwords1_CheckedChanged above needs to be on the CheckChanged event for each of the three radio buttons. **This means that each radio button goes to that one piece of code and runs it.**



Sender passes all the properties from the radiobutton through to a 'fake' radiobutton, that holds the properties of the real radiobutton. The case statement looks for the Text on the fakerradiobutton, (fakeRB.Text) and depending on what's written there runs each of the case statements.

```

private void btnsave_Click(object sender, EventArgs e)
{
    string fileName = "";

    // string word = "";
    int total = lbwords.Items.Count;
    try
    {
        if (txtfilename.Text != "")
        {
            //I wrote it to the temp file as it didn't want to go to C Drive
            fileName = "C:\\Temp\\\" + txtfilename.Text;
            StreamWriter strwriter = new StreamWriter(fileName);

            for (int j = 0; j < total; j++)

```

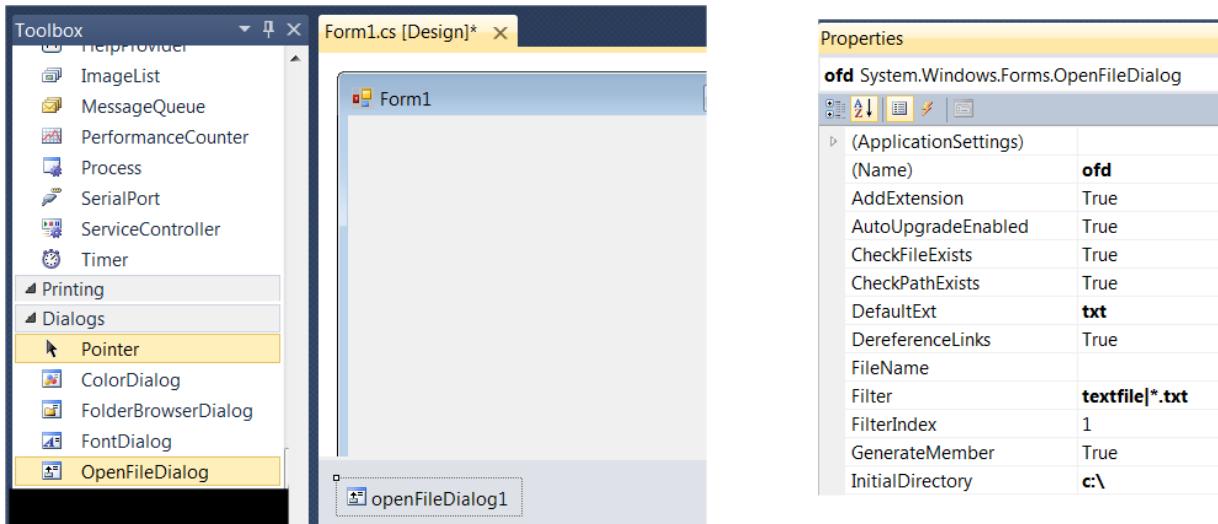
```
{  
    // word = Convert.ToString(lbwords.Items[j]);  
    strwriter.WriteLine(lbwords.Items[j]);  
}  
strwriter.Close();  
MessageBox.Show(fileName + " saved");  
}  
}  
catch (Exception error)  
{  
    MessageBox.Show("Nope, thats not going to work, did you put a .txt on it or  
can you write to the C drive OK? " + error);  
    // throw;  
}  
}  
}  
}
```

Open File Dialog

The **OpenFileDialog** component allows users to browse the folders of their computer or any computer on the network and select one or more files to open. The dialog box returns the path and name of the file the user selected in the dialog box.

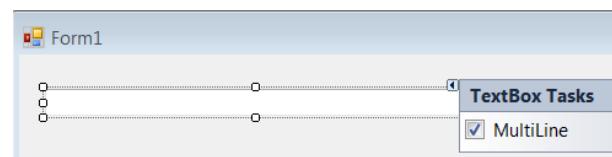
Once the user has selected the file to be opened you can create an instance of the **StreamReader** class to open the file.

Open a new project find Open File Dialog and drag it onto your form.

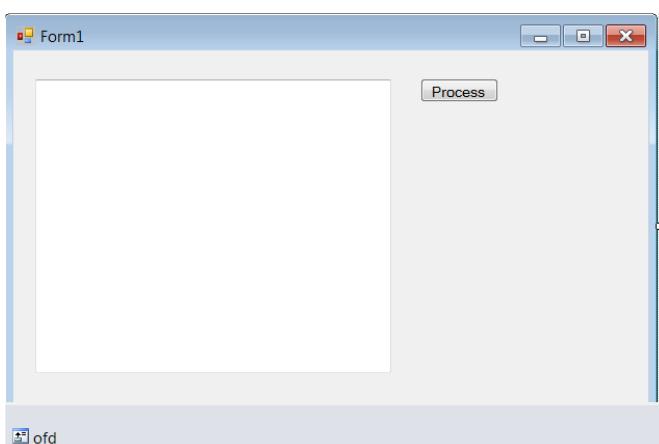


Rename the Properties to ofd. And filter it for only text files. See above for settings.

Add a textbox and make it multiline



Add a button, Btnrun, and stretch out the text box



Under the button add the following code to test that your Open File Dialog works and passes through the filename you select with ofd.filename

```
private void btnrun_Click(object sender, EventArgs e)
{
    //if the ofd clicked is Open (Yes or OK) then ...
    if (ofd.ShowDialog() == DialogResult.OK)
    { // show the file name you have selected
        MessageBox.Show(ofd.FileName);
    }
}
```

Once that is working lets add in the StreamReader code.

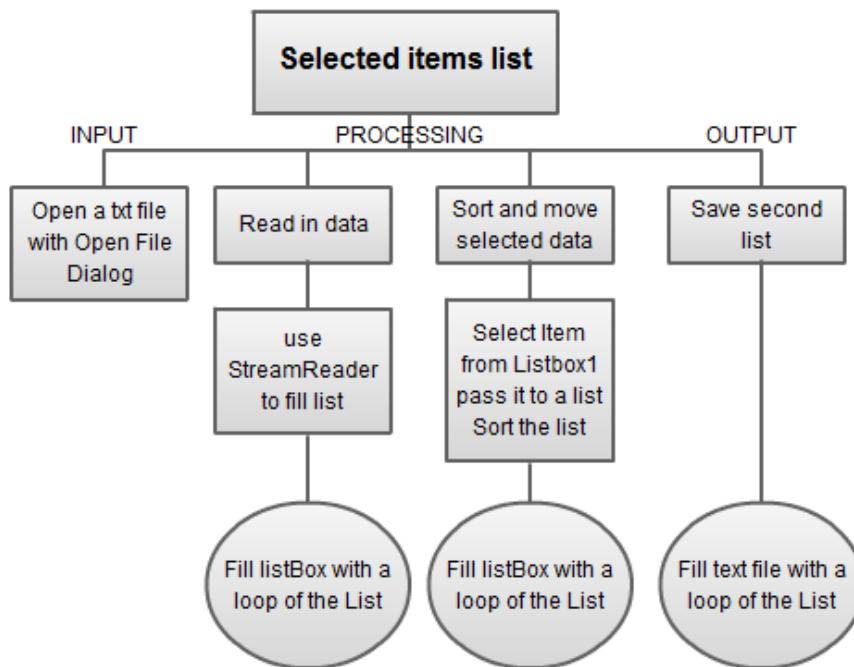
```
private void btnrun_Click(object sender, EventArgs e)
{
    //if the ofd clicked is Open (Yes or OK) then ...
    if (ofd.ShowDialog() == DialogResult.OK)
    {
        StreamReader reader = new StreamReader(ofd.FileName);
        string text = "";
        //note the new != NOT so while its NOT end of the stream
        while (!reader.EndOfStream)
        {
            text += reader.ReadLine();
        }
        txtmemo.Text = text;
        reader.Close();
    }
}
```

Open, Select, Sort and Save a list exercise

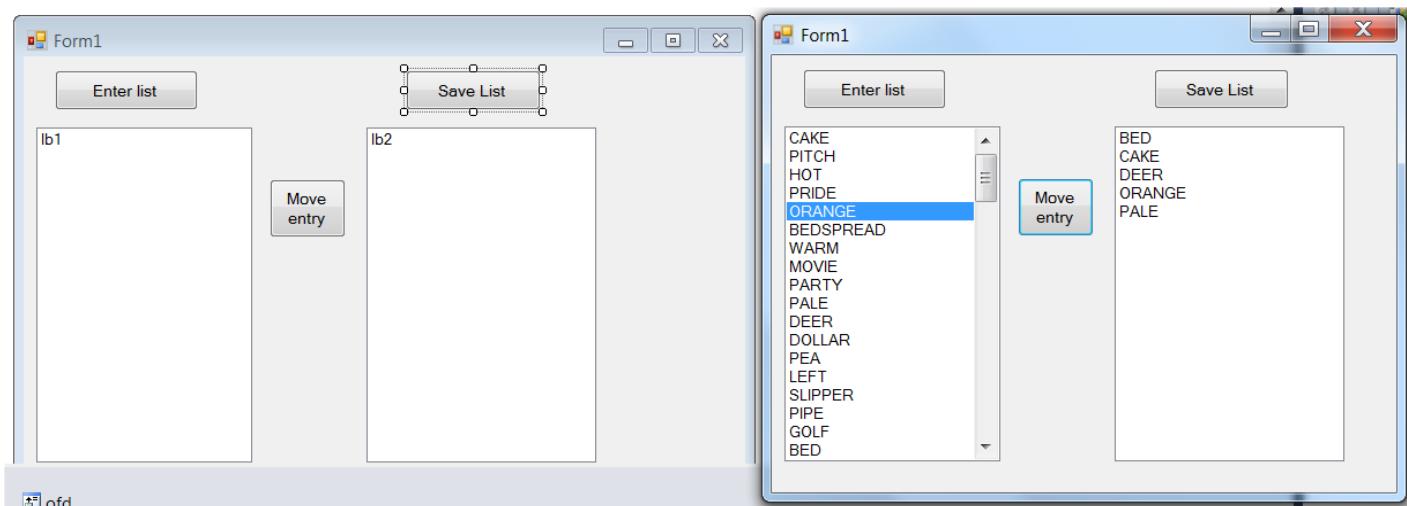
This exercise uses the elements we have learned earlier.

We are going to Open a list file (Words1, Words2, or Words3) using Open file dialog.

Read in the data to a list then fill it to a ListBox. When you click on an item in the ListBox it gets added to a second list, then that list is sorted, and the output is passed to another ListBox. The new list is then saved back to the drive and a message tells the user that the file has been saved.

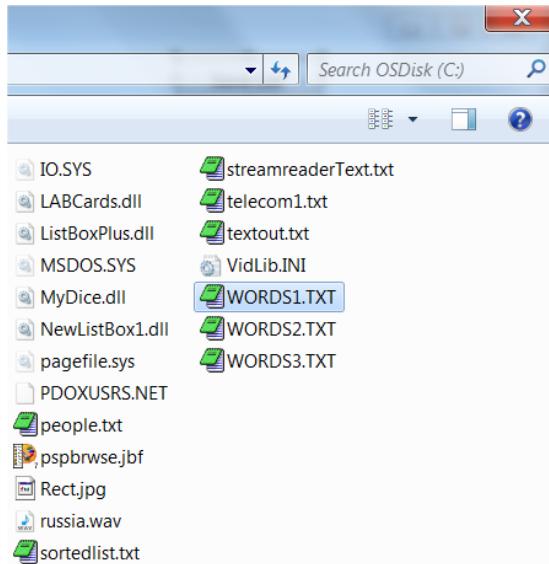


I have used a Move Entry button to add the selected item to the list, however can you make it so that clicking on the item automatically moves it to the list and then fills the second text box.

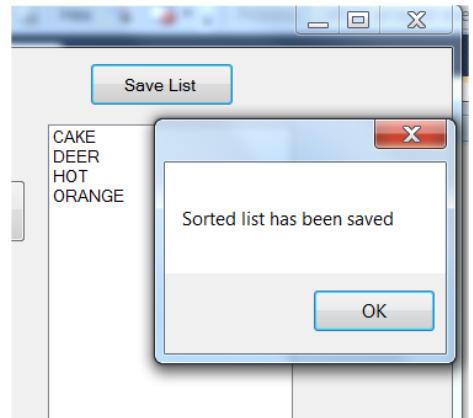


Note that the second ListBox is filled with a sorted list of words.

Open File Dialog



Saving the file as Sortedlist.txt



Also:

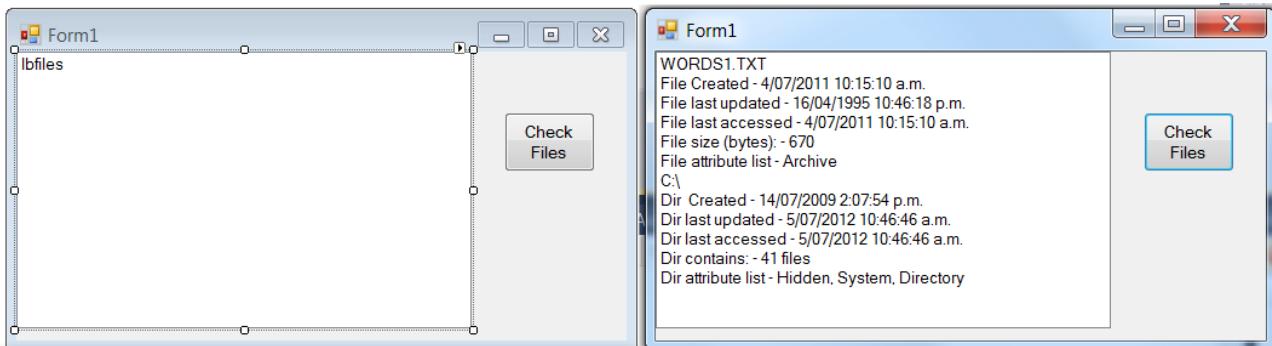
Don't let blank entries be added to the second list.

Sort the first list before it loads into the ListBox.

Create a reset so you can load lists, this involves clearing the ListBox and the first list.

File and Directory Information Program

The following application takes a file path and then displays information about the file, the containing directory, and the drive. Create an **Open File Dialog** to access the drive and file that you want.



The code shows the **File** and the **Directory**, carry on and create the code for the **Drive** as well.

```
private void btncheck_Click(object sender, EventArgs e)
{
    string filename = "";
    if (ofd.ShowDialog() == DialogResult.OK)
    {
        // show the file name you have selected
        filename = ofd.FileName;
    }
    FileInfo file = new FileInfo(filename);
    if (file.Exists)
    {
        lbfiles.Items.Add(file.Name);
        lbfiles.Items.Add("File Created - " + file.CreationTime.ToString());
        lbfiles.Items.Add("File last updated - " + file.LastWriteTime.ToString());
        lbfiles.Items.Add("File last accessed - " + file.LastAccessTime.ToString());
        lbfiles.Items.Add("File size (bytes): - " + file.Length.ToString());
        lbfiles.Items.Add("File attribute list - " + file.Attributes.ToString());
    }
    DirectoryInfo dir = file.Directory;
    if (dir.Exists)
    {
        lbfiles.Items.Add(dir.Name);
        lbfiles.Items.Add("Dir Created - " + dir.CreationTime.ToString());
        lbfiles.Items.Add("Dir last updated - " + dir.LastWriteTime.ToString());
        lbfiles.Items.Add("Dir last accessed - " + dir.LastAccessTime.ToString());
        lbfiles.Items.Add("Dir contains: - " + dir.GetFiles().Length.ToString() + " files");
        lbfiles.Items.Add("Dir attribute list - " + dir.Attributes.ToString());
    }
}
```

Later on we will expand the program to copy and move directories and files. Useful for backing data.

Tip of the day program

Create a program that opens a text file of sayings or quotes, loads them into an array and shows a random one each time you press a button.

18. Introduction to Classes

The Object Orientated Toaster ...

Once upon a time, in a kingdom not far from here, a king summoned two of his advisors for a test. He showed them both a shiny metal box with two slots in the top, a control knob, and a lever. "What do you think this is?"

One advisor, an Electrical Engineer, answered first. "It is a toaster," he said.

The king asked, "How would you design an embedded computer for it?"

The advisor: "Using a four-bit microcontroller, I would write a simple program that reads the darkness knob and quantifies its position to one of 16 shades of darkness, from snow white to coal black.

The program would use that darkness level as the index to a 16-element table of initial timer values. Then it would turn on the heating elements and start the timer with the initial value selected from the table.

At the end of the time delay, it would turn off the heat and pop up the toast. Come back next week, and I'll show you a working prototype."

The second advisor, a software developer, immediately recognized the danger of such short-sighted thinking. He said, "Toasters don't just turn bread into toast, they are also used to warm frozen waffles.

What you see before you is really a breakfast food cooker. As the subjects of your kingdom become more sophisticated, they will demand more capabilities.

They will need a breakfast food cooker that can also cook sausage, fry bacon, and make scrambled eggs. A toaster that only makes toast will soon be obsolete. If we don't look to the future, we will have to completely redesign the toaster in just a few years."

"With this in mind, we can formulate a more intelligent solution to the problem. First, create a class of breakfast foods. Specialize this class into subclasses: grains, pork, and poultry.

The specialization process should be repeated with grains divided into toast, muffins, pancakes, and waffles; pork divided into sausage, links, and bacon; and poultry divided into scrambled eggs, hard-boiled eggs, poached eggs, fried eggs, and various omelette classes."

"The ham and cheese omelette class is worth special attention because it must inherit characteristics from the pork, dairy, and poultry classes. Thus, we see that the problem cannot be properly solved without multiple inheritance.

At run time, the program must create the proper object and send a message to the object that says, 'Cook yourself.' The semantics of this message depend, of course, on the kind of object, so they have a different meaning to a piece of toast than to scrambled eggs."

"Reviewing the process so far, we see that the analysis phase has revealed that the primary requirement is to cook any kind of breakfast food.

In the design phase, we have discovered some derived requirements.

Specifically, we need an object-oriented language with multiple inheritance. Of course, users don't want the eggs to get cold while the bacon is frying, so concurrent processing is required, too."

"We must not forget the user interface. The lever that lowers the food lacks versatility, and the darkness knob is confusing. Users won't buy the product unless it has a user-friendly, graphical interface.

When the breakfast cooker is plugged in, users should see a cowboy boot on the screen. Users click on it, and the message 'Booting UNIX v.8.3' appears on the screen. (UNIX 8.3 should be out by the time the product gets to the market.) Users can pull down a menu and click on the foods they want to cook."

"Having made the wise decision of specifying the software first in the design phase, all that remains is to pick an adequate hardware platform for the implementation phase. An Intel Pentium with 48MB of memory, a 1.2GB hard disk, and a SVGA monitor should be sufficient.

If you select a multitasking, object oriented language that supports multiple inheritance and has a built-in GUI, writing the program will be a snap."

The king wisely had the software developer beheaded, and they all lived happily ever after.

From [Reddit](#) More fun on the link...

Properties and their Generation

What are Properties?

Properties provide the opportunity to protect a **field** in a class by reading and writing to it through that property.

Properties combine aspects of both fields (variables) and methods.

```
//****Declare Properties ****
public double num1 { get; set; }
public double num2 { get; set; }
```

To the user of an object, a property appears to be a field, accessing the property requires the same syntax. To the implementer of a class, a property is one or two code blocks, representing a **get** accessor and/or a **set** accessor.

The code block for the get accessor is executed when the property is **read (data out)**; the code block for the set accessor is executed when the property is **assigned a new value (data in)**.

A property without a set accessor is considered read-only. A property without a get accessor is considered write-only. A property that has both accessors is read-write.

Always favour properties over fields for public members. An automatically implemented property looks like a field to the outside world, but you can always add extra behaviour when necessary.

Automatically generate a Short Property layout

Type in **prop** and press Tab two times. This ...

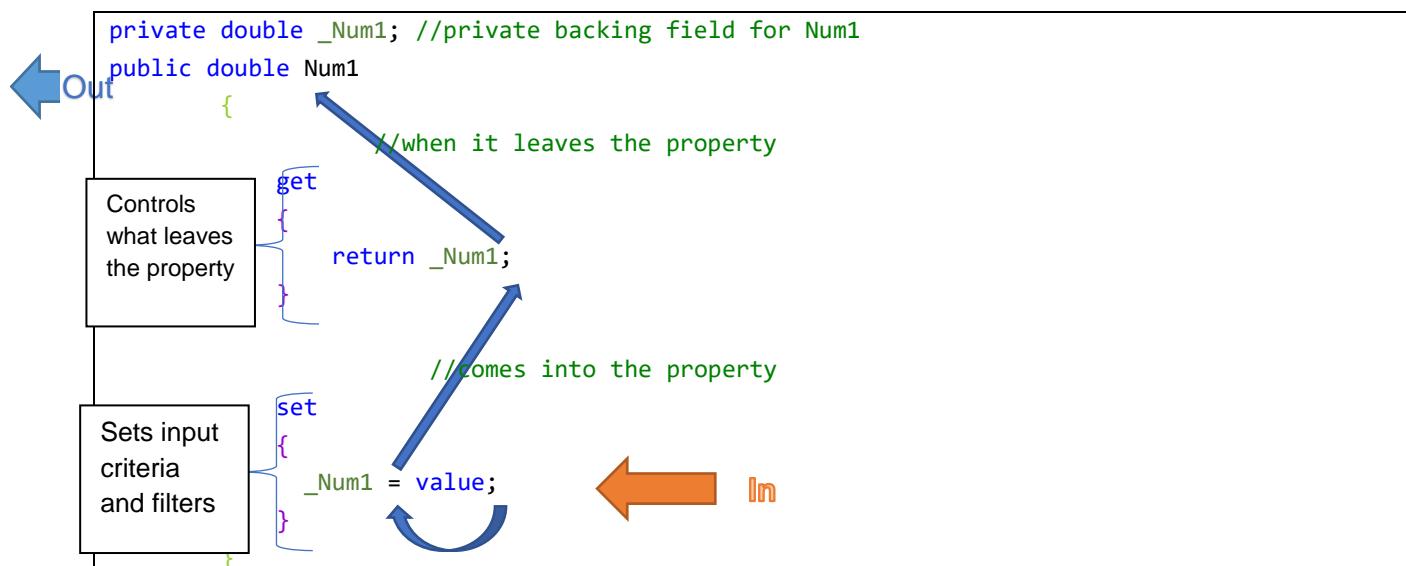
```
public double Num1 { get; set; }
```

... is the same as this

Type **Propfull** and press tab this will give full properties including the fields you need that can be automatically changed as needed.

```
private double _Num3;
public double Num3 {
    get { return _Num3; }
    set { _Num3 = value; }
}
```

Which is the same as this



Unless you are going to use the Get and Set for business rules and business logic then only use **short properties**.

Automatically generate Full Properties with Propfull

Type **Propfull** and press tab this will give full properties including the fields you need that can be automatically changed as needed.

```
private int _MyProperty;
    public int MyProperty
    {
        get { return _MyProperty; }
        set { _MyProperty = value; }
    }
```

Automatically generate Summary Comments that appear on the tooltip

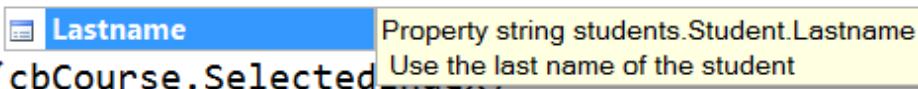
Press /// three times, and a summary code block is added to your code.

```
/// <summary>
/// Use the last name of the student
/// </summary>
```

If you put it directly above the property then the tooltip will appear on that property

```
public String Lastname { get; set; }
```

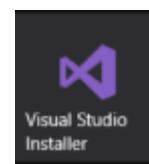
```
mystudent.1
```



```
switch (cbCourse.Selected.....)
```

How to create Class Diagrams and Code Map

Add in the Two Components. Open the Visual Studio Installer



What happened to Class Diagrams?

Modifying - Visual Studio Enterprise 2017 (15.0.26228.9)

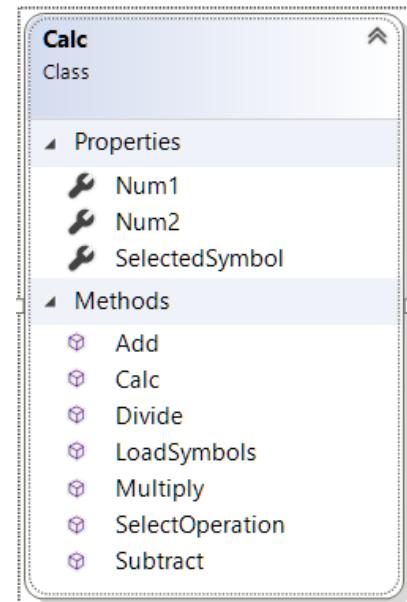
Workloads Individual components Language packs

- SQL Server Express 2016 LocalDB
- SQL Server Native Client
- Web Deploy

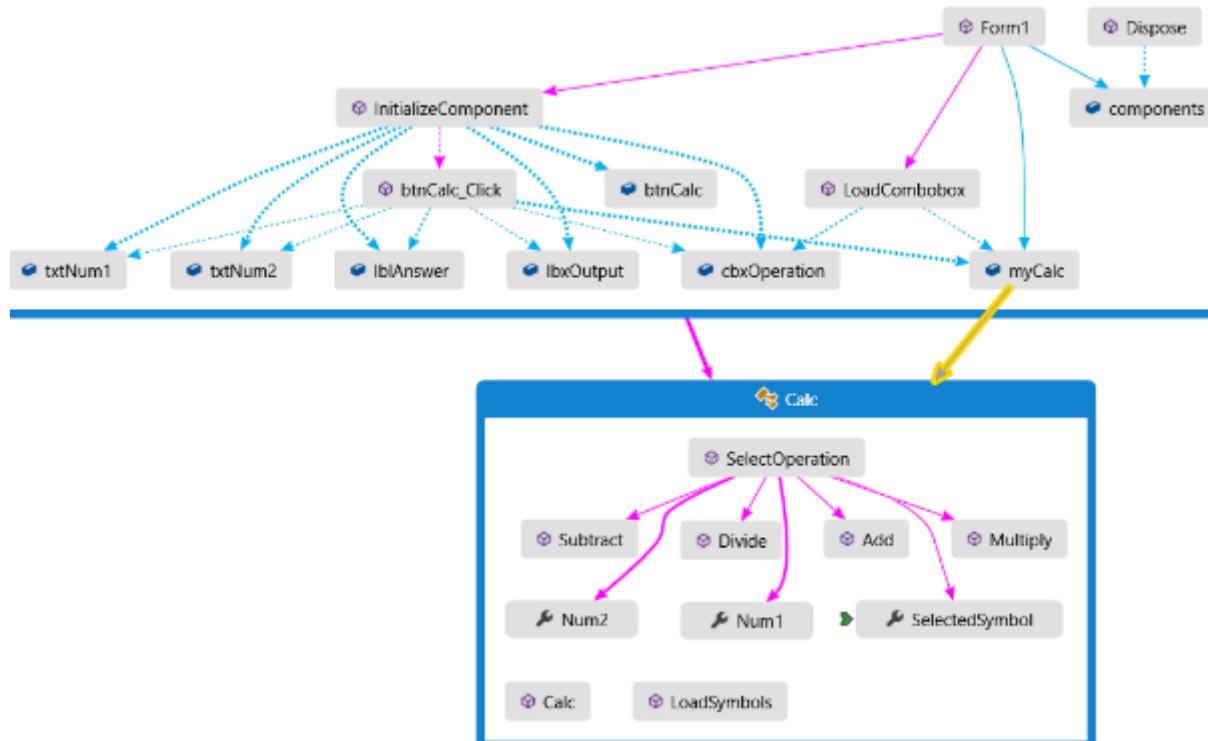
Code tools

- Class Designer
- ClickOnce Publishing
- Code Clone
- Code Map
- Developer Analytics tools
- DGML editor
- Git for Windows
- Help Viewer
- LINQ to SQL tools
- Live Dependency Validation
- NuGet package manager
- PowerShell tools
- PreEmptive Protection - Dotfuscator
- Static analysis tools
- Text Template Transformation

Class Designer



Code Map



New C# Property features in C# 6

<http://www.pluralsight.com/courses/csharp-6-whats-new>

Auto Property Initializers.

You can use an auto implemented property and still give it an initial value in one action.

So here we are assigning the value to the Get. Therefore, you don't need to write a constructor method to pass the data across.

Old way...

```
public User()
{
    Id = Guid.NewGuid();
}

public Guid Id { get; protected set; }
```

This is instead of using the Constructor methods and then assigning the value to the property.

New way...

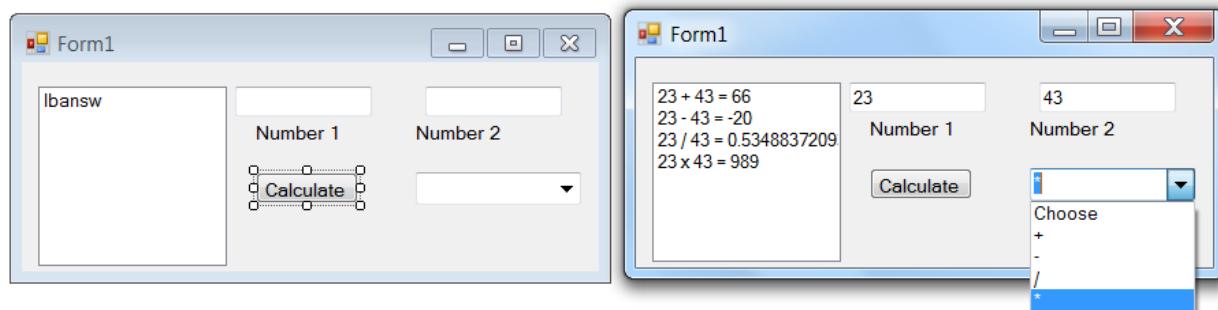
```
public Guid Id { get; protected set; } = Guid.NewGuid();
```

Guid.NewGuid creates a unique globally unique identifier (GUID)

This is a convenient **static** method that you can call to get a new [Guid](#).

Calculator with Class – Intro to classes

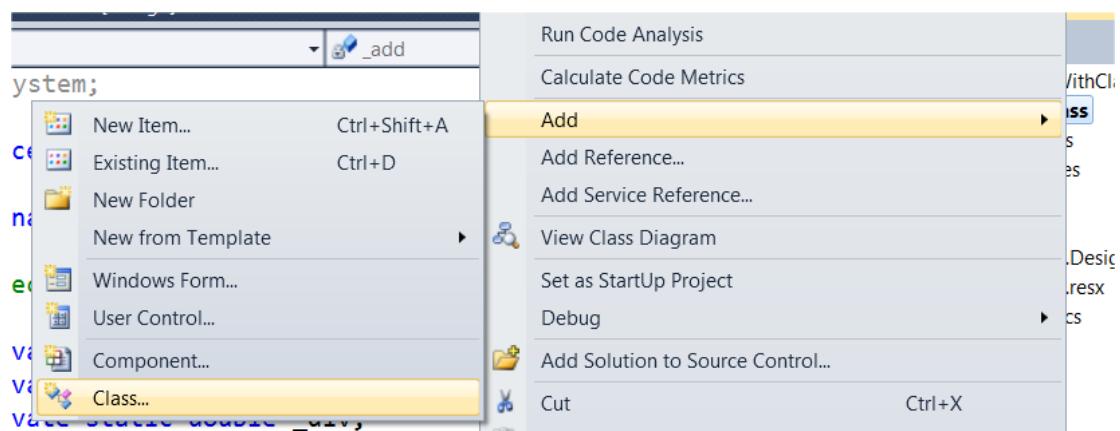
This is an introductory lesson, to show the principles behind OOP. We are going to make another calculator.



Let's start by making the Class then putting the Form code last.

The Calc class.

To add a class to your project right click on the name of the project and go Add and click on class



Name it **Calc**.

The Standard class layout

The standard layout for a class is as follows. We will cover all these internals later. This lesson just uses **Fields**, **Properties**, and **Methods**

- Fields
- Constructors
- Nested Enums, structs, classes
- Properties
- Methods

Here is our class below we can see it by going to **View Class Diagram** by right clicking on the class file

```
namespace CalcWithClass
{
    public class Calc
    {
        //****Declare Fields - None ****
        //****Declare Properties ****
        public double Num1 { get; set; }
        public double Num2 { get; set; }

        //show the symbol in the listbox
        public string SelectedSymbol { get; set; }
        //****Set up the constructors****
        //(no constructors needed)

        //****Create the methods****
        /// <summary>
        /// Add two numbers
        /// </summary>
        public double Add(double one, double two)
        {
            return one + two;
        }
        /// <summary>
        /// Subtract two numbers
        /// </summary>
        public double Subtract(double one, double two)
        {
            return one - two;
        }
        /// <summary>
        /// Divide two numbers
        /// </summary>
        public double Divide(double one, double two)
        {
            return one / two;
        }
        /// <summary>
```

A screenshot of Microsoft Visual Studio showing the code editor with the provided C# code. A callout box from the 'Add' method's implementation points to the method signature in the code. The callout box contains the following information:

- double Calc.Add(double one, double two)
- Add two numbers

```
/// Multiply two numbers
/// </summary>
public double Multiply(double one, double two)
{
    return one * two;
}

/// <summary>
/// Load the symbols to the ComboBox
/// </summary>
public string[] LoadSymbols()
{
    //make an array of symbols
String[] ComboBoxItems = new[] { "Choose an operation", "*", "-", "+", "/" };
    return ComboBoxItems;
}
/// <summary>
/// Input the Symbol and select the calculation
/// </summary>
public double SelectOperation()
{
    double Answer = 0;
    switch (SelectedSymbol)
    {
        case "*":
            Answer = Multiply(Num1, Num2);
            break;
        case "-":
            Answer = Subtract(Num1, Num2);
            break;
        case "+":
            Answer = Add(Num1, Num2);
            break;
        case "/":
            Answer = Divide(Num1, Num2);
            break;
    }
    return Answer;
}
```

Form Details

To join our calc class to our form we have to declare it and create a new instance of it

```
private calc mycalc = new calc();
```

This means that everything in the **calc** class that is public can be access by myCalc

```
public partial class Form1 : Form
{
    //create a new instance of the Calc class
    private Calc myCalc = new Calc();

    public Form1()
    {
        InitializeComponent();
        LoadCombobox();
    }

    private void LoadCombobox()
    {
        //pass the array across to the combobox
        cbxOperation.Items.AddRange(myCalc.LoadSymbols());
        //start at the first item
        cbxOperation.SelectedIndex = 0;
    }

    //the button click,
    private void btnCalc_Click(object sender, EventArgs e)
    {
        double result;
        //Check for numbers coming in
        if (double.TryParse(txtNum1.Text, out result))
        {
            myCalc.Num1 = result;
        }

        if (double.TryParse(txtNum2.Text, out result))
        {
            myCalc.Num2 = result;
        }
        //check selected operation
        myCalc.SelectedSymbol = cbxOperation.SelectedItem.ToString();

        //output answer
        lblAnswer.Text = Convert.ToString(myCalc.SelectOperation());
        lbxOutput.Items.Add(myCalc.Num1 + " " + myCalc.SelectedSymbol + " " + myCalc.Num2 + " = " +
        lblAnswer.Text);
    }
}
```

19. ADO.Net and SQL Database Creation

Watch these video series

The heart of creating databases is the language most used to access and manipulate the data. That language is SQL - Structured Query Language.

SQL is just as important as any programming language you want to learn. Every programmer needs to have a good grasp of SQL.

<https://msdn.microsoft.com/en-us/library/aa302325.aspx> Learn ADO.net

Introduction to SQL

This course is a gentle introduction to Structured Query Language (SQL).

<http://pluralsight.com/training/Courses/TableOfContents/introduction-to-sql>

Database Fundamentals

Developing Microsoft SQL Server Databases

Microsoft MTA: Database Administration Fundamentals

This is a practical course for someone new to relational databases that moves you through basic concepts right into real-world usage, demonstrating core tasks in Microsoft SQL Server itself.

<http://pluralsight.com/training/Courses/TableOfContents/database-admin-fundamentals>

More Advanced – Integration with Visual Studio

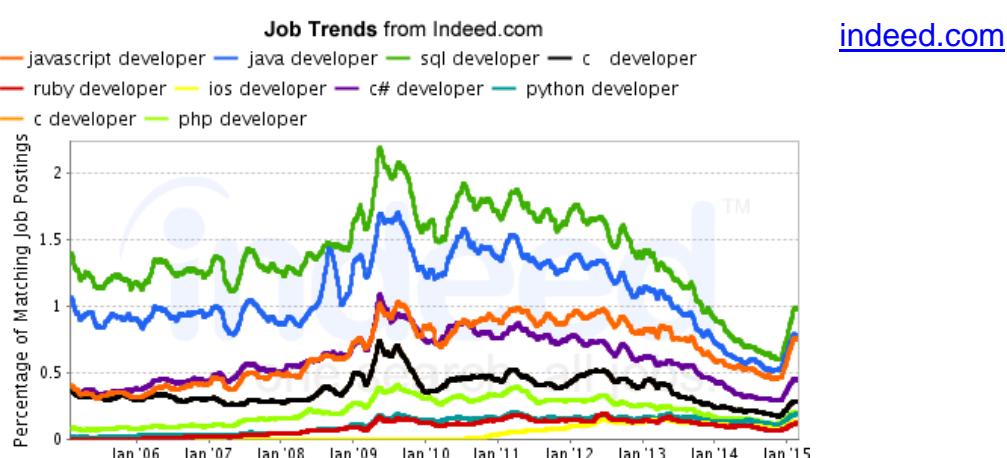
<http://pluralsight.com/training/Courses/TableOfContents/sql-server-tsql>

ADO.Net Very important in C#

<http://pluralsight.com/training/Courses/TableOfContents/adodotnet-fundamentals>

Learn SQL

<http://www.tutorialspoint.com/sql/> <https://www.codeschool.com/courses/try-sql>



TSQL Cheat Sheet

String Functions

Exact Numerics

| | | |
|---------|---------|----------|
| Bit | Tinyint | Smallint |
| Bigint | Decimal | Money |
| Numeric | | |

Approximate Numerics

| | |
|-------|------|
| Float | Real |
|-------|------|

Date and Time

| | |
|---------------|----------|
| Smalldatetime | Datetime |
| Timestamp | |

Strings

| | | |
|------|---------|------|
| Char | Varchar | Text |
|------|---------|------|

Unicode Strings

| | | |
|-------|----------|-------|
| Nchar | Nvarchar | Ntext |
|-------|----------|-------|

Binary Strings

| | |
|-----------|-------|
| Binary | Image |
| Varbinary | |

Miscellaneous

| | | |
|-------------|-------|-----|
| Cursor | Table | Xml |
| Sql_variant | | |

Common Functions

| | |
|---------|----------------------------|
| AVG() | Returns the average value |
| COUNT() | Returns the number of rows |
| FIRST() | Returns the first value |
| LAST() | Returns the last value |
| MAX() | Returns the largest value |
| MIN() | Returns the smallest value |
| SUM() | Returns the sum |

UCASE() - Converts a field to upper case
LCASE() - Converts a field to lower case
MID() - Extract characters from a text field
LEN() - Returns the length of a text field
ROUND() - Rounds a numeric field to the number of decimals specified
NOW() - Returns the current system date and time
FORMAT() - Formats how a field is to be displayed

Date Functions

| |
|----------------------------------|
| DATEADD (datepart, number, date) |
| DATEDIFF (datepart, start, end) |
| DATENAME (datepart, date) |
| DATEPART (datepart, date) |
| DAY (date) |

GETDATE()
 GETUTCDATE()
 MONTH (date)
 YEAR (date)

Operators Allowed in the WHERE Clause

With the WHERE clause, the following operators can be used:

| Operator | Description |
|-----------------|-----------------------|
| = | Equal |
| <> | Not equal |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal |
| <= | Less than or equal |

BETWEEN Between an inclusive range
 LIKE Search for a pattern
 IN To specify multiple possible values for a column

Note: In some versions of SQL the <> operator may be written as !=

Logical processing order of select

FROM table
ON join condition
JOIN table
WHERE clauses
GROUP BY columns
WITH CUBE / WITH ROLLUP
HAVING condition
SELECT columns
DISTINCT
ORDER BY columns
TOP % or number

CTEs - Common Table Expressions

WITH cteName (columnList)
AS (**SELECT** statement)
SELECT columns
FROM cteName
INNER JOIN table **ON** condition

Recursive CTEs

WITH cteName (columnList)
AS (-- Anchor statement:
SELECT columns **FROM** table...
UNION ALL
 Recursion statement:
SELECT columns **FROM** table...
INNER JOIN cteName **ON** ...

```
)  
SELECT columns  
FROM cteName
```

Over and partition by

```
/* Aggregate functions include COUNT,  
MIN, MAX, AVG, ROW_COUNT(), etc. */  
SELECT  
agg_func(col1) OVER(),  
agg_func(col1)  
OVER(PARTITION BY col2),  
columns  
FROM table...
```

Create a Stored Procedure

```
CREATE PROCEDURE name  
@variable AS datatype = value  
AS  
-- Comments
```

```
SELECT * FROM table
```

```
GO
```

Create a Trigger

```
CREATE TRIGGER name  
ON  
table  
FOR  
DELETE, INSERT, UPDATE  
AS  
-- Comments  
SELECT * FROM table  
GO
```

20. Using SQL Server Management Studio

Setting up SQL Server Management Studio

We are running **Microsoft's SQL Server Management Studio** which can be easily downloaded for free from the Microsoft website. Get the **SQLEXPWT.exe** version.

<https://www.microsoft.com/en-us/download/details.aspx?id=42299>

Express with Tools (SQLEXPWT_Architecture_Language.exe)

This package contains everything needed to install and configure SQL Server as a database server including the full version of SQL Server 2014 Management Studio

Choose the download you want

| <input type="checkbox"/> | File Name | Size |
|-------------------------------------|--|----------|
| <input type="checkbox"/> | ExpressAndTools 32BIT\SQLEXPWT_x86_ENU.exe | 840.8 MB |
| <input checked="" type="checkbox"/> | ExpressAndTools 64BIT\SQLEXPWT_x64_ENU.exe | 833.2 MB |

Don't download the whole SQL Server, its not needed and will make your computer run slow.

Run the installer

These next features are what you can change. Otherwise just click NEXT to bounce through the windows.



New SQL Server stand-alone installation or add features to an existing installation

Launch a wizard to install SQL Server 2014 in a non-clustered environment or to add features to an existing SQL Server 2014 instance.

I think you can get by with just these selected Instance Features, otherwise just Click Select All and install it (I have space issues)

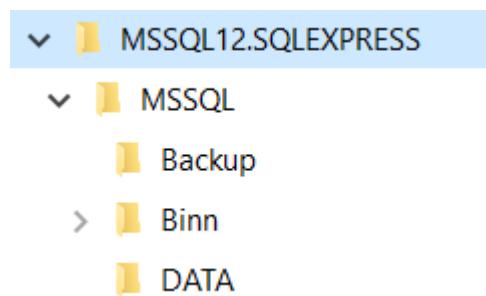
| Features: |
|---|
| Instance Features |
| <input checked="" type="checkbox"/> Database Engine Services |
| <input type="checkbox"/> SQL Server Replication |
| Shared Features |
| <input checked="" type="checkbox"/> Client Tools Connectivity |
| <input type="checkbox"/> Client Tools Backwards Compatibility |
| <input type="checkbox"/> Client Tools SDK |
| <input checked="" type="checkbox"/> Management Tools - Basic |
| <input type="checkbox"/> Management Tools - Complete |
| <input type="checkbox"/> SQL Client Connectivity SDK |
| <input checked="" type="checkbox"/> LocalDB |
| Redistributable Features |

This option will create the folder with the name of the Server Instance on it, You can call it what you like, but inside of it will be all your databases.

Default instance
Named instance: SQLEXPRESS
Instance ID: SQLEXPRESS
SQL Server directory: C:\Program Files\Microsoft SQL Server\MSSQL12.SQLEXPRESS

→ This PC > Local Disk (C:) > Program Files > Microsoft SQL Server > MSSQL12.SQLEXPRESS

Here is where your Databases will be stored as a default, in the Data Folder.

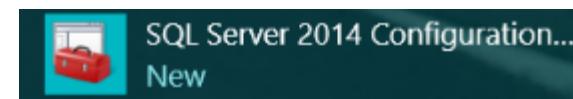


When you have installed it you have the following shortcuts loaded onto your Start menu
→

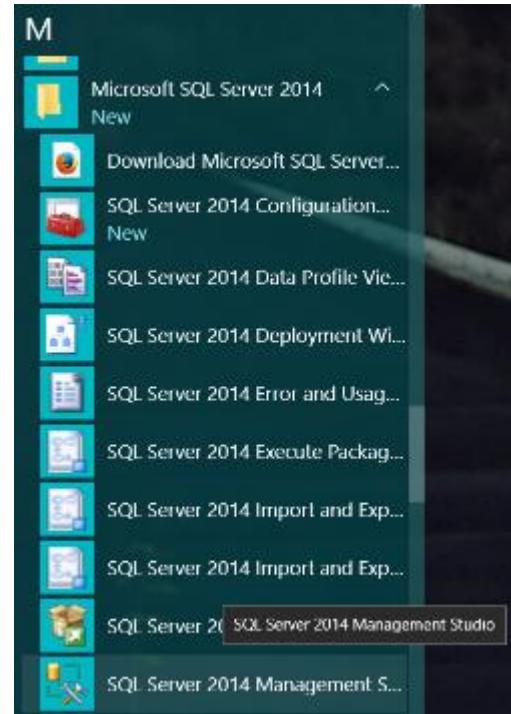
Click on SQL Management Studio to activate the SQL Server



You will want this

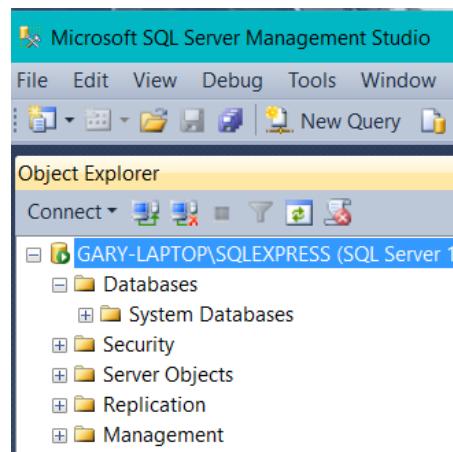


if your server doesn't like to run.



Start the  SQL Server 2014 Management S... and it will load the program.

The top line is the name of your **computer\SQLExpress**

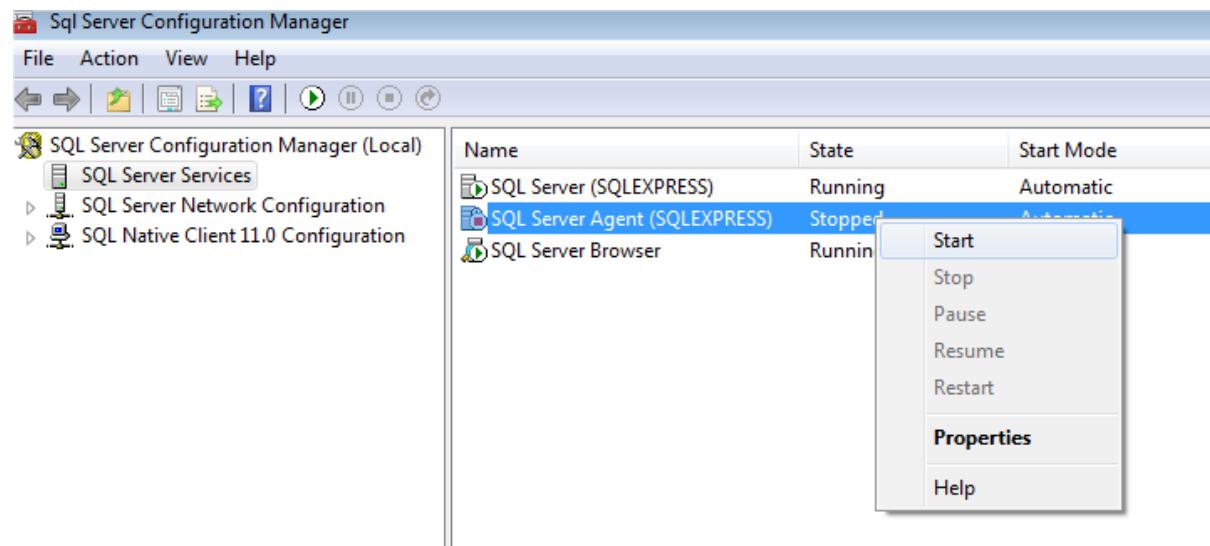


If you DON'T see this then you need to do the following.

Go to the SQL Server Configuration Manager back on your Start Menu bar



Right Click on the Server Agent and start it. Turn on the Server Agent.



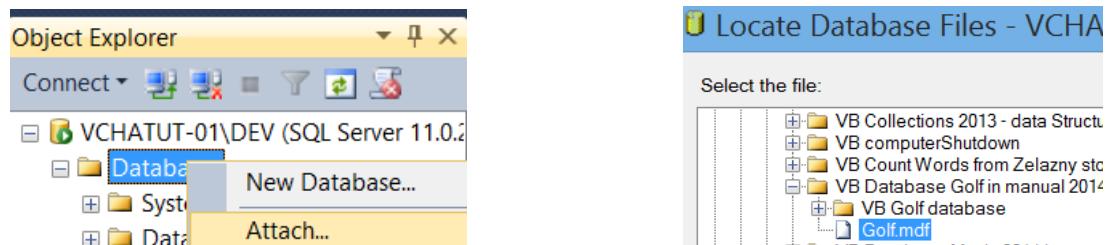
Hopefully that will get you going.

If you see nothing in that window, you are really screwed, the wrong version of SQL Management Studio has been installed and the Server part is missing. Uninstall your version and get the version mentioned earlier.

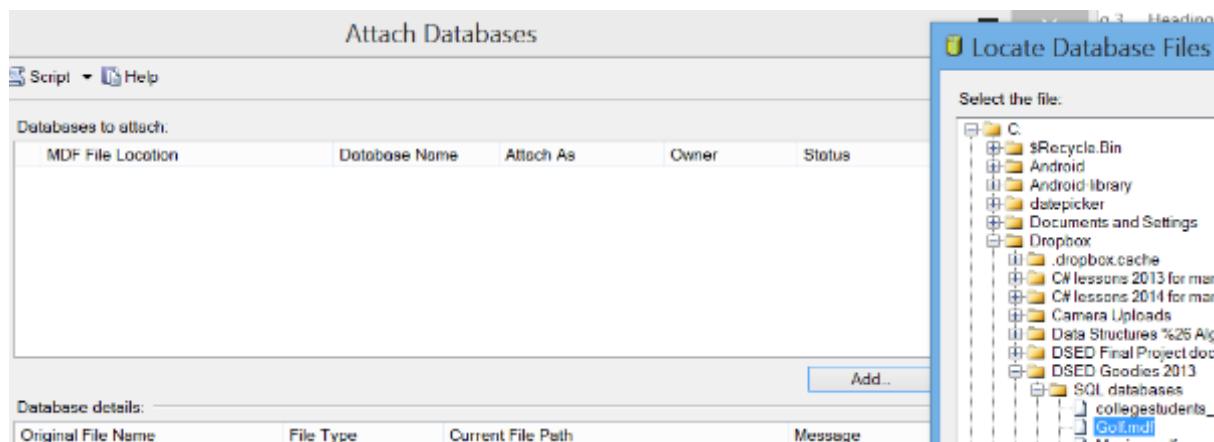
How to Attach an existing Database

I put my databases in the same folder as my project.
So move it there first or do it later.

Right Click on the Database folder and then Choose **Attach**.



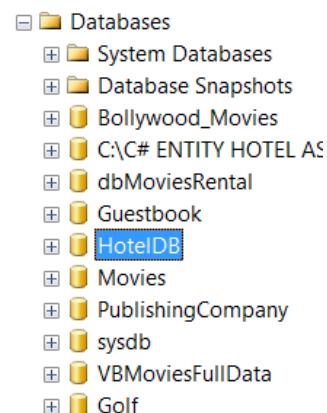
Click **Add** and find your Database.



The program will also look for the Log file, but won't find it, as there is only the Main file. (That's not a problem), just click **Remove** to remove it. Then Click **OK** to connect it.

| 'Golf' database details: | | | |
|--------------------------|-----------|---|-----------|
| Original File Name | File Type | Current File Path | Message |
| Golf.mdf | Data | C:\Dropbox\DSED Goodies 2013\SQL ... | ... |
| Golf_log.ldf | Log | c:\Program Files\Microsoft SQL Server\... | Not Found |

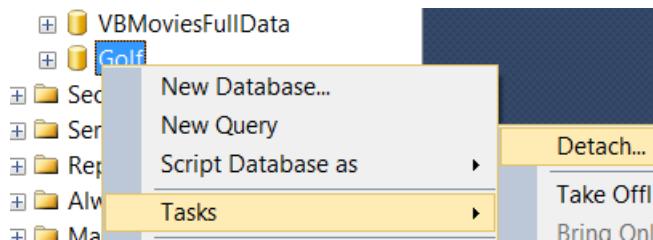
The Database should be Attached, although you might have to right click and choose Refresh on the Database folder to see it.



How to Remove a Database from SQLSMS

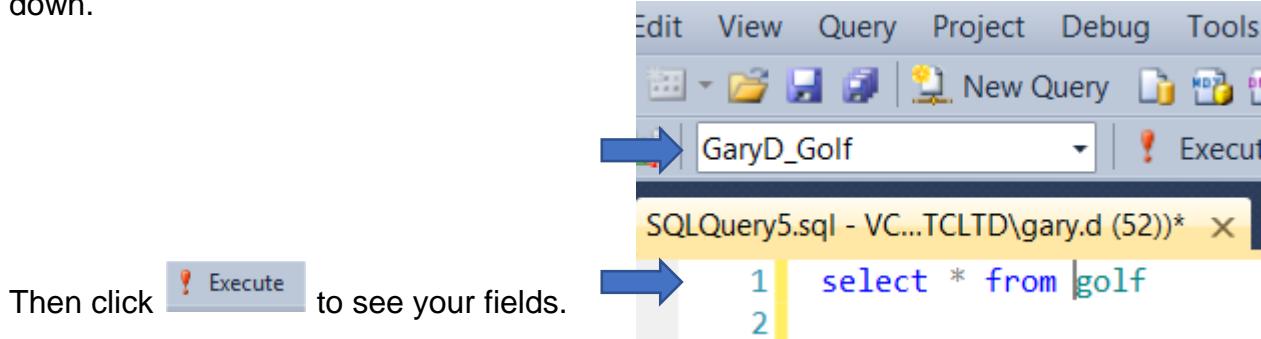
To remove a Database from the Server **Right click** on it, Go **Tasks**, then **Detach**. This doesn't delete the database, it's still in its folder, it just removes it from the server.

YOU WILL NEED TO DO THIS IF YOU WANT TO COPY OR MOVE IT TO ANOTHER PLACE.



How to Run a Query in SQL Express Management Studio

Click on New Query and in the window below type **SELECT * FROM golf**. This will select all the fields from the Golf table. Make sure you choose the Golf Database in the pull down.



You should have “Query Executed Successfully” and 32 rows at the bottom of the screen

| ID | Surname | Firstname | Title | Gender | DOB | Street | Suburb | City | Available week days |
|----|------------|---------------|-------|--------|-------------------------|-----------------------|--------------|--------------|---------------------|
| 1 | Green | Jane | Mrs | F | 1973-04-14 00:00:00.000 | 432 River Road | Avonhead | Christchurch | 1 |
| 2 | Birch | Michael | Mr | M | 1966-08-18 00:00:00.000 | 78 Pinewood Avenue | Avonhead | Akaroa | 0 |
| 3 | Sharpe | Mathew James | Mr | M | 1981-08-14 00:00:00.000 | 123 Clarence Street | Spreydon | Christchurch | 0 |
| 4 | Lee | Kim Soon | Mrs | F | 1931-08-27 00:00:00.000 | 67 Meadow View Street | Avonhead | Christchurch | 1 |
| 5 | Birdwood | Reginald Ross | Mr | M | 1968-08-17 00:00:00.000 | 67 Bay Road | Merivale | Christchurch | 1 |
| 6 | Taylor | Rachel | Ms | F | 1969-08-17 00:00:00.000 | 56 English Street | Merivale | Christchurch | 0 |
| 7 | Thompson | Jeanette Jane | Ms | F | 1961-08-19 00:00:00.000 | 678 Highpark Road | Avonhead | Rangiora | 1 |
| 8 | Manakau | Rangi | Mr | M | 1966-08-18 00:00:00.000 | 365 Memorial Avenue | Merivale | Christchurch | 0 |
| 9 | Hayes | Linda Judith | Miss | F | 1987-08-13 00:00:00.000 | 567 Seaview Road | New Brighton | Christchurch | 0 |
| 10 | Drinkwater | Sarah-Jane | Mrs | F | 1924-08-28 00:00:00.000 | 4 Brickworks Road | Avonhead | Christchurch | 1 |
| 11 | Johnstone | Brian Bruce | Mr | M | 1970-08-17 00:00:00.000 | 454 Madras Street | city | Christchurch | 1 |
| 12 | Robertson | Melissa Jane | Ms | F | 1963-08-19 00:00:00.000 | 560 Valley Road | Heathcote | Christchurch | 1 |

Query executed successfully. | DESKTOP-PC\SQLEXPRESS (10.0... | desktop-PC\desktop (55) | golf | 00:00:00 | 32 rows

Right click on the Database then rename it to **Golf**

Owing to issues with the dates later on, set the DOB field to the Nvarchar(50) or smaller

| Column Name | Data Type | AI |
|-------------|--------------|----|
| ID | int | |
| Surname | nvarchar(50) | |
| Firstname | nvarchar(50) | |
| Title | nvarchar(50) | |
| Gender | nvarchar(50) | |
| DOB | nvarchar(50) | |
| Street | nvarchar(50) | |

[Set the ID field to be Auto Incrementing.](#)

Although we will cover it later in the next exercise, you could also set the ID to be a

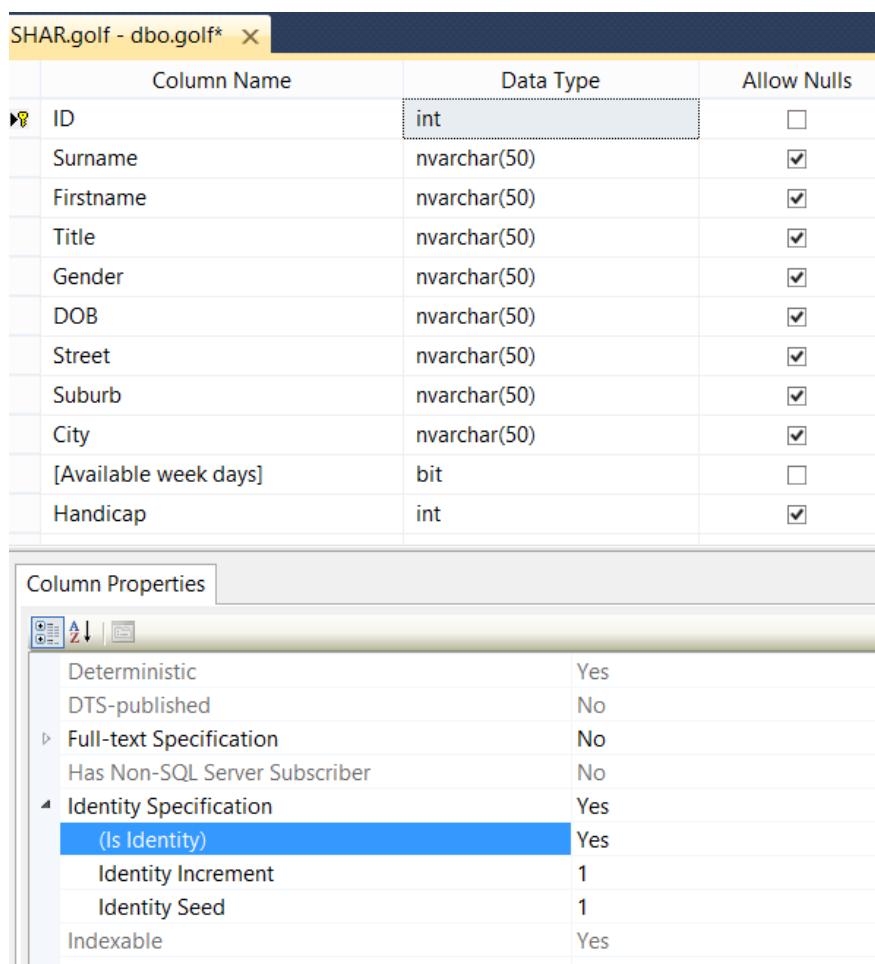
Primary Key  with the Identity Specification set to **Yes**. This means that you won't have to set the ID any more as each new entry will automatically have a new one.

The drawback with this is if you try to pass data to the ID field you will get an error.

However if you do add it, and why not, then you **don't have to give the Primary Key a value** each time you add a new row of data to the table.

Right click on the ID Field and choose **Modify**.

Set **Is Identity** to Yes.

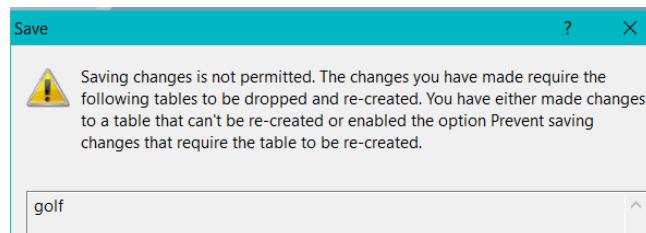


The screenshot shows the 'golf' table structure in SQL Server Management Studio. The table has 10 columns:

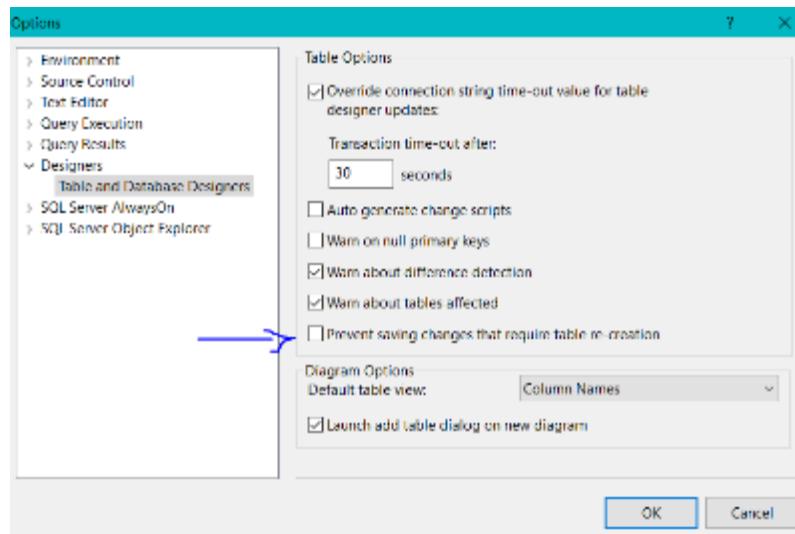
| Column Name | Data Type | Allow Nulls |
|-----------------------|--------------|-------------------------------------|
| ID | int | <input type="checkbox"/> |
| Surname | nvarchar(50) | <input checked="" type="checkbox"/> |
| Firstname | nvarchar(50) | <input checked="" type="checkbox"/> |
| Title | nvarchar(50) | <input checked="" type="checkbox"/> |
| Gender | nvarchar(50) | <input checked="" type="checkbox"/> |
| DOB | nvarchar(50) | <input checked="" type="checkbox"/> |
| Street | nvarchar(50) | <input checked="" type="checkbox"/> |
| Suburb | nvarchar(50) | <input checked="" type="checkbox"/> |
| City | nvarchar(50) | <input checked="" type="checkbox"/> |
| [Available week days] | bit | <input type="checkbox"/> |
| Handicap | int | <input checked="" type="checkbox"/> |

In the 'Column Properties' pane, under 'Identity Specification', the '(Is Identity)' checkbox is selected (highlighted in blue), and the 'Identity Increment' is set to 1.

When you try to save it however you get an error so cancel out of the error and change the following.



To fix this go **Tools / Options** and **Table and Database Designers**.



Untick the field above and click OK.

Now you can save your Table.

How to Run an SQL Script to create a Database (Reference)

If you have an SQL script you can drag it into the SQL window to load it.

Find the script **Golf_Script.sql** and drag it into the window.

Here is the script when loaded in the program. You can scroll through it and see the data that is being held.

This is a wonderful way to move small databases around, by scripting them out and then running the script.

```

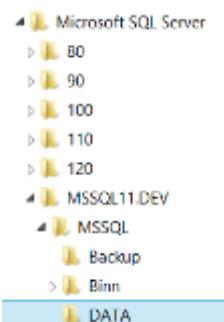
1 USE [master]
2 GO
3 /****** Object: Database [GaryD_GolfFromScript] Script Date: 26/03/2014 1:35:05 p.m. *****/
4 CREATE DATABASE [GaryD_Golf] ON PRIMARY
5 ( NAME = N'GaryD_Golf', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL11.DEV\MSSQL\DATA
6 \GaryD_Golf.mdf' , SIZE = 5000KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
7 LOG ON
8 ( NAME = N'GaryD_Golf_log', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL11.DEV\MSSQL\DATA
9 \GaryD_Golf_log.ldf' , SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
10 GO
11 ALTER DATABASE [GaryD_Golf] SET COMPATIBILITY_LEVEL = 100

```

However before you can run your Script you have to **change the path to where you store your Databases.**

The path shown in the script is the path to the where that database was created. It will be different to your installed path as the server name at least can be different.

Here is the one from above **N'C:\Program Files\Microsoft SQL Server\MSSQL11.DEV\MSSQL\DATA\GaryD_Golf.mdf'**



Databases are by default stored in the data folder in the folder with the name of your Server. Mine is MSSQL11.DEV so the path is actually the same however you need to replace it with the path to YOUR data file. The big difference might be just the name of the server.

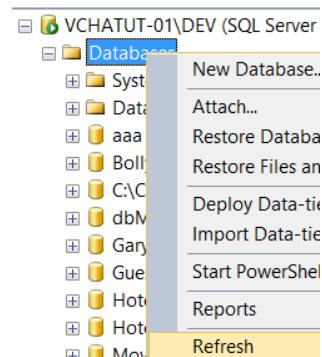
Once you have changed that, then run the Execute and see if the Database is created.

If successful then right click on Database and click Refresh to see your new database. If you look in your Data folder you will find it stored there.

Here is a YouTube video on how to do it as well.

<https://www.youtube.com/watch?v=olgJOG70-vg>

Check that your data has arrived safely by running a simple SQL query



Why Can't SQL Server Management Studio Access The "My Documents" Folder

This seems to be a problem with some students machines.

<http://superuser.com/questions/69879/why-cant-sql-server-management-studio-access-the-my-documents-folder-in-windo>

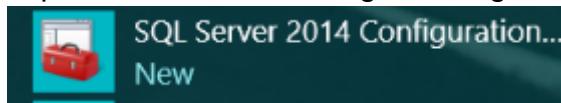
<http://bidn.com/Blogs/unable-to-browse-to-database-or-backup-file-location-from-ssms>

While the folder that contains my backup file is located within the folder sharrison, it appears as if there is not.

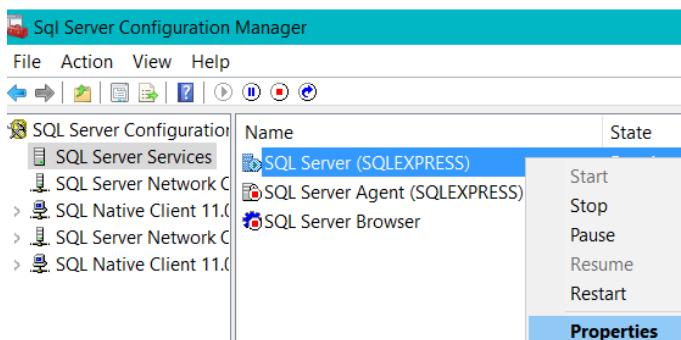
This is because the SQL Server account does not have permission to reach the folder you are trying to browse.

To give your SQL Server account access to this folder, we first have to verify the account used to login to SQL Server.

Open SQL Server Manager Configuration Manager and navigate to SQL Server Services.



Right-click SQL Server (MSSQLSERVER) and select properties.



On the Log On tab, you should see where the account to login as is specified.



NT Service\iervice\MSSQL\$SQLEXPRESS

Once you know the account being used by SQL Server, give that account the needed permissions on your directory.

21. The Golf Project using ADO.net

What is ADO.Net overview

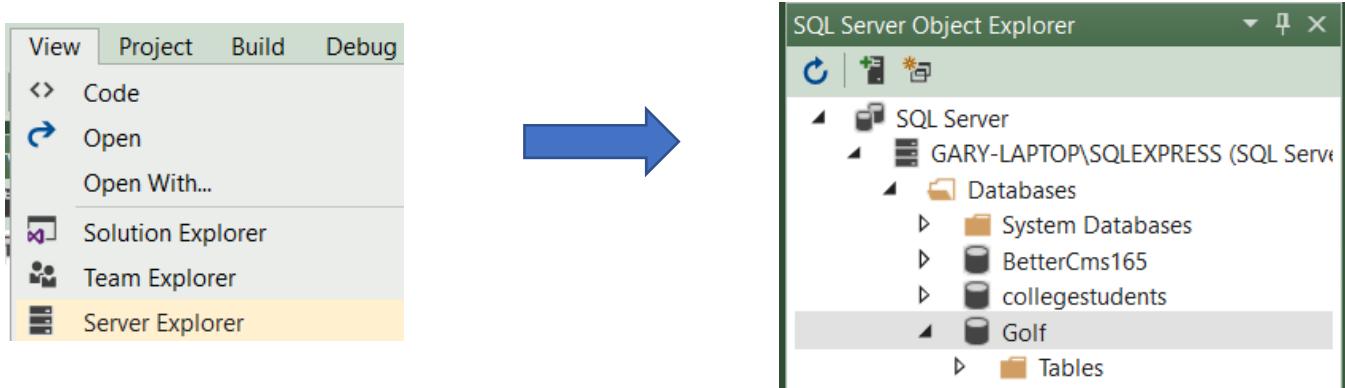
[Learn ADO.net](#) Read this!

In Visual Studio create a new project called Golf. From the earlier examples attach the Golf DB to your SQL Server. Below we then join the two together via a connection string.

[How to find your Connection String.](#)

Connection strings can be a pain in the neck, but there is an easy way to find out what your one is to connect to the Database you want.

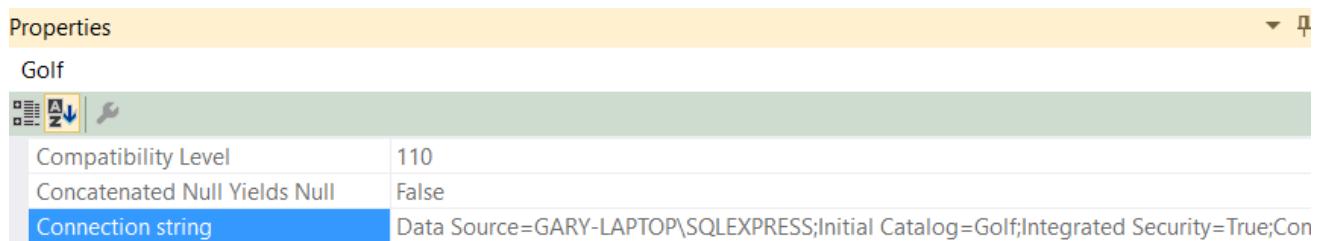
Open Server Explorer in Visual Studio go **View / Server Explorer**



Find your Database, right click and go to Properties.

Then just look for the Connection string, Copy and Paste the first bit into your program.

```
//Data Source=GARY-LAPTOP\SQLEXPRESS;Initial Catalog=Golf;Integrated Security=True;
```



In your code later you will add in your Connection String

```
string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial Catalog=golf;Integrated Security=True";
```

The @ symbol tells the compiler that the following string is a string literal. Which means the \ won't be seen as an escape character, but as a part of the string.

The **Data Source** key indicates which server to access. In this case, the **GARY-LAPTOP\SQLEXPRESS** value refers to the SQL Server Express Edition installation on the local workstation.

The **Initial Catalog=golf** key tells the connection which database within the hosted database engine to use as the default. In this sample string, **golf** is the name of the default database catalog to use. You must have the appropriate security credentials to access this database.

The **Integrated Security=True** key with a value of True tells ADO.NET to use your existing Microsoft Windows security credentials to access the database

Display data in a ListView

Let's create a **Listview** to load the data in so that we know everything is working OK.

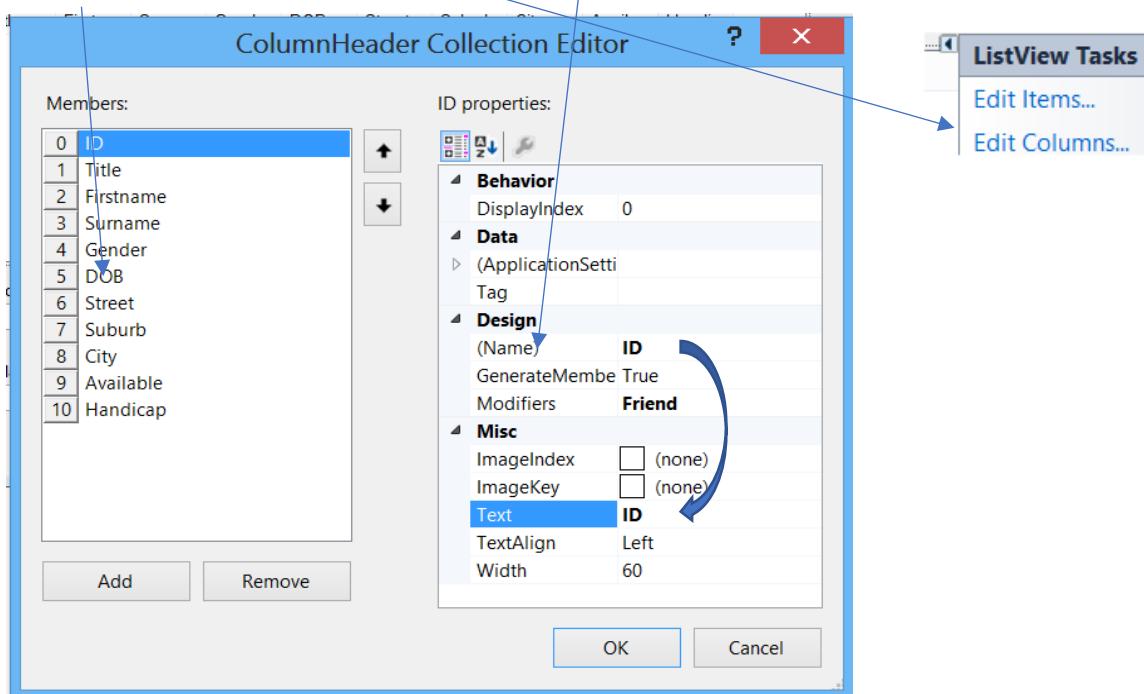
The Listview in **final project**, on the right, with the DataGrid on the left. We just want the Listview at this stage.

| ID | Title | Firstname | Surname | Gender | DOB | Street | Suburb | City | Available | Handicap |
|----|-------|-----------|----------|--------|-----------|-----------|----------|-----------|-----------|----------|
| 1 | Mrs | Jane | Green | F | 14/04/... | 432 Ri... | Avonh... | Christ... | True | 12 |
| 2 | Mr | Michael | Birch | M | 18/08/... | 78 Pin... | Avonh... | Akaroa | False | 19 |
| 3 | Mr | Mathew | Sharpe | M | 14/08/... | 123 Cl... | Sprey... | Christ... | False | 6 |
| 4 | Mrs | Kim S... | Lee | F | 27/08/... | 67 Me... | Avonh... | Christ... | True | 30 |
| 5 | Mr | Reginald | Birdw... | M | 17/08/... | 67 Ba... | Meriv... | Christ... | True | 31 |
| 6 | Ms | Rachel | Taylor | F | 17/08/... | 56 En... | Meriv... | Christ... | False | 0 |
| 7 | Ms | Jeane... | Thom... | F | 19/08/... | 678 Hi... | Avonh... | Rangi... | True | 6 |
| 8 | Mr | Rangi | Mana... | M | 18/08/... | 365 M... | Meriv... | Christ... | False | 12 |
| 9 | Miss | Linda ... | Hayes | F | 13/08/... | 567 S... | New B... | Christ... | False | 17 |

From the Toolbox drag on a ListView and name it **LVGolf**

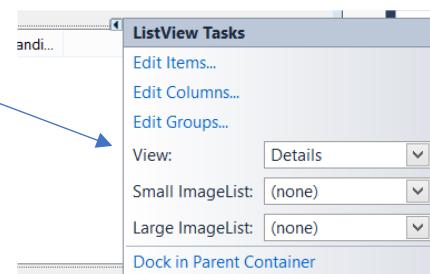
Create the Headings for the Listview.

In the **Edit Columns Option** name 10 columns with the appropriate names as shown below. I changed the TEXT property and the NAME property to the same.



In the **View Property** set it to **Details**

We need to add the code to read the data into a Listview.



Three things we need to do.

```
Using (SqlConnection linkToDB = new SqlConnection(ConnectionString)) {
linkToDB.Open();
// -----Do stuff with the data you get here.
}
```

1 Make a connection with the Database on the server by passing in the connection string to establish the connection.

This connection is wrapped in a `using` Statement, so that when the data has finished being moved, the connection is closed, and any unwanted resources are deleted.

```
using (SqlConnection connection = new SqlConnection(connectionString)) {

string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial Catalog=golf;Integrated Security=True";
```

2 We need send a command to the server. In this case the command is send us back all the data in the golf table.

```
string queryString = "SELECT * FROM Golf ORDER by ID";

//read in the data with a datareader open the data connection
SqlCommand Command = new SqlCommand(queryString, connection);
Down the pipe we have made we open the connection, Connection.Open();
```

3 Send a request to the database for data. Do stuff with it.

Get that data back and close the connection. `Connection.Close();`

The Code for Golf

(Really 90% of the following should be in a class, but that's in the future)

The columns are really pseudo columns. Although we have a whole bunch of columns you can't just specify them in a Listview as it only really holds one item down the page.

The code below declares two objects, one for the Command and one for the DataReader.

The Command object holds and executes the SELECT statement to send to the data source. `SqlCommand Command = new SqlCommand(queryString, connection);`

The DataReader is the object that retrieves the data from the result set that comes back from the SELECT statement. `SqlDataReader reader = Command.ExecuteReader();`

Using the ADO.NET DataReader

DataSets, DataTables, and DataReaders are used to retrieve records from data sources.

The DataReader object is a forward-only type of cursor that provides the fastest way to retrieve records from a data source.

Because its direction is limited to forward-only, it provides great performance for programmatically processing results or loading list boxes, combo boxes, etc.

You loop through each of the rows in the DataReader by invoking the Read method.
`reader.Read()`.

The Read method moves the cursor from one row to the next. After the Read method has executed, you can pass the name of the column you wish to retrieve to the Item property on the DataReader. This returns the actual data from that column.

The data returned from the Item property comes back as an Object data type. Because the items you add to the list box are of type Object, no conversion is required when using the Add method of the Items collection on the list box.

In this manner, you continue looping until the Read method returns a False. This means you have hit the end of the rows of data that were returned from the SELECT statement.

```
public void loadDatabase() {
    lvGolf.Items.Clear(); //clear out old data in the listbox, we will need this later
        //load the connection string and pass it to the command
        string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial Catalog=golf;Integrated Security=True";

    using (SqlConnection connection = new SqlConnection(connectionString)) {
        string queryString = "SELECT * FROM Golf ORDER by ID";

        //read in the data with a datareader open the data connection
        SqlCommand Command = new SqlCommand(queryString, connection);
            connection.Open();
            SqlDataReader reader = Command.ExecuteReader();
            while (reader.Read()) {
                //add each row to the listbox
                ListViewItem item = new ListViewItem(new[] {
                    reader["ID"].ToString(), reader["Title"].ToString(),
                    reader["Firstname"].ToString(), reader["Surname"].ToString(),
                    reader["Gender"].ToString(), reader["DOB"].ToString(), reader["Street"].ToString(),
                    reader["Suburb"].ToString(), reader["City"].ToString(), reader["Available week days"].ToString(),
                    reader["Handicap"].ToString()
                });
                LVGolf.Items.Add(item);
            }
            reader.Close();
    }
}
```

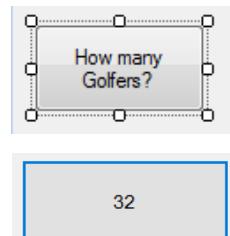
Because the columns in the ListView are not 'real' they only **look** like columns, you can't operate on them as columns, you can't click on them, or select a column, they are pretty useless as tools but just display data.

Run and see the data load.

| ID | Title | Firstn... | Surna... | Gender | DOB | Street |
|----|-------|-----------|----------|--------|-----------|---------|
| 3 | Mr | Mathe... | Sharpe | M | 14/08/... | 123 Cl |
| 4 | Mrs | Kim S... | Lee | F | 27/08/... | 67 Ma |
| 5 | Mr | Regin... | Birdw... | M | 17/08/... | 67 Ba. |
| 6 | Ms | Rachel | Taylor | F | 17/08/... | 56 En. |
| 7 | Ms | Jeane... | Thom... | F | 19/08/... | 678 Hi |
| 8 | Mr | Rangi | Mana... | M | 18/08/... | 365 M. |
| 9 | Miss | Linda ... | Hayes | F | 13/08/... | 567 S. |
| 10 | Mrs | Sarah... | Drink... | F | 28/08/... | 4 Bric. |

Returning a Scalar (single) value

Create a new button. Call it btnCount



You create Scalar queries to return a **single value**. In this example we will use a simple COUNT to show **on the button** how many Golfers there are.

- Create a **SqlConnection** object and give it a **connection string**.
`SqlConnection Con = new SqlConnection();`
- Create a **SqlCommand** object and assign to it the connection object you opened. Place an SQL statement into the Command object `SqlCommand update = new SqlCommand(updatestatement, Con)`
- Open the connection to the database. `connection.Open();`
- Execute the SQL statement by invoking the **ExecuteNonQuery** method on the Command object. `update.ExecuteNonQuery();`
- Close the Connection. `connection.Close();`

```
private void btnCount_Click(object sender, EventArgs e) {
    string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial Catalog=golf;Integrated Security=True";
    // a simple Scalar query just returning one value.
    string queryString = "SELECT COUNT(ID) FROM Golf";

    using (SqlConnection connection = new SqlConnection(connectionString)) {
        SqlCommand Command = new SqlCommand(queryString, connection);
        connection.Open();
        btnCount.Text = Command.ExecuteScalar().ToString();
        connection.Close();
    }
}
```

32

Button will show 32 on the text field

Insert /New Command



The **Insert command** has three main areas, the **Textboxes** `txtFirstname.Text` where you type in the data, the **Parameters** you pass the data to, and the **Database fields you want the data to go to..**



Using Parameters

Adding a Parameter between the Textbox and the Database gives an extra layer of protection for your database.

We could just add Data from the Textbox and pass it directly to the SQL, but that's nasty and leaves you open to SQL attacks.

With parameters you don't need to know the name of the textbox in the SQL command, it's much cleaner although can be more tedious to write. .

Placeholders, as mentioned, always start with an @ symbol `@Title`. They do not need to be named after the database column that they represent, but it is often easier if they are, and it helps to self-document your code.

The term *parameters* here refers to the parameters required to **provide data to your SQL statement** or stored procedure.

Using Parameters to hold the data we can create text boxes that will allow us to input our data to our database. In this example data from the `txtTitle` textbox is passed to the `@Title` parameter, then the `@Title` passes to the SQL `Title` command.

```
newdata.Parameters.AddWithValue("@Title", txttitle.Text)

string NewEntry = "INSERT INTO Golf (Title, Firstname, Surname, Gender, DOB,
Street, Suburb, City, [Available week days], Handicap) VALUES ('@Title',
@Firstname, @Surname, @Gender, @DOB, @Street, @Suburb, @City, @Available,
@Handicap');
```

A diagram illustrating the mapping of parameters. A blue curved arrow originates from the `txttitle.Text` part of the `AddWithValue` method call and points to the `@Title` placeholder in the SQL `VALUES` clause. Another blue curved arrow originates from the `txttitle.Text` part of the `AddWithValue` method call and points to the `txttitle.Text` part of the `.AddWithValue` method call.

However using Parameters `cmd.Parameters.AddWithValue("@Parameter", txtTextBox1.Text)`; does not solve everything because the value inserted isn't restricted to a type, it goes in as an **object**.

If you want a Date make sure that ONLY a Date can be added in.

`cmd.Parameters.AddWithValue("@Parameter", SqlDbType.DateTime).Value = MyDateTimeVariable;`
 Read this article here, and how to specify the Type so that errors, and attacks are prevented. [Can we stop using AddWithValue\(\) already?](#)

The **AddWithValue** method here accepts the name of the parameter and the object that you want to add.

Create the extra textboxes and their label names below. Get the names from the next section of code.

The insert command inserts the data into the table at the last row.

| | | | | | |
|---------------------------------|--------------------------------------|-------------------------------|-----------------------------------|---------------------------|--|
| ID | Title | Ffirstname | Surname | Gender | |
| <input type="text"/> | Mr | <input type="text"/> Howard | <input type="text"/> TheDuck | <input type="text"/> M | <input type="button" value="New Entry"/> |
| DOB | Street | Suburb | City | Available | Handicap |
| <input type="text"/> 10/10/1969 | <input type="text"/> 12 Quack Street | <input type="text"/> Spreydon | <input type="text"/> Christchurch | <input type="text"/> True | <input type="text"/> 12 |

No Id field is needed as it is added automatically.

Note in the code that [] go around names that have spaces in them [Available week days].

Here is the main piece of code. Adding = "James"; to the end gives you some default data so you don't have to type into the boxes every time.

`newdata.Parameters.AddWithValue("@Firstname", txtFirstname.Text).Value = "James";`
 Delete out the `.Value = "James"`; and the other values when you get your code working.

```
private void btnNew_Click(object sender, EventArgs e) {
    // this puts the parameters into the code so that the data in the text boxes is added to the
    // database
    string NewEntry = "INSERT INTO Golf (Title, Ffirstname, Surname, Gender, DOB, Street, Suburb, City, [Available
    week days], Handicap) VALUES ( @Title, @Firstname, @Surname, @Gender, @DOB, @Street, @Suburb, @City, @Available,
    @Handicap)";
    SqlConnection Con = new SqlConnection();
    string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial Catalog=golf;Integrated
    Security=True";
    Con.ConnectionString = connectionString;
    using (SqlCommand newdata = new SqlCommand(NewEntry, Con)) {

        newdata.Parameters.AddWithValue("@Title", txttitle.Text).Value = "Mr";
        newdata.Parameters.AddWithValue("@Firstname", txtFirstname.Text).Value = "James";
        newdata.Parameters.AddWithValue("@Surname", txtSurname.Text).Value = "Bond";
        newdata.Parameters.AddWithValue("@Street", txtStreet.Text).Value = "123 Bond Street";
        newdata.Parameters.AddWithValue("@Suburb", txtSuburb.Text).Value = "Merivale";
        newdata.Parameters.AddWithValue("@City", txtCity.Text).Value = "Christchurch";
        newdata.Parameters.AddWithValue("@Gender", txtGender.Text).Value = "M";
        newdata.Parameters.AddWithValue("@DOB", txtDOB.Text).Value = "1/2/1935";
        newdata.Parameters.AddWithValue("@Handicap", txtHandicap.Text).Value = "2";
        newdata.Parameters.AddWithValue("@Available", txtAvailable.Text).Value = "";

        Con.Open(); //open a connection to the database
        //its a NONQuery as it doesn't return any data its only going up to the server
    }
}
```

```
    newdata.ExecuteNonQuery(); //Run the Query  
    //a happy message box  
    MessageBox.Show("Data has been Inserted !! ");  
}  
//Run the LoadDatabase method we made earlier to see the new data.  
loadDatabase();  
}
```

There is so much that can be optimised, but to date I have left it untouched to keep it easier to follow.

We are repeating heaps of code under each button that can be globalised, such as

```
SqlConnection Con = new SqlConnection();  
string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial  
Catalog=golf;Integrated Security=True";  
Con.ConnectionString = connectionString;
```

So this should be looking familiar for each Method we make. We should only have one connection string for the entire program, to make maintenance easier.

Update Command

Update is similar to Insert. It changes the data and uploads it over the existing data.

To modify data within a database

- Create a **SqlConnection** object and give it a **connection string**. **SqlConnection Con = new SqlConnection();**
- Create a **SqlCommand** object and assign to it the connection object you opened. Place an SQL statement into the Command object. **SqlCommand update = new SqlCommand(updatestatement, Con)**
- Open the connection to the database. **Con.Open();**
- Execute the SQL statement by invoking the **ExecuteNonQuery** method on the Command object. **update.ExecuteNonQuery();**
- Close the Connection. **Con.Close();**

```
private void btnUpdate_Click(object sender, EventArgs e) {
    //this updates existing data in the database where the ID of the data equals the ID in
    //the text box

    string updatestatement = "UPDATE Golf set Title=@Title, Firstname=@Firstname,
    Surname=@Surname, Gender=@Gender, DOB=@DOB, Street=@Street, Suburb=@Suburb, City=@City,
    [Available week days]=@Available, Handicap=@Handicap where ID = @ID";

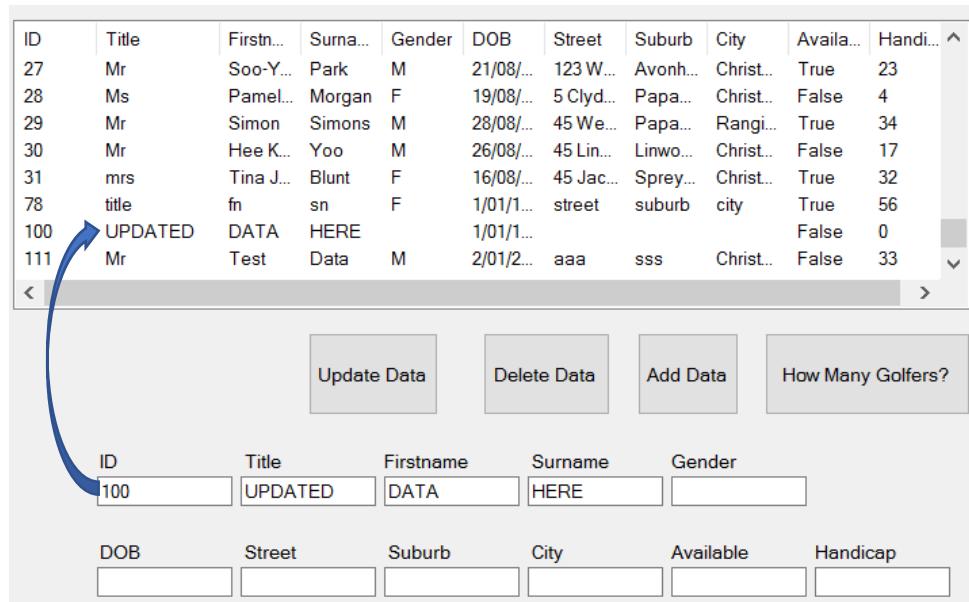
    SqlConnection Con = new SqlConnection();
    string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial
    Catalog=golf;Integrated Security=True";
    Con.ConnectionString = connectionString;

    SqlCommand update = new SqlCommand(updatestatement, Con)
    //create the parameters and pass the data from the textboxes

    update.Parameters.AddWithValue("@ID", txtID.Text);
    update.Parameters.AddWithValue("@Title", txtTitle.Text);
    update.Parameters.AddWithValue("@Firstname", txtFirstname.Text);
    update.Parameters.AddWithValue("@Surname", txtSurname.Text);
    update.Parameters.AddWithValue("@Street", txtStreet.Text);
    update.Parameters.AddWithValue("@Suburb", txtSuburb.Text);
    update.Parameters.AddWithValue("@City", txtCity.Text);
    update.Parameters.AddWithValue("@Gender", txtGender.Text);
    update.Parameters.AddWithValue("@DOB", txtDOB.Text);
    update.Parameters.AddWithValue("@Handicap", txtHandicap.Text);
    update.Parameters.AddWithValue("@Available", txtAvailable.Text);

    Con.Open();
    //its NONQuery as data is only going up
    update.ExecuteNonQuery();
    Con.Close();
    loadDatabase();
}
```

My example below shows #100 changing with the update command.



| ID | Title | Firstn... | Surna... | Gender | DOB | Street | Suburb | City | Availa... | Handi... |
|-----|---------|-----------|----------|--------|-----------|-----------|----------|-----------|-----------|----------|
| 27 | Mr | Soo-Y... | Park | M | 21/08/... | 123 W... | Avonh... | Christ... | True | 23 |
| 28 | Ms | Pamel... | Morgan | F | 19/08/... | 5 Clyd... | Papa... | Christ... | False | 4 |
| 29 | Mr | Simon | Simons | M | 28/08/... | 45 We... | Papa... | Rangi... | True | 34 |
| 30 | Mr | Hee K... | Yoo | M | 26/08/... | 45 Lin... | Linwo... | Christ... | False | 17 |
| 31 | mrs | Tina J... | Blunt | F | 16/08/... | 45 Jac... | Sprey... | Christ... | True | 32 |
| 78 | title | fn | sn | F | 1/01/1... | street | suburb | city | True | 56 |
| 100 | UPDATED | DATA | HERE | | 1/01/1... | | | | False | 0 |
| 111 | Mr | Test | Data | M | 2/01/2... | aaa | sss | Christ... | False | 33 |

| | | | | |
|----------------------------------|--------------------------------------|-----------------------------------|-----------------------------------|----------------------|
| ID | Title | Firstname | Surname | Gender |
| <input type="text" value="100"/> | <input type="text" value="UPDATED"/> | <input type="text" value="DATA"/> | <input type="text" value="HERE"/> | <input type="text"/> |
| DOB | Street | Suburb | City | Available |
| <input type="text"/> | <input type="text"/> | <input type="text"/> | <input type="text"/> | <input type="text"/> |
| Handicap | | | | |

Delete Command

Delete is easy to set up, all we have to do is tell the command to delete the entry where the ID equals one you choose. `string DeleteCommand = "Delete Golf where ID = @ID";`

```
private void btnDelete_Click(object sender, EventArgs e) {
    SqlConnection Con = new SqlConnection();
    string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial Catalog=golf;Integrated Security=True";
    Con.ConnectionString = connectionString;

    string DeleteCommand = "Delete Golf where ID = @ID";

    SqlCommand DeleteData = new SqlCommand(DeleteCommand, Con);
    DeleteData.Parameters.AddWithValue("@ID", txtID.Text);

    Con.Open();
    DeleteData.ExecuteNonQuery();
    Con.Close();
    loadDatabase();
}
```

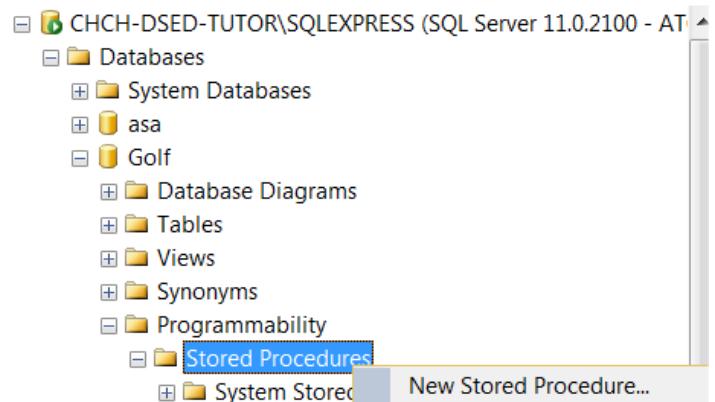
Stored Procedure - The SelectCommand Property

<https://msdn.microsoft.com/en-nz/library/ms345415.aspx>

Instead of writing SQL commands (Select, Insert, Update, Delete) you could instead execute a stored procedure - a group of SQL statements stored in the database under a unique name and executed as a unit.

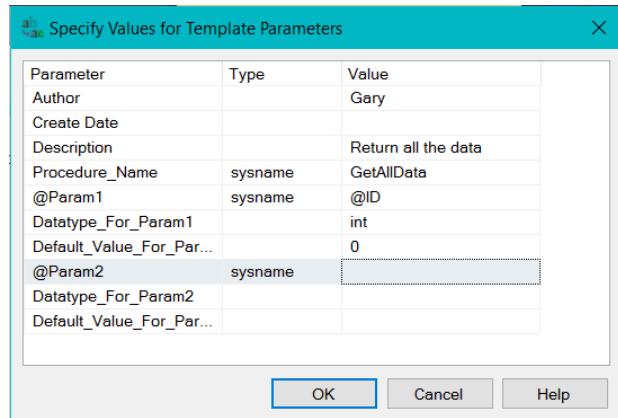
The benefits of Stored Procedures are reduced server/client network traffic, stronger security, reusable code, easier maintenance and improved performance.

Go to your SQL Management program and follow the path below. Right click on **Stored Procedures** and Choose **Stored Procedure**



On the Query menu, click Specify Values for Template Parameters.

In the Specify Values for Template Parameters dialog box, enter the following values for the parameters shown, with your name.



```
-- =====
-- Author:      Gary
-- Create date:
-- Description: Return all the data
-- =====
CREATE PROCEDURE GetAllData
    -- Add the parameters for the stored procedure here
    @ID int = 0,
    =
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

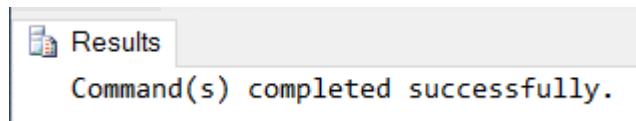
    -- Insert statements for procedure here
    SELECT @ID,
    =
END
GO
```

Click OK, you will see some errors, it wanted two parameters.

Tidy it up and add in the Query we want.

To test the syntax, on the Query menu, click Parse or click the tick

If an error message is returned, compare the statements with the information above and correct as needed.

 Results
Command(s) completed successfully.

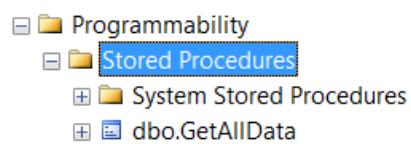
```
-- =====
-- Author:      Gary
-- Create date: 
-- Description: Return all the data
-- =====
CREATE PROCEDURE GetAllData
    -- Add the parameters for the stored procedure here
    @ID int = 0
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT *
    FROM golf
    WHERE ID = @ID
END
GO
```

To create the procedure, from the Query menu, click Execute or 

The procedure is created as an object in the database.

To see the procedure listed in Object Explorer, right-click Stored Procedures and select Refresh.



To run the procedure, in Object Explorer, right-click the stored procedure name and select Execute Stored Procedure.

In the Execute Procedure window, enter 20 as the value for the parameter @ID.

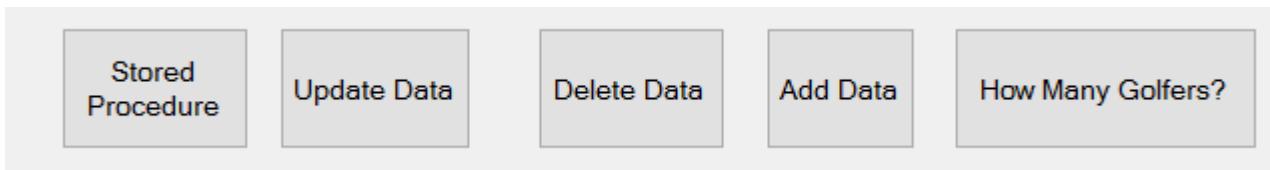
| Parameter | Data Type | Output Para... | Pass Null V... | Value |
|-----------|-----------|----------------|--------------------------|-------|
| @ID | int | No | <input type="checkbox"/> | 20 |

Output shows its working below.

| 100 % | | | | | | | | |
|---------|----------|-----------|-----------|--------|-----|-------------------------|-------------------|---------------------------|
| Results | | Messages | | | | | | |
| ID | Surna... | Firstname | Ti... | Gen... | DOB | Street | Suburb | City |
| 1 | 20 | Lassiter | Lucy Anne | Ms | F | 1982-08-14 00:00:00.000 | 76 Parkhouse Road | New Brighton Christchurch |

Calling the Stored procedure from your code

<https://msdn.microsoft.com/en-us/library/d7125bke.aspx>



Accessing a stored procedure is more verbose (but not more difficult) than accessing a normal SQL. The approach is as follows:

1. Create a SqlCommand object.
2. Configure it to access a stored procedure by setting the CommandType property.
3. Add parameters that exactly match those in the stored procedure itself.
4. Execute the stored procedure using one of the SqlCommand object's ExecuteX methods.

The CommandText property now specifies the name of the stored procedure that you want to execute instead of the SQL string that was specified in the previous example.

```
cmd.CommandText = "GetAllData";
```

Also notice the CommandType property. It is set to a value of **CommandType.StoredProcedure**, which indicates that the CommandText property contains the name of a stored procedure to be executed.

```
cmd.CommandType = CommandType.StoredProcedure;
```

```
private void btnStoredP_Click(object sender, EventArgs e) {
    string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial Catalog=golf;Integrated Security=True";
    using (SqlConnection Con = new SqlConnection(connectionString))
        using (SqlCommand cmd = Con.CreateCommand()) {
            cmd.CommandText = "GetAllData";
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@ID", txtID.Text);

            Con.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read()) {
                //add each row to the listbox
                ListViewItem item = new ListViewItem(new[]
                {
                    reader["ID"].ToString(), reader["Title"].ToString(), reader["Firstname"].ToString(),
                    reader["Surname"].ToString(), reader["Gender"].ToString(), reader["DOB"].ToString(),
                    reader["Street"].ToString(), reader["Suburb"].ToString(), reader["City"].ToString(),
                    reader["Available week days"].ToString(), reader["Handicap"].ToString()
                });
            }
        }
}
```

```
    LVGolf.Items.Add(item);
    //add names to text boxes just to show how it can be done
    txtTitle.Text = reader["Title"].ToString();
    txtFirstname.Text = reader["Firstname"].ToString();
    txtSurname.Text = reader["Surname"].ToString();
}
reader.Close();
Con.Close();
}
}
```

Here I have selected person #3. It has added that person to the bottom of the ListView (not shown), and also back into the text boxes.

| ID | Title | Firstn... | Surna... | Gender | DOB | Street | Suburb | City | Availa... | Handi... |
|----|-------|-----------|----------|--------|-----------|-----------|----------|-----------|-----------|----------|
| 0 | Mr | Joe | Smith | M | 2/01/2... | aaa | sss | Christ... | False | 33 |
| 3 | Mr | Mathe... | Sharpe | M | 14/08/... | 123 Cl... | Sprey... | Christ... | False | 6 |
| 4 | Mrs | Kim S... | Lee | F | 27/08/... | 67 Me... | Avonh... | Christ... | True | 30 |
| 5 | Mr | Regin... | Birdw... | M | 17/08/... | 67 Ba... | Meriv... | Christ... | True | 31 |
| 6 | Ms | Rachel | Taylor | F | 17/08/... | 56 En... | Meriv... | Christ... | False | 0 |
| 7 | Ms | Jeane... | Thom... | F | 19/08/... | 678 Hi... | Avonh... | Rangi... | True | 6 |
| 8 | Mr | Rangi | Mana... | M | 18/08/... | 365 M... | Meriv... | Christ... | False | 12 |
| 9 | Miss | Linda ... | Hayes | F | 13/08/... | 567 S... | New B... | Christ... | False | 17 |
| 10 | Mrs | Sarah... | Drink... | F | 28/08/... | 4 Bric... | Avonh... | Christ... | True | 17 |

Stored Procedure

Update Data

Delete Data

Add Data

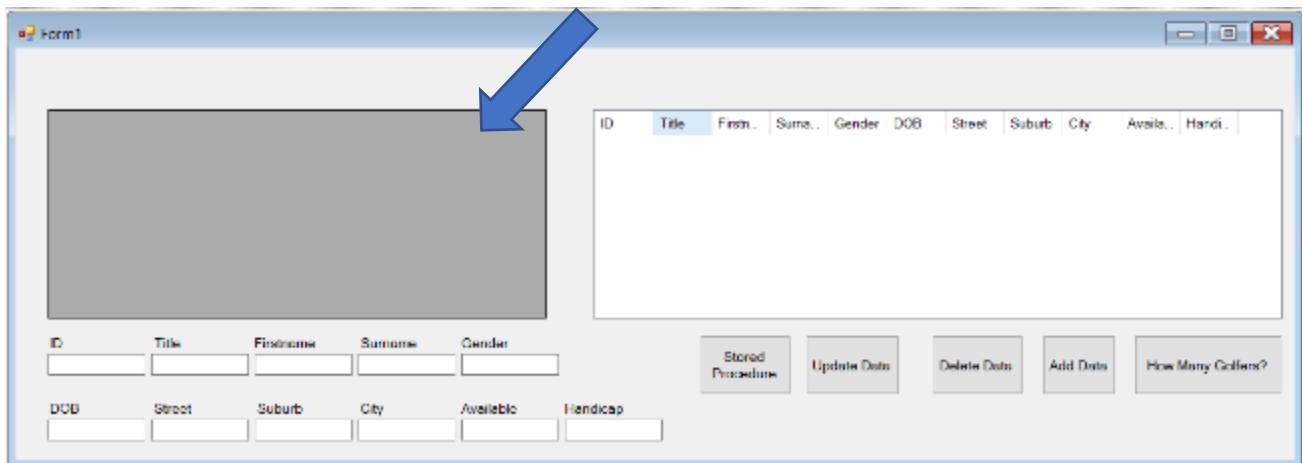
How Many Golfers?

| ID | Title | Firstname | Surname | Gender |
|----|-------|-----------|---------|--------|
| 3 | Mr | Mathew | James | Male |

| DOB | Street | Suburb | City | Available | Handicap |
|-----|--------|--------|------|-----------|----------|
| | | | | | |

Adding a DataGridView

Using a ListView might look nice, but really it's not very practical. Here is a DataGridView which we will add in. I also moved the textboxes to the left.



First we need to add the columns and the data to a **DataTable** named **GolfTable**

Add a **DataTable** **GolfTable = new DataTable();** to your program.

```
Public partial class Form1 : Form {
    //We need a connection to the Database
    SqlConnection Con = new SqlConnection();
    DataTable GolfTable = new DataTable();
//http://www.dotnetperls.com/datatable
```

Create the following sub that generates the table Column Titles

```
public void datatablecolumns() {
    //clear the old data
    GolfTable.Clear();
    //add in the column titles to the datatable
    try {
        GolfTable.Columns.Add("ID");
        GolfTable.Columns.Add("Title");
        GolfTable.Columns.Add("Firstname");
        GolfTable.Columns.Add("Surname");
        GolfTable.Columns.Add("Gender");
        GolfTable.Columns.Add("DOB");
        GolfTable.Columns.Add("Street");
        GolfTable.Columns.Add("Suburb");
        GolfTable.Columns.Add("City");
        GolfTable.Columns.Add("Available week days");
        GolfTable.Columns.Add("Handicap");
    } catch {
    }
}
```

Add the Method name to the LoadDB

```
private void loaddir(){  
    //load datatable columns  
    datatablecolumns();}
```

To get data into the DataGridView add the following in your existing LoadDB sub so that the reader reads the data in after the `Do While` `reader.Read()`

```
while (reader.Read()) {  
    //add in each row to the datatable  
    GolfTable.Rows.Add(reader["ID"], reader["Title"], reader["Firstname"],  
    reader["Surname"], reader["Gender"], reader["DOB"], reader["Street"],  
    reader["Suburb"], reader["City"], reader["Available week days"],  
    reader["Handicap"]);  
    //add the datatable to the Data Grid View
```

At the end pass the data in GolfTable to the DataGridView

```
reader.Close();  
    // connection.Close();  
    DGVgolf.DataSource = GolfTable;
```

[Click on a DGV row and fill the text boxes](#)

If you double click on the DataGridView it creates a method

```
private void DGVgolf_CellContentClick(object sender,
DataGridViewCellEventArgs e) {
}
```

This is a pain in the neck.

CellContentClick means it only runs when you click on any **content in the cell**, not when you click on the cell itself. So empty cells, or cells with only a little bit of data sometimes don't trigger the event. Then you spend hours wondering why the click event runs sometimes and doesn't run other times.

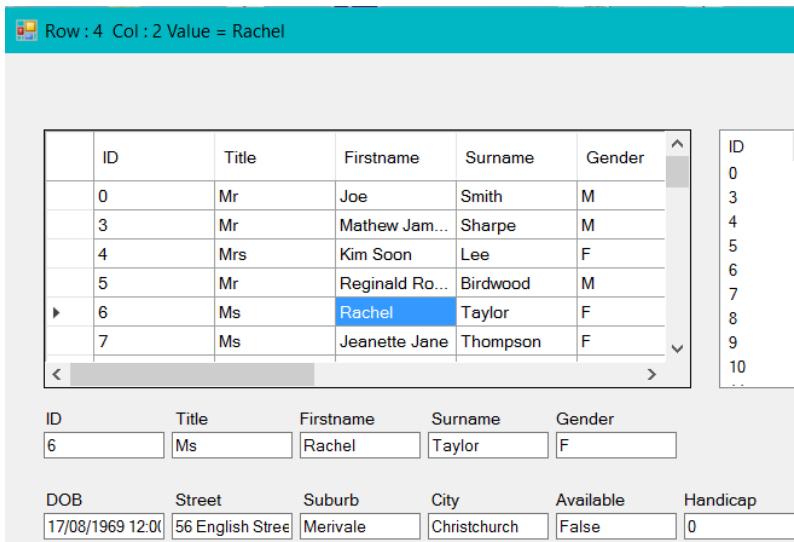
So under your Events in your Properties, change it to be the CellClick event.

| | |
|------------------------|--------------------------|
| CellBorderStyleChanged | |
| CellClick | DGVgolf_CellContentClick |
| CellContentClick | |

Using **CellClick** however does mean that when you click just outside the cell, in the header for example it triggers an error, there is no data it can pull out, so wrap it in a **try catch** to stop those errors.

When everything works you should be able to click on a row in your DataGridView and see the data in the Text Boxes.

```
private void DGVgolf_CellContentClick(object sender, DataGridViewCellEventArgs e) {
try {
    //show the data in the DGV in the text boxes
    string newvalue = DGVgolf.Rows[e.RowIndex].Cells[e.ColumnIndex].Value.ToString();
    //show the output on the header
    this.Text = "Row : " + e.RowIndex.ToString() + " Col : " +
e.ColumnIndex.ToString() + " Value = " + newvalue;
    //pass data to the text boxes
    txtID.Text = DGVgolf.Rows[e.RowIndex].Cells[0].Value.ToString();
    txtTitle.Text = DGVgolf.Rows[e.RowIndex].Cells[1].Value.ToString();
    txtFirstname.Text = DGVgolf.Rows[e.RowIndex].Cells[2].Value.ToString();
    txtSurname.Text = DGVgolf.Rows[e.RowIndex].Cells[3].Value.ToString();
    txtGender.Text = DGVgolf.Rows[e.RowIndex].Cells[4].Value.ToString();
    txtDOB.Text = DGVgolf.Rows[e.RowIndex].Cells[5].Value.ToString();
    txtStreet.Text = DGVgolf.Rows[e.RowIndex].Cells[6].Value.ToString();
    txtSuburb.Text = DGVgolf.Rows[e.RowIndex].Cells[7].Value.ToString();
    txtCity.Text = DGVgolf.Rows[e.RowIndex].Cells[8].Value.ToString();
    txtAvailable.Text = DGVgolf.Rows[e.RowIndex].Cells[9].Value.ToString();
    txtHandicap.Text = DGVgolf.Rows[e.RowIndex].Cells[10].Value.ToString();
} catch {
}
}
```



The screenshot shows a Windows application window. At the top left is a status bar with the text "Row : 4 Col : 2 Value = Rachel". Below it is a DataGridView containing a list of people with columns for ID, Title, Firstname, Surname, and Gender. A context menu is open over the cell at row 6, column 2 ("Rachel"). The menu items are: "Copy", "Cut", "Paste", "Delete", "Insert", "Select All", "Find", "Find Next", "Find Previous", "Sort Ascending", "Sort Descending", and "Sort By Column". To the right of the DataGridView is a vertical list of indices from 0 to 10. Below the DataGridView is a form with fields for ID, Title, Firstname, Surname, Gender, DOB, Street, Suburb, City, Available, and Handicap.

| ID | Title | Firstname | Surname | Gender |
|----|-------|----------------|----------|--------|
| 0 | Mr | Joe | Smith | M |
| 3 | Mr | Mathew Jam... | Sharpe | M |
| 4 | Mrs | Kim Soon | Lee | F |
| 5 | Mr | Reginald Ro... | Birdwood | M |
| 6 | Ms | Rachel | Taylor | F |
| 7 | Ms | Jeanette Jane | Thompson | F |

| | | | | |
|----|-------|-----------|---------|--------|
| ID | Title | Firstname | Surname | Gender |
| 6 | Ms | Rachel | Taylor | F |

| | | | | | |
|------------------|-------------------|----------|--------------|-----------|----------|
| DOB | Street | Suburb | City | Available | Handicap |
| 17/08/1969 12:01 | 56 English Street | Merivale | Christchurch | False | 0 |

You can even capture the **Row** `e.RowIndex` and **Column** `e.ColumnIndex` as well as the contents of the cell you click on. `DGVgolf.Rows[e.RowIndex].Cells[e.ColumnIndex].Value`

The following code allows you to extract out information such as the ID from your DataGridView and then use it to make another SQL call.

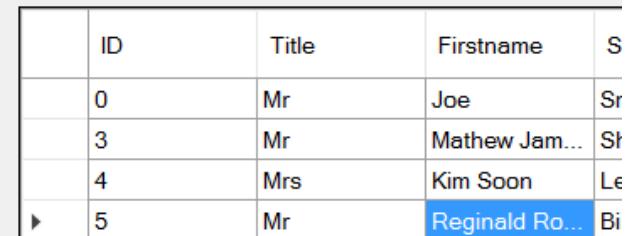
```
//show the output on the header
this.Text = "Row : " + e.RowIndex.ToString() + " Col : " +
e.ColumnIndex.ToString() + " Value = " + newvalue;
```

`e.RowIndex.ToString()` Row : 3

`e.ColumnIndex.ToString()` Col : 2

`newvalue` Value = Reginald Ross

```
string newvalue =
DGVgolf.Rows[e.RowIndex].Cells[e.ColumnIndex].Value.ToString();
```



The screenshot shows a DataGridView with a single cell at row 5, column 2 ("Reginald Ross") highlighted in blue. The rest of the grid is visible with data in columns ID, Title, Firstname, and Surname.

| ID | Title | Firstname | Surname |
|----|-------|----------------|----------|
| 0 | Mr | Joe | Smith |
| 3 | Mr | Mathew Jam... | Sharpe |
| 4 | Mrs | Kim Soon | Lee |
| 5 | Mr | Reginald Ro... | Birdwood |
| 6 | Ms | Rachel | Taylor |

Golf Exercise using SQL

Create Buttons, or any other method to run your queries.

Find the following

1. Which people have a handicap under 11
2. Which Golfers are from Spreydon?
3. Which Golfers are NOT from Christchurch?
4. How many people are male?
5. Order the handicap in ascending order
6. How many people are available to play on the weekday?
7. How many people live in a street?
8. Without using the gender field, how many people are female?
9. How many people have a surname starting with B?
10. How many people are not from Christchurch or Rangiora?

22. Presentation, Business, and Data Layers / Tiers

"Programs should be written for people to read, and only incidentally for machines to execute." — Structure and Interpretation of Computer Programs by Abelson and Sussman

<https://msdn.microsoft.com/en-nz/library/ee658109.aspx?f=255&MSPPError=-2147217396>

<http://www.codeproject.com/Articles/36847/Three-Layer-Architecture-in-C-NET>

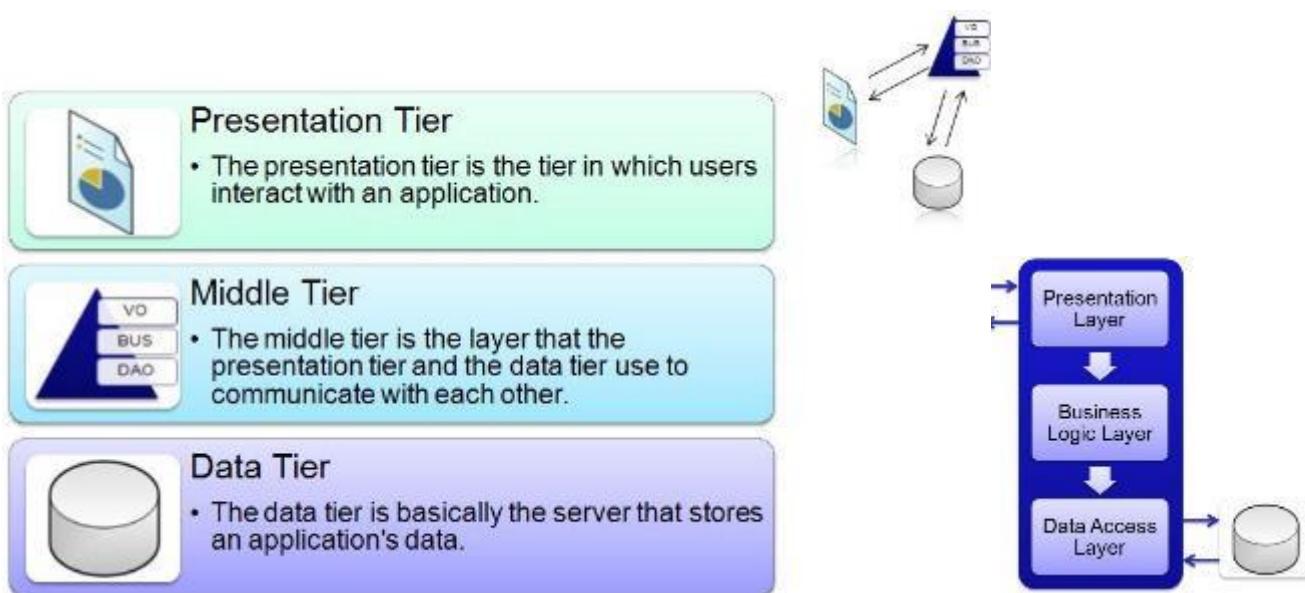
<http://www.codeproject.com/Articles/679185/Understanding-Three-Layer-Architecture-and-its>

<https://medium.com/@msandin/strategies-for-organizing-code-2c9d690b6f33#.3z451rjt4>

When a unit of code grows too large and contains too many elements it becomes hard to navigate, hard to get an overview of, and hard to understand: it becomes complex. Our main weapon against this complexity is **divide and conquer**: we split the unit into smaller parts which we can understand in isolation.

At the moment we have all our code stuffed under our Form. But that is not ideal, its hard to find, it gets really messy, and it makes growing the program hard.

We need to use **Layers or Tiers**.



The main benefits of the N-tier/3-tier architectural style are:

- **Maintainability.** Because each tier is independent of the other tiers, updates or changes can be carried out without affecting the application as a whole.
- **Scalability.** Because tiers are based on the deployment of layers, scaling out an application is reasonably straightforward.
- **Flexibility.** Because each tier can be managed or scaled independently, flexibility is increased.
- **Availability.** Applications can exploit the modular architecture of enabling systems using easily scalable components, which increases availability.

Create a Data Layer for your Golf Project

Here is my rough draft at present, with the Database calls now in their own place.

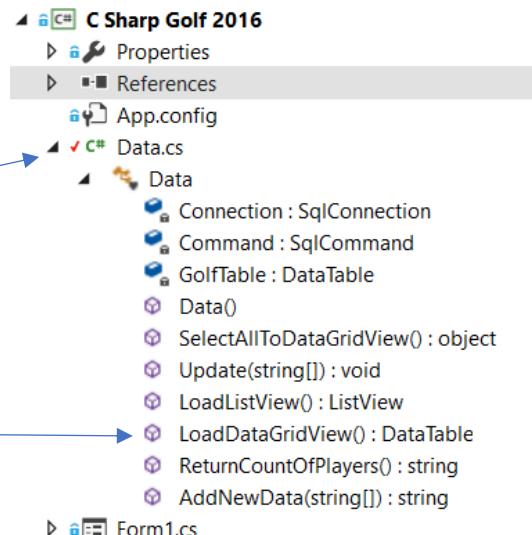
The hardest part is working out how to move the data between the different layers.

On my Form I instantiated the Data class as myData under the globals.

```
Data myData = new Data();
```

Then I could use it to call the methods in the class

```
DGVgolf.DataSource =
myData.LoadDataGridView();
```



Three ways of moving Data to the Data layer from the Form

On our Form we have a 11 text boxes holding the data that we need to move to our Data Class.

| | | | | | |
|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|
| ID | Title | Firstname | Surname | Gender | |
| <input type="text"/> | |
| DOB | Street | Suburb | City | Available | Handicap |
| <input type="text"/> |

We can do this in a number of ways and which one you use will depend on what you want to do with that data, and how tidy you are with your coding.

[Here is the final code](#) (if you want to skip ahead)

The easiest is to create fields, or properties, in your class and move the textbox data to them, like this **example** of the **Update Query**.

In the Data Class

```
public string Firstname, Surname, Title;
```

In the **Form** you just pass the Textboxes to the fields.

```
myData.Firstname = txtFirstname.Text;
myData.Surname = txtSurname.Text;
myData.Title = txtTitle.Text;
```

Then in your **Class Update Method** you just add the fields to the parameters for whatever you are doing.

```
update.Parameters.AddWithValue("@Title", Title);
update.Parameters.AddWithValue("@Firstname", Firstname);
```

```
update.Parameters.AddWithValue("@Surname", Surname);
```

This does mean that your code fills up with lots of connections however and can get messy.

Another way is to pass the data to the `myData.Update` method via the parameters.

```
myData.Update(txtFirstname.Text, txtSurname.Text, txtTitle.Text);
```

Then get them out and use them and use them in your code as above.

```
Update(String Firstname, String Surname, String Title);
```

However this quickly runs into problems, you don't want more than 3 parameters, and 11 is a REALLY BAD look.

So I used an Array instead.

In the Form I made a Global variable ...

```
Private String[] AllTextBoxes;
... and under the btnUpdate_Click method I passed in the text boxes and called
the myData.Update(AllTextBoxes) method passing through the array.
private void btnUpdate_Click(object sender, EventArgs e)
{
    AllTextBoxes = new string[]{txtID.Text, txtTitle.Text, txtFirstname.Text,
    txtSurname.Text, txtStreet.Text, txtSuburb.Text, txtCity.Text, txtGender.Text,
    txtDOB.Text, txtHandicap.Text, txtAvailable.Text };

    myData.Update(AllTextBoxes);
```

Then in the Update method in Data I just called the Array holding the data by its place in the array

```
public void Update(string[] AlltextBoxes) {
    string updatestatement ="UPDATE Golf set Title=@Title, Fristname=@Firstname, Surname=@Surname,
    Gender=@Gender, DOB=@DOB, Street=@Street, Suburb=@Suburb, City=@City, [Available week days]=@Available,
    Handicap=@Handicap where ID = @ID";
    using (SqlCommand update = new SqlCommand(updatestatement, Connection)) {
        //create the parameters and pass the data from the textboxes
        update.Parameters.AddWithValue("@ID", AlltextBoxes[0]);
        update.Parameters.AddWithValue("@Title", AlltextBoxes[1]);
        update.Parameters.AddWithValue("@Firstname", AlltextBoxes[2]);
```

Don't forget this is an example, I had 11 textboxes, here I only show three.

Returning data to the Form

To the DataGridView.

Sending data back to the form in a way that can be consumed can take some thinking. Some are easy.

For example the `LoadDataGridView ()` Class has a `DataTable GolfTable = new DataTable()`, which you have to fill with data and send to the DataGridView.

Here is the code for the first part of the Data Class.

```
class Data {
    private SqlConnection Connection = new SqlConnection();
    private SqlCommand Command = new SqlCommand();
    DataTable GolfTable = new DataTable();
    public Data() {
        Connection.ConnectionString = @"Data Source = GARY-LAPTOP\SQLEXPRESS; Initial Catalog = Golf;
Integrated Security = True";
        Command.Connection = Connection;
    }

    public DataTable LoadDataGridView() {
        string queryString = "SELECT * FROM Golf ORDER by ID";
        GolfTable.Clear();
        //add in the column titles to the datatable
        GolfTable.Columns.Add("ID");
        GolfTable.Columns.Add("Title");
        GolfTable.Columns.Add("Firstname");
        GolfTable.Columns.Add("Surname");
        GolfTable.Columns.Add("Gender");
        GolfTable.Columns.Add("DOB");
        GolfTable.Columns.Add("Street");
        GolfTable.Columns.Add("Suburb");
        GolfTable.Columns.Add("City");
        GolfTable.Columns.Add("Available week days");
        GolfTable.Columns.Add("Handicap");
        using (SqlCommand Command = new SqlCommand(queryString, Connection)) {
            Connection.Open();
            SqlDataReader reader = Command.ExecuteReader();
            while (reader.Read()) {
                //add in each row to the datatable
                GolfTable.Rows.Add(reader["ID"], reader["Title"], reader["Firstname"], reader["Surname"],
reader["Gender"], reader["DOB"], reader["Street"], reader["Suburb"], reader["City"],
reader["Available week days"], reader["Handicap"]);
            }
            reader.Close();
            Connection.Close();
            return GolfTable;
        }
    }
}
```

Meanwhile back on the form I made a method called `LoadFormFromClass()` that will hold the loading of the DataGrid and the ListView. The DGV gets the data from the class and passed to its Data Source.

```
public Form1() {
    InitializeComponent();
    LoadFormFromClass();
}

public void LoadFormFromClass() {
    DGVgolf.DataSource = myData.LoadDataGridView();
}
```

Add new Data

Here is the `AddNewData` Method with a Happy message returning when its successful.

You can expand this to return error messages as well. See how the Array `string[]`

`AlltextBoxes` takes in the data from the Text boxes. Note that when we add new data we don't add an **ID field**.

```
public string AddNewData(string[] AlltextBoxes) {
    // this puts the parameters into the code so that the data in the
    // text boxes is added to the database
    string QueryString = "INSERT INTO Golf (Title, Firstname, Surname, Gender, DOB, Street,
    Suburb, City, [Available week days], Handicap) VALUES (@Title, @Firstname, @Surname, @Gender, @DOB,
    @Street, @Suburb, @City, @Available, @Handicap)";

    SqlConnection Con = new SqlConnection();
    try {
        using (SqlCommand Com = new SqlCommand(QueryString, Con)) {
            Com.Parameters.AddWithValue("@Title", AlltextBoxes[1]);
            Com.Parameters.AddWithValue("@Firstname", AlltextBoxes[2]);
            Com.Parameters.AddWithValue("@Surname", AlltextBoxes[3]);
            Com.Parameters.AddWithValue("@Street", AlltextBoxes[4]);
            Com.Parameters.AddWithValue("@Suburb", AlltextBoxes[5]);
            Com.Parameters.AddWithValue("@City", AlltextBoxes[6]);
            Com.Parameters.AddWithValue("@Gender", AlltextBoxes[7]);
            Com.Parameters.AddWithValue("@DOB", AlltextBoxes[8]);
            Com.Parameters.AddWithValue("@Handicap", AlltextBoxes[9]);
            Com.Parameters.AddWithValue("@Available", AlltextBoxes[10]);

            Con.Open();
            //its a NONQuery as it doesn't return any data its only going up to the server
            Com.ExecuteNonQuery();
            Con.Close();
            //a happy message box
        }
        return AlltextBoxes[2] + " " + AlltextBoxes[3] + " has been Inserted !! ";
    }
} catch (Exception ex) {
```

```
        return ex.ToString();
    }
}
```

On the Form. The MessageBox shows the happy message (This is debatable as to its code smell. The question I would ask is “Why does making a post to the DB return back a string?”) It just feels wrong.

```
private void btnNew_Click(object sender, EventArgs e) {
    AllTextBoxes = new string[] {txtID.Text, txtTitle.Text,
    txtFirstname.Text, txtSurname.Text, txtStreet.Text, txtSuburb.Text,
    txtCity.Text, txtGender.Text, txtDOB.Text, txtHandicap.Text,
    txtAvailable.Text
};
    MessageBox.Show(myData.AddNewData(AllTextBoxes));
```

To the ListView

This was harder than I thought owing to Obtuse error messages. But we return a ListView that is passed to the ListView on the Form.

This side, in the Data class is straightforward, each time the data loops in pass it to a ListViewItem, `ListViewItem item = new ListViewItem` then pass that row of data to a fake ListView. `fakeLVGolf.Items.Add(item)`

```
public ListView LoadListView() {
    string queryString = "SELECT * FROM Golf ORDER by ID";
    ListView fakeLVGolf = new ListView();
using (SqlCommand Command = new SqlCommand(queryString, Connection)) {
    Connection.Open();
    SqlDataReader reader = Command.ExecuteReader();

    while (reader.Read()) {
//add each row to the listbox
        ListViewItem item = new ListViewItem(new[] { reader["ID"].ToString(),
            reader["Title"].ToString(), reader["Firstname"].ToString(), reader["Surname"].ToString(),
            reader["Gender"].ToString(), reader["DOB"].ToString(), reader["Street"].ToString(),
            reader["Suburb"].ToString(), reader["City"].ToString(), reader["Available week days"].ToString(),
            reader["Handicap"].ToString() });
        fakeLVGolf.Items.Add(item);
    }
    reader.Close();
    Connection.Close();
}
return fakeLVGolf;
}
```

The problem happens at the other end, getting the ListView data to pass to another ListView, the one on the form. [This Answer](#) solved it for me, using `items.Clone()` on the end.

I have added this to the existing `LoadFormFromClass()` class that is called on load from the form.

```
public void LoadFormFromClass() {
//Clear the old items in the Listview
LVGolf.Items.Clear();
foreach (ListViewItem items in myData.LoadListView().Items) {
    LVGolf.Items.Add((ListViewItem)items.Clone());
}
DGVgolf.DataSource = myData.LoadDataGridView();
}
```

Returning the Count of Players.

This method is easy, we are just returning a string.

```
public string ReturnCountOfPlayers() {
    // a simple Scalar query just returning one value.
    string queryString = "SELECT COUNT(ID) FROM Golf";
    using (SqlConnection connection = new
    SqlConnection(Connection.ConnectionString)) {
        SqlCommand Command = new SqlCommand(queryString, connection);
        connection.Open();
        return Command.ExecuteScalar().ToString();
    }
}
```

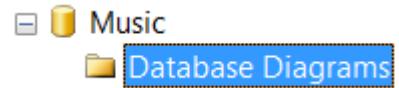
On the form its just

```
private void btnCount_Click(object sender, EventArgs e) {
    btnCount.Text = myData.ReturnCountOfPlayers();
```

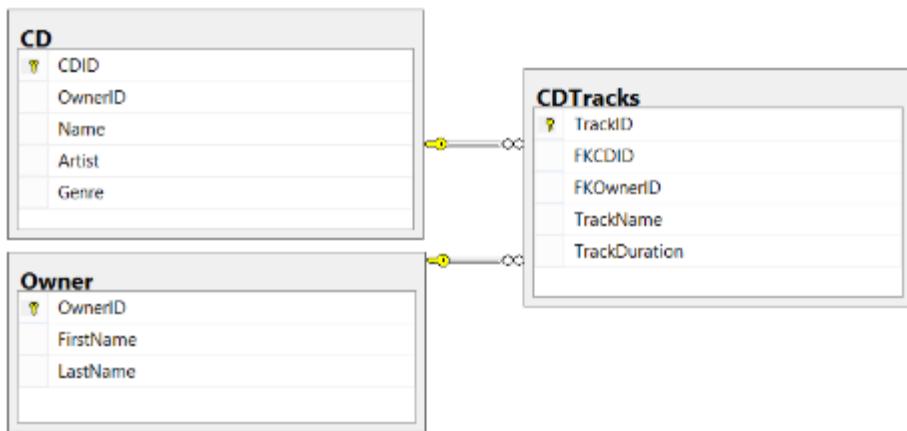
23. Music Database - Relational Database

Primary Key / Foreign Key relationships

The Music database is a RELATIONAL database. A database with more than one table and each table is connected to each other. This relationship scheme from Database Diagrams shows how the three tables are related.



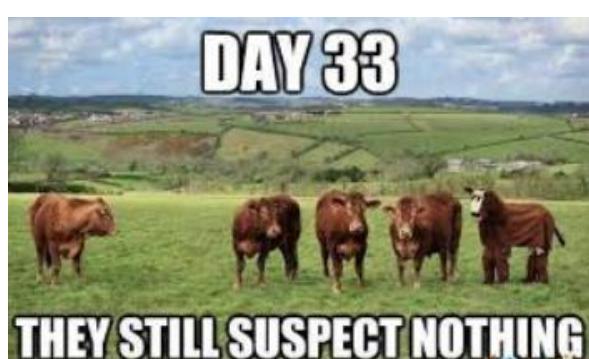
Each table has a Primary Key, which indicates that the field is one that will never repeat, it will always increment automatically 1, 2, 3, 4 etc. If you delete 3 then it will never replace it, the next record will be 5, then 6 then 7.



The tables are joined together with A **One to Many Relationship**.

The **Primary Key** side (One) shows the table that has the Primary Key in it, and the Infinite sign (Many) shows the table that can have many entries for that single key (**Foreign Key**).

Foreign keys are the alien invaders from the other primary key tables.



The Primary Key – Foreign Key connection establishes the relationship between the tables in a One to Many relationship.

For example, ONE Owner (OwnerID in the Owner Table) can have MANY CD's. (Owner ID in the CD Table).

ONE CD (CDID in the CD Table) has MANY tracks (CDID in the CDTracks Table).

Thus, by following the relationships you can easily see how many music tracks each Owner has, and their name and duration.

To make the next section easier (it will hopefully avoid the problems you will read later) go **Tools / Options / Designers** and **UNTICK** the **Prevent saving changes** that require table re-creation. This is not recommended for a 'real' database as it could mess up your data but for this practice it's an easy workaround.

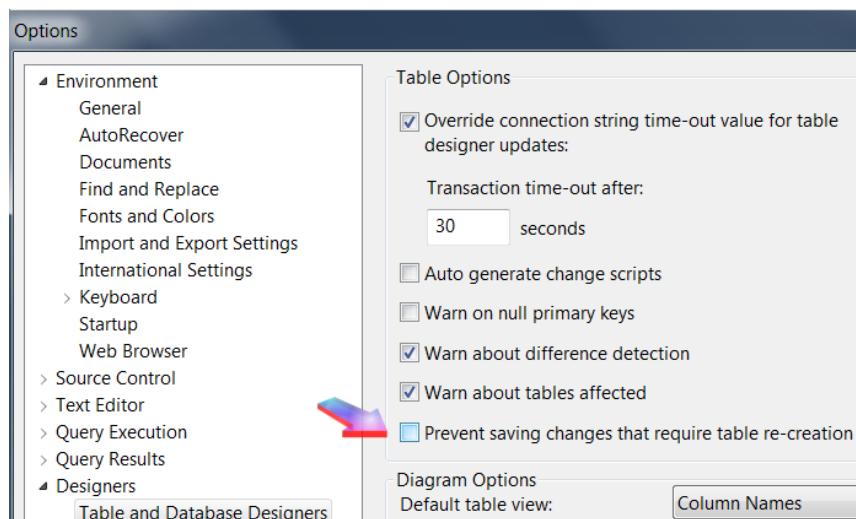


Table structure to date.

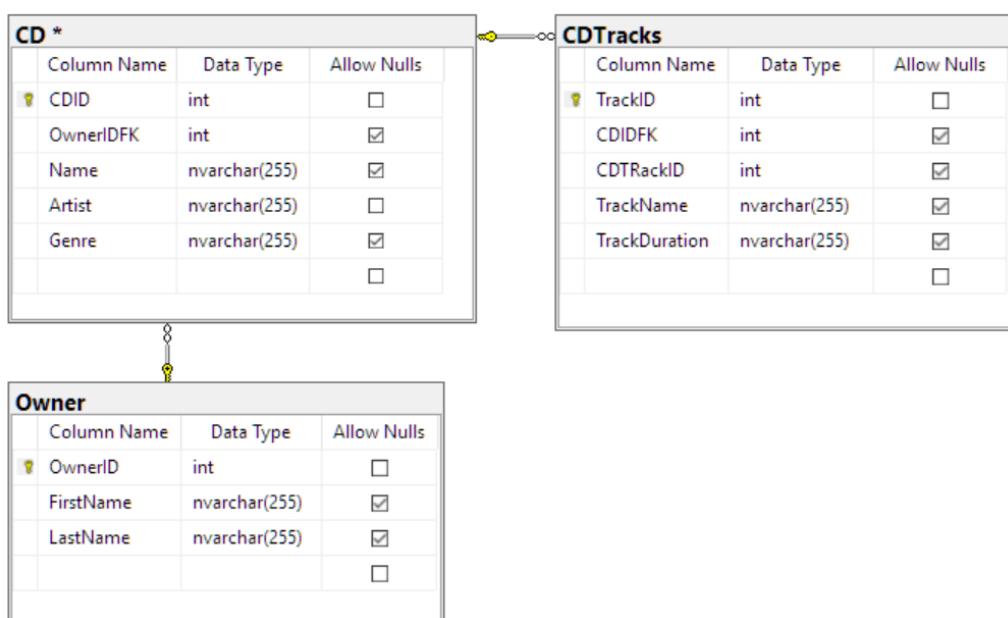


Table Data

Owner Table

| | OwnerID | FirstName | LastName |
|---|---------|-----------|-------------|
| 1 | 1 | John | Smith |
| 2 | 2 | Arnold | Swartznager |
| 3 | 3 | Harry | Houdini |
| 4 | 4 | Barry | Bartholomew |
| 5 | 5 | Craig | Carick |
| 6 | 6 | Pablo | Rod |

CD Table

| | CDID | OwnerIDFK | Name | Artist | Genre |
|----|------|-----------|-----------------------------------|------------------|------------------|
| 1 | 1 | 1 | ABBA Gold: Greatest Hits | ABBA | Pop |
| 2 | 2 | 1 | Led Zeppelin IV | Led Zeppelin | Pop |
| 3 | 3 | 2 | Their Greatest Hits (1971–1975) | Eagles | Hard rock |
| 4 | 4 | 2 | Saturday Night Fever | Bee Gees | Soundtrack |
| 5 | 5 | 3 | The Dark Side of the Moon | Pink Floyd | Progressive rock |
| 6 | 6 | 3 | Bat Out of Hell | Meat Loaf | Rock |
| 7 | 7 | 4 | Rumours | Fleetwood Mac | Rock |
| 8 | 8 | 4 | Sgt. Pepper's Lonely Hearts Cl... | The Beatles | Rock |
| 9 | 9 | 5 | Goodbye Yellow Brick Road | Elton John | Rock |
| 10 | 10 | 5 | Born in the U.S.A. | Bruce Springs... | Rock |
| 11 | 11 | 6 | 1984 | David Bowie | Rock |
| 12 | 16 | 6 | Rock and Roll Forever | David Dowie | Soundtrack |

Tracks Table

| TrackID | CDIDFK | CDTRackID | TrackName | TrackDurati... |
|---------|--------|-----------|---------------------------------|----------------|
| 1 | 1 | 1 | Dancing Queen | 3:45 |
| 2 | 1 | 2 | Knowing Me, Knowing You | 4:00 |
| 3 | 1 | 3 | Take a Chance on Me | 4:01 |
| 4 | 1 | 4 | Mamma Mia | 3:42 |
| 5 | 1 | 5 | Lay All Your Love on Me | 3:32 |
| 6 | 1 | 6 | Ring Ring | 3:02 |
| 7 | 1 | 7 | I Do, I Do, I Do, I Do, I Do | 3:15 |
| 8 | 1 | 8 | The Winner Takes It All | 4:54 |
| 9 | 1 | 9 | Money, Money, Money | 3:05 |
| 10 | 1 | 10 | S.O.S | 3:19 |
| 11 | 2 | 1 | Black Dog | 4:54 |
| 12 | 2 | 2 | Rock and Roll | 3:40 |
| 13 | 2 | 3 | The Battle of Evermore | 5:51 |
| 14 | 2 | 4 | Stairway to Heaven | 8:02 |
| 15 | 3 | 1 | Take It Easy | 3:29 |
| 16 | 3 | 2 | Witchy Woman | 4:10 |
| 17 | 3 | 3 | Lyin' Eyes | 6:21 |
| 18 | 3 | 4 | Already Gone | 4:13 |
| 19 | 4 | 1 | Stayin' Alive | 4:45 |
| 20 | 4 | 2 | How Deep Is Your Love | 4:05 |
| 21 | 4 | 3 | Night Fever | 3:33 |
| 22 | 4 | 4 | More Than a Woman | 3:18 |
| 23 | 5 | 1 | Speak to Me | 1:30 |
| 24 | 5 | 2 | Breathe | 2:43 |
| 25 | 5 | 3 | On the run | 3:30 |
| 26 | 5 | 4 | Time | 6:53 |
| 27 | 5 | 5 | The great gig int eh sky | 4:15 |
| 28 | 6 | 1 | Bat Out of Hell | 9:45 |
| 29 | 6 | 2 | You Took the Words Right ou... | 5:04 |
| 30 | 6 | 3 | Heaven Can Wait | 4:38 |
| 31 | 6 | 4 | All Revved Up with No Place ... | 4:19 |
| 32 | 7 | 1 | Second Hand News | 2:43 |
| 33 | 7 | 2 | Dreams | 4:14 |
| 34 | 7 | 3 | never going back again | 2:02 |
| 35 | 7 | 4 | Don't Stop | 3:11 |
| 36 | 7 | 5 | Go your own way | 3:38 |

| | | | | |
|----|----|---|-----------------------------------|-------|
| 37 | 8 | 1 | With a Little Help from My Fr... | 2:02 |
| 38 | 8 | 2 | Lucy in the Sky with Diamonds | 2:44 |
| 39 | 8 | 3 | Getting better | 3:28 |
| 40 | 8 | 4 | Fixing a Hole | 2:48 |
| 41 | 9 | 1 | Funeral for a Friend/Love Lies... | 11:09 |
| 42 | 9 | 2 | Candle in the Wind | 3:50 |
| 43 | 9 | 3 | Bennie and the Jets | 5:23 |
| 44 | 9 | 4 | Goodbye Yellow Brick Road | 3:13 |
| 45 | 10 | 1 | Born in the U.S.A. | 4:39 |
| 46 | 10 | 2 | Cover Me | 3:27 |
| 47 | 10 | 3 | Darlington County | 4:49 |
| 48 | 10 | 4 | Working on the Highway | 3:11 |
| 49 | 10 | 5 | Downbound Train | 3:35 |
| 50 | 10 | 6 | I'm on fire | 2:37 |
| 51 | 11 | 1 | No Idea | 3:00 |
| 52 | 11 | 2 | Still No Idea | 3:00 |
| 53 | 11 | 3 | Oh Dear | 3:00 |

Creating Database Views – Queries

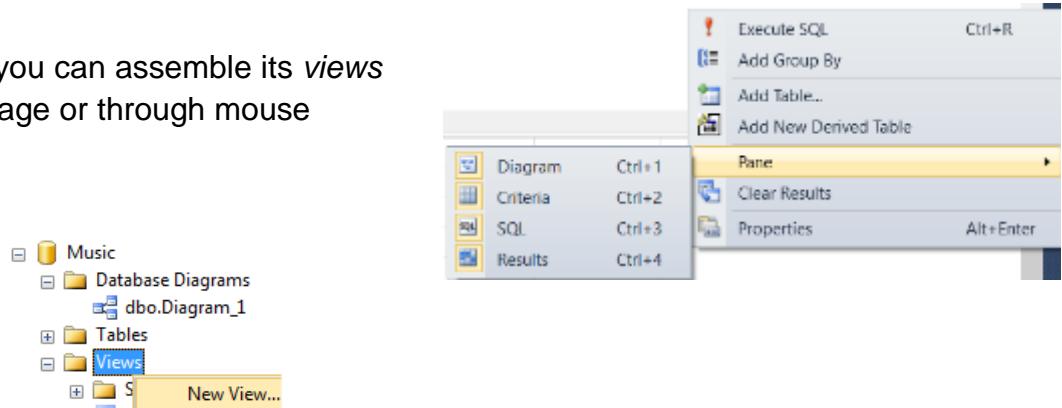
CDs and Owners

Views are Queries, or questions that you make from your tables using SQL. Luckily Views generate the SQL for you. [Here is a great Intro to it.](#)

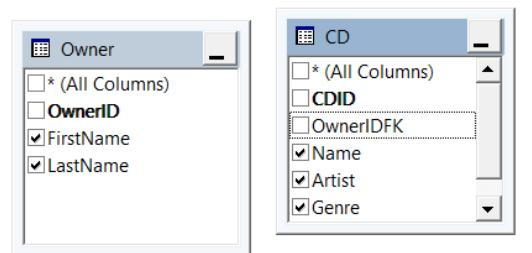
The Views window is divided in four parts: **Diagram Panel, Criteria Panel, SQL Panel and Results Panel.**

Through these panels you can assemble its *views* through the SQL language or through mouse selection.

Create a new view

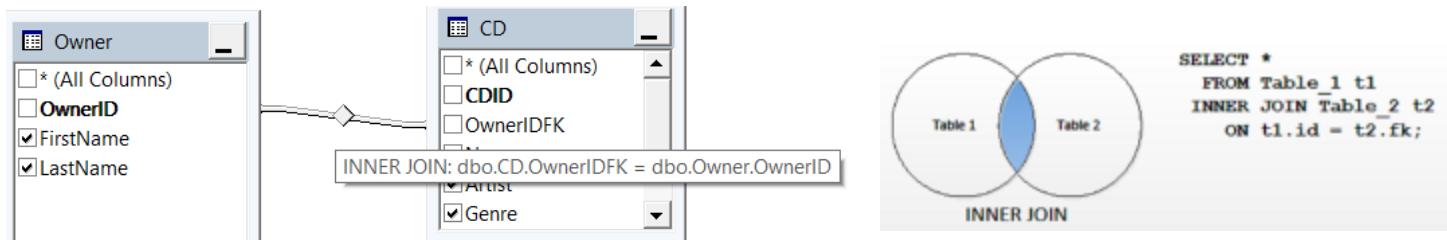


Add the Owner and CD tables by right clicking on the first section and going **Add Table**.



Click on the OwnerIDFK and drag it home to the Owner Table

This builds a connection and chooses the most obvious SQL See how the SQL is written for you.



```
SELECT dbo.CD.Name, dbo.CD.Artist, dbo.CD.Genre, dbo.Owner.FirstName, dbo.Owner.LastName
FROM   dbo.CD INNER JOIN
       dbo.Owner ON dbo.CD.OwnerIDFK = dbo.Owner.OwnerID
```

Hit the  Execute/Run button to see the results generate at the bottom of the screen.
TaDa!

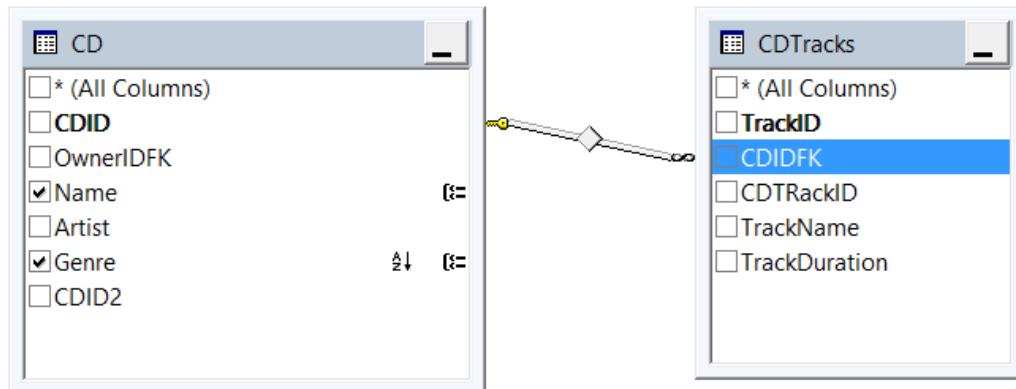
| | Name | Artist | Genre | FirstName | LastName |
|---|-----------------|--------------|------------------|-----------|-------------|
| ▶ | ABBA Gold: ... | ABBA | Pop | John | Smith |
| | Led Zeppelin... | Led Zeppelin | Pop | John | Smith |
| | Their Greate... | Eagles | Hard rock | Arnold | Swartznager |
| | Saturday Ni... | Bee Gees | Soundtrack | Arnold | Swartznager |
| | The Dark Si... | Pink Floyd | Progressive rock | Harry | Houdini |

Save as **OwnerCD**

Count of Tracks by CD

Lets make a more interesting View. [Using Filtering](#)

We want to **count how many Tracks** there are on each CD. Create a new View and add CD and CDTTracks Table. Drag CDIDFK home to the CD table.



Click a tick beside Name and TrackID or choose them from the Column and Table field pull downs.

Click the GroupBy button  then when the Group By column appears choose Count for TrackID

| Column | Alias | Table | Output | Sort Type | Sort Order | Group By | F |
|---------|-------|----------|-------------------------------------|-----------|------------|----------|---|
| Name | | CD | <input checked="" type="checkbox"/> | | | Group By | |
| TrackID | Expr1 | CDTracks | <input checked="" type="checkbox"/> | | | Count | |

You will now have the view below laid out Click the Run  button and see what you get.

| Name | Expr1 |
|---------------------------------------|-------|
| Sgt. Pepper's Lonely Hearts Club Band | 4 |
| Their Greatest Hits (1971–1975) | 4 |
| ABBA Gold: Greatest Hits | 10 |
| Bat Out of Hell | 4 |

Note that the column that shows how many tracks on each CD is called **Expr1**. Change the name in the Alias column to read **CountofTracks**. Now run again and see the new column name. Save the View as **CountOfTracks**.

Group By Genre

Create a new view grouping all the records by Genre. Save as **GroupByGenre**.

| Column | Alias | Table | Output | Sort Type | Sort Order | Group By |
|--------|------------|-------|-------------------------------------|-----------|------------|----------|
| Genre | | CD | <input checked="" type="checkbox"/> | | | Group By |
| Genre | GenreCount | CD | <input checked="" type="checkbox"/> | | | Count |
| | | | <input type="checkbox"/> | | | |
| | | | <input type="checkbox"/> | | | |
| | | | <input type="checkbox"/> | | | |

```
SELECT TOP (100) PERCENT dbo.CD.Genre, COUNT(dbo.CD.Genre) AS GenreCount
FROM dbo.CD INNER JOIN
      dbo.CDTracks ON dbo.CD.CDID = dbo.CDTracks.CDIDFK
GROUP BY dbo.CD.Genre
```

| Genre | GenreCount |
|------------------|------------|
| Hard rock | 4 |
| Pop | 14 |
| Progressive rock | 5 |
| Rock | 23 |
| Soundtrack | 4 |

Unique Genre

Make a new View Save it as **UniqueGenre** – to show just unique names from the Genre list instead of each entry

The screenshot shows the creation of a new view named **UniqueGenre** in SQL Server Management Studio. The view is defined as:

```
SELECT DISTINCT Genre
FROM CD
```

A screenshot of the **CD** table properties is also shown, highlighting the **Genre** column.

Who has the CD with the most Tracks? Max of Tracks

To answer this question we need to first find the CD with the most Tracks and save it as MaxTracks Save and close it. **UseCDTrackID**

The screenshot shows a SQL query being run in SSMS:

```

SELECT MAX(CDIDFK) AS MaxTracks
FROM   dbo.CDTracks

```

The results show a single row with MaxTracks = 10.

Make a new View and add the following two views and two tables below.

Drag the **MaxTracks** field (the one with the tick) from its Table box over to CountofTracks and drop it over the **CountofTracks**.

This will show the program that MaxTracks is related to CountofTracks and build the little connection.

Do the same for CountOfTracks **Name** and CD **Name**. That will make a path all the way to Owner showing John Smith owns the record Abba with the most tracks.

The diagram illustrates the relationships between four tables:

- MaxTracks**: Contains a single column **MaxTracks** (checked).
- CountOfTracks**: Contains columns **Name** and **CountOfTracks**.
- CD**: Contains columns **CDID**, **OwnerIDFK**, **Name**, **Artist**, and **Genre**.
- Owner**: Contains columns **OwnerID**, **FirstName**, and **LastName**.

Relationships are established via foreign keys:

- A relationship exists between **MaxTracks** and **CountOfTracks** based on **MaxTracks**.
- A relationship exists between **CountOfTracks** and **CD** based on **CountOfTracks**.
- A relationship exists between **CD** and **Owner** based on **OwnerIDFK**.

The final query is:

```

SELECT dbo.MaxTracks.MaxTracks, dbo.Owner.FirstName, dbo.Owner.LastName
FROM   dbo.CountOfTracks INNER JOIN
       dbo.MaxTracks ON dbo.CountOfTracks.CountOfTracks = dbo.MaxTracks.MaxTracks INNER JOIN
       dbo.CD INNER JOIN
       dbo.Owner ON dbo.CD.OwnerIDFK = dbo.Owner.OwnerID ON dbo.CountOfTracks.Name = dbo.CD.Name

```

The results show a single row with MaxTracks = 10, FirstName = John, and LastName = Smith.

24. Music Database in Visual Studio

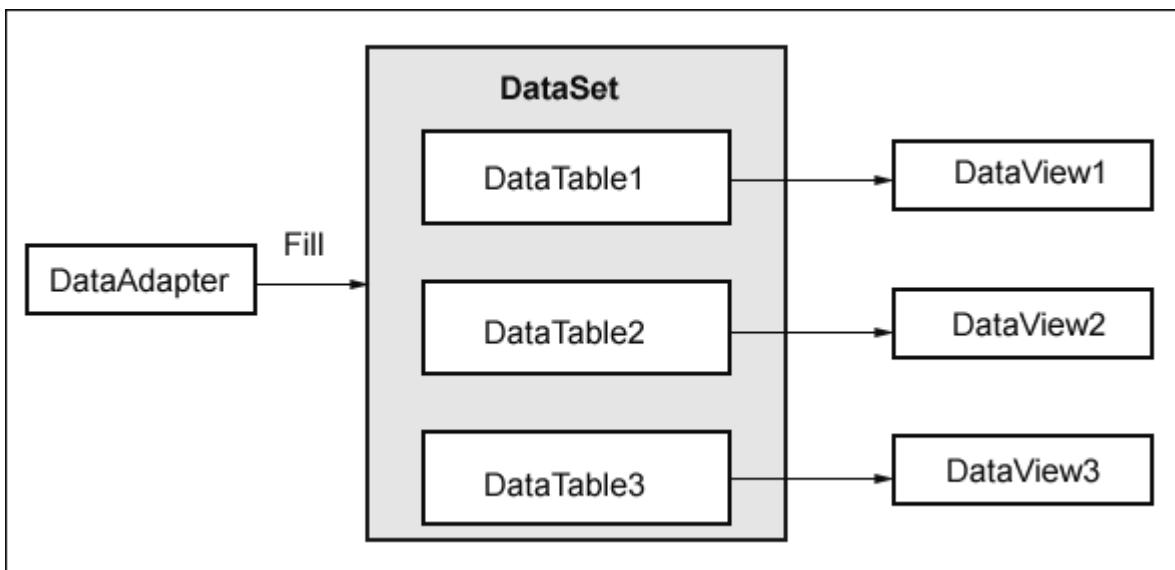
Create a new Project in Visual Studio and add the connections to connect to the Music database.

In the Golf exercise we just used a **DataTable**. This holds only one table's data.

As we are working with **multiple tables** in this exercise we have to use a **DataAdapter**, and sometimes a **DataSet** which hold multiple DataTables and all the schema, headings etc. that go with them.

The DataAdapter serves as a bridge between a DataSet and SQL Server for retrieving and saving data. [Populating a DataSet from a DataAdapter - Read](#). [Using the DataAdapter](#).

The DataAdapter provides this bridge by mapping **Fill**, which changes the data in the DataSet to match the data in the data source,



Using a DataGridView

Add a DataGridView to your form and the following code. The DataGridView control reads the schema information and creates the correct number of columns for your data.

It also uses the column names in the schema as the column names for the grid and each column has the default width.

The **SqlDataAdapter** reads the data from the database and populates the DataTable using the Fill method. Previously we used a **Reader** to read the data out. This does it all at once.

Note that you don't have to actually open and close the connection explicitly as the DataAdapter's Fill() method leaves the connection in the same state as when the method was invoked.

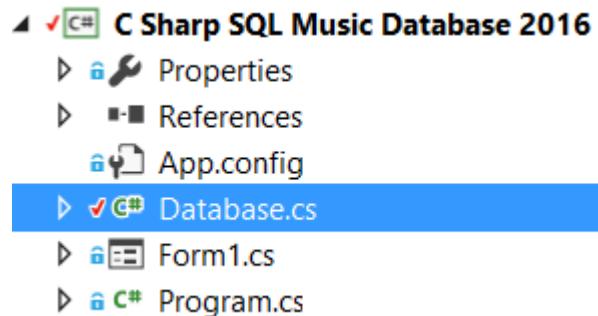
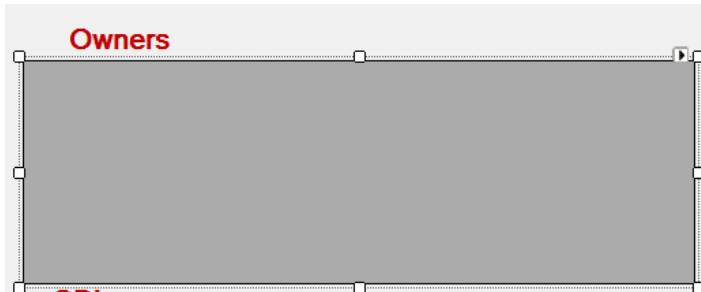
Read this great overview of the DataAdapter [Five-different-overloads-of-the-DataAdapter-Fill](#)

The Owners DataGridView and Class

Owing to the complexity of this program we are going to build it piece by piece.

Create a new **DataGridView**, Name it **DGVOwner**

Add a **Label**, have the **LblOwner.text = "Owners"**.



Create a new Class to hold your Database Calls in.

In your **Database.CS** class set some private fields for the SQL to be used only in the class. Set also defaults such as the Connection string and join it to the Commands

```
class Database
{
    //Create Connection and Command, and an Adapter.
    private SqlConnection Connection = new SqlConnection();
    private SqlCommand Command = new SqlCommand();
    private SqlDataAdapter da = new SqlDataAdapter();
    //THE CONSTRUCTOR SETS THE DEFAULTS UPON LOADING THE CLASS
    public Database()
    {
        //change the connection string to run from your own music db
        string connectionString = @"Data Source=GARY-LAPTOP\sqlexpress;Initial Catalog=Music;Integrated Security=True";
        Connection.ConnectionString = connectionString;
        Command.Connection = Connection;
    }

    public DataTable FillDGVOwnerWithOwner()
    {
        //create a datatable as we only have one table, the Owner
        DataTable dt = new DataTable();
        using (da = new SqlDataAdapter("select * from Owner ", Connection))
        {
            //connect in to the DB and get the SQL
            Connection.Open();
            //open a connection to the DB
            da.Fill(dt);
        }
    }
}
```

Default settings

```

        //fill the datatable from the SQL
        Connection.Close(); //close the connection
    }
    return dt; //pass the datatable data to the DataGridView
}

```

Meanwhile back on the form we need to Instantiate the class `Database myDatabase = new Database()`; so we can access the methods we are making.

We make a LoadDB method, `LoadDB()`; to hold all the calls to fill each of the three DataGridView's with table data. Then make a method `DisplayDataGridViewOwner()` to pass the data to the DataGridView.

```

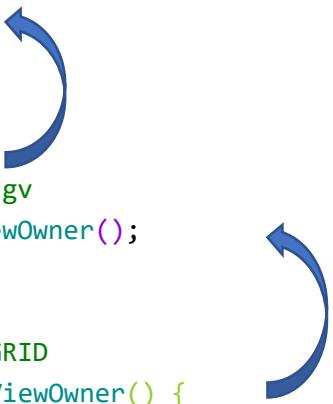
public partial class Form1 : Form {
    //create an instance of the Database class
    Database myDatabase = new Database();

    public Form1() {
        InitializeComponent();
        loadDB();
    }

    public void loadDB() {
        //load the owner dgv
        DisplayDataGridViewOwner();
    }

    //LOAD THE OWNER DATAGRID
    private void DisplayDataGridViewOwner() {
        //clear out the old data
        DGVOwner.DataSource = null;
        try {
            DGVOwner.DataSource = myDatabase.FillDGVOwnerWithOwner();
            //pass the datatable data to the DataGridView
        DGVOwner.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells);
        } catch (Exception ex) {
            MessageBox.Show(ex.Message);
        }
    }
}

```



Fill the ListBox from the Database

Create a small ListBox to hold the names of the Genres



Add the code to fill it from the View to your Database Class.

```
public List<string> FillListBoxWithGenre() {  
  
    var myCommand = new SqlCommand("select genre from UniqueGenre",  
Connection);  
    //Create a list to hold all the genre, then pass it back to the  
listbox on the form  
    List<string> newgenre = new List<string>();  
    Connection.Open();  
    SqlDataReader reader = myCommand.ExecuteReader();  
    //loop through the genres and pass it to a reader, that gets  
added to the list  
    while (reader.Read()) {  
        newgenre.Add(reader["genre"].ToString());  
    }  
    reader.Close();  
    Connection.Close();  
    return newgenre; //send the list back to the listbox  
  
}
```

Your Form Code

```
private void DisplayListBox() {
    lbgenre.DataSource = null;
    try {
        lbgenre.DataSource = myDatabase.FillListBoxWithGenre();
    } catch (Exception ex) {
        MessageBox.Show(ex.Message);
    }
}
```

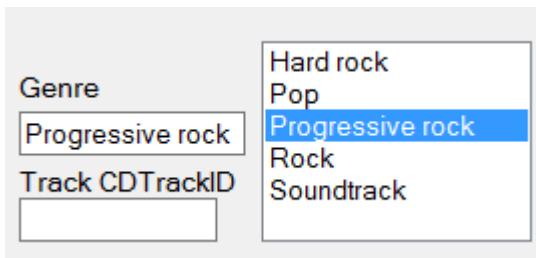
Add `DisplayListBox()` to the `loadDB()` method so it runs at startup.

```
public void loadDB() {
    //just to show the listbox with the genres in it
    DisplayListBox();
```

Double click on your ListBox on the form to generate a `lbgenre_SelectedIndexChanged` event

```
private void lbgenre_SelectedIndexChanged(object sender, EventArgs e) {
    txtCDGenre.Text = lbgenre.SelectedItem.ToString();
}
```

Now when you click on a genre it gets added to the Genre textbox. This will make it quicker to add new data.



Using Views in the Form

The Views are just like tables. So replace the SQL table with the View name like this

```
SQL = "select * from GroupByGenre"
```

After much pondering and experimentation I found that although you can name your views **Owner-CD** in the Management Studio, Visual Studio or SQL doesn't like the Hyphen so rename the view to **OwnerCD**

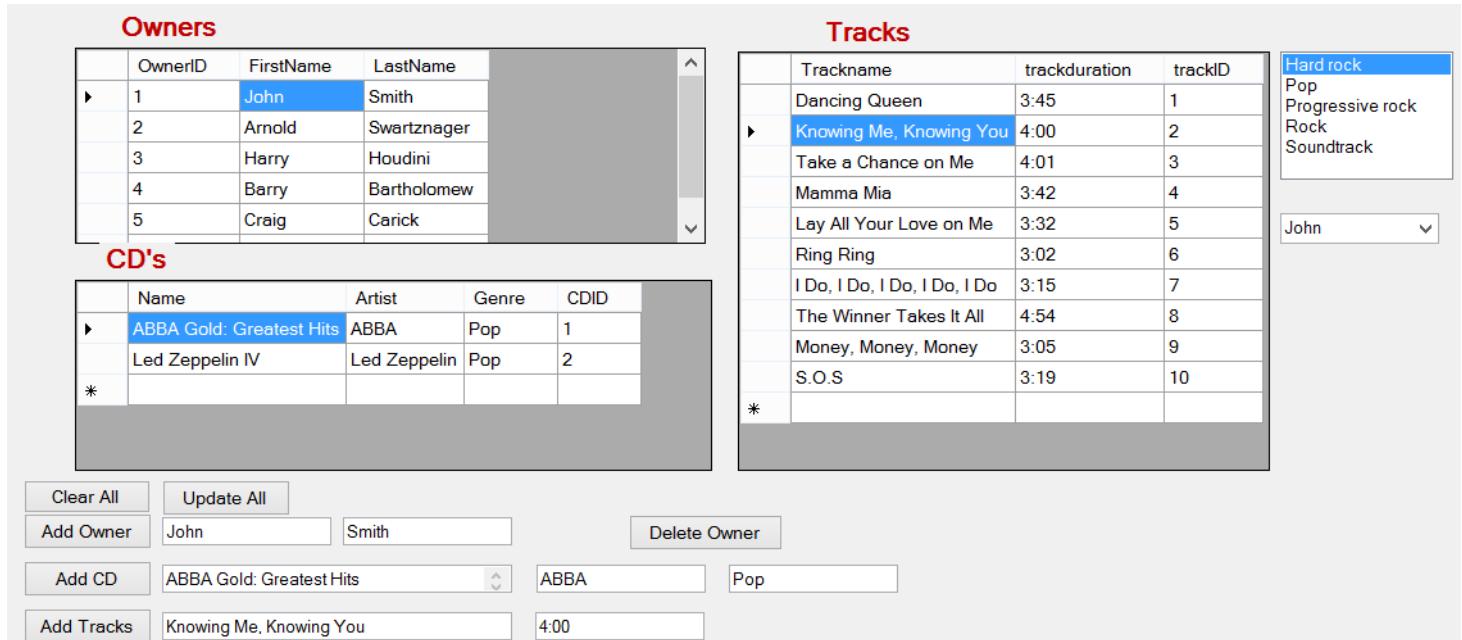
```
SQL = "select * from OwnerCD"
```

Linking DataGridViews to each other

In the image below you can see 3 DataGridView, each drawing from a different table. Owners, CD's and CD Tracks.

When you click on an owner in the Owners DGV the second DGV CD's shows the CD's that the owner has.

When you click on a CD in the second DGV you see the tracks of the CD in the third DGV. A nice click based navigation system.



The Names, CD, and tracks clicked on also fill the text boxes at the bottom.

There are two parts to making this magic happen.

1. Fill a DGV with data from a table.
2. Create a click event so clicking on a record sets the next DGV. Both of these are easy to create.

Lets create the **Owner** DGV. The other two DGVs' **Music** and **Tracks** are virtually identical so you can copy this and use it later changing the names in the code. You will recognise the first DGV from the previous exercise.

The Owner DataGridView Click Event

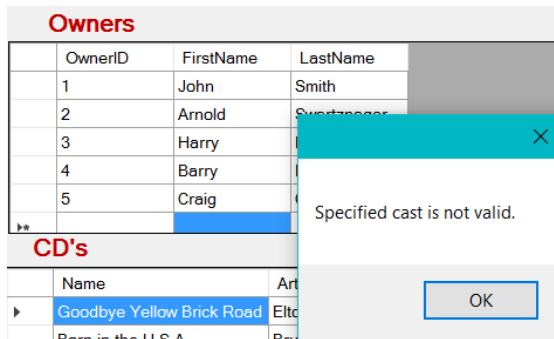
Double click on the DGVOwner to load the code for the click event.

Remember to change the Event trigger for DGVOwner to CellClick



The code is nested in a **Try Catch** in case you click in a place that doesn't return a value, otherwise it will make an error.

```
} catch (Exception ex) {
    MessageBox.Show(ex.Message);
}
```



The heart of this code is this line which returns every cell you click on.

```
OwnerID = (int)DGVOwner.Rows[e.RowIndex].Cells[0].Value;
txtFN.Text = DGVOwner.Rows[e.RowIndex].Cells[1].Value.ToString();
txtLN.Text = DGVOwner.Rows[e.RowIndex].Cells[2].Value.ToString();
```

We want everything from the row that you click on `Rows[e.RowIndex]` but we need to extract out the data in each row. Columns start at 0, so its 0, 1, 2.

`Cells[0]` `Cells[1]` `Cells[2]`

| OwnerID | FirstName | LastName |
|---------|-----------|-------------|
| 1 | John | Smith |
| 2 | Arnold | Swartznager |
| 3 | Harry | Houdini |

The **OwnerID** is in the first column `Cells[0]`,

The **First Name** is in the second column.
`Cells[1]`

The **Last Name** is in the 3rd column. `Cells[2]`

Before we can make the method for the click event we have to create the Database call method in the Class first.

To see the method above [click here](#) and bypass the next lesson at your peril.

Fill the DataGridView With the Owner Click

You need to make the method below in your **Database Class** first.

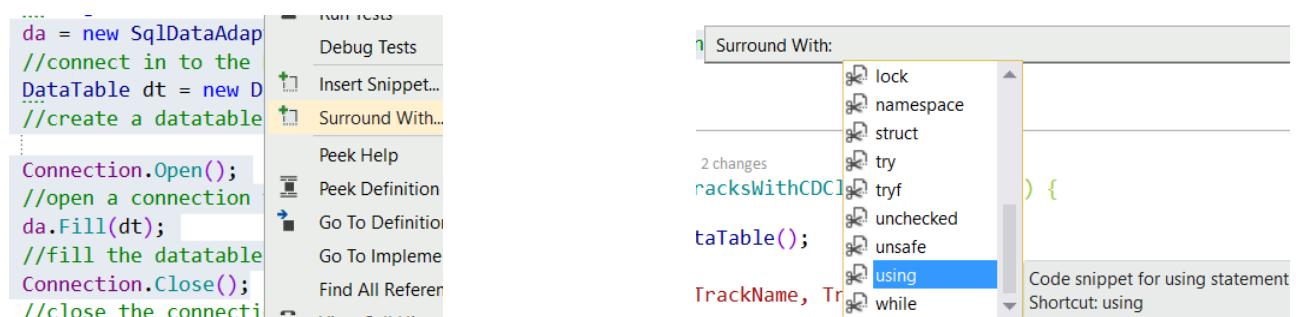
```
public DataTable FillDGVCWithOwnerClick(string Ownervalue) {
    string SQL = "select Name, Artist, Genre, CDID from CD where OwnerIDFK = '" + Ownervalue + "' ";
    da = new SqlDataAdapter(SQL, Connection);
    //connect in to the DB and get the SQL
    DataTable dt = new DataTable();
    //create a datatable as we only have one table, the Owner

    Connection.Open();
    //open a connection to the DB
    da.Fill(dt);
    //fill the datatable from the SQL
    Connection.Close();
    //close the connection

    return dt;
}
```

This Method `myDatabase.FillDGVCWithOwnerClick(OwnerId.ToString());` gets all the data and sends it back as a `DataTable` to the `DGVCD`,

This needs to be wrapped with a **Using** Statement below. The easiest way is to highlight the part you want wrapped, Right Click and choose Surround With. Then scroll down and choose **Using**.



```
public DataTable FillDGVCWithOwnerClick(string OwnerID)
{
    string SQL = "select Name, Artist, Genre, CDID from CD where OwnerIDFK = '" + OwnerID + "' ";
    using (da = new SqlDataAdapter(SQL, Connection))
    {
        //connect in to the DB and get the SQL
        DataTable mydt = new DataTable();
        //create a datatable as we only have one table, the Owner
        Connection.Open();
        //open a connection to the DB
        da.Fill(mydt);
        //fill the datatable from the SQL
        Connection.Close();
        //close the connection

        return mydt;
    }
}
```

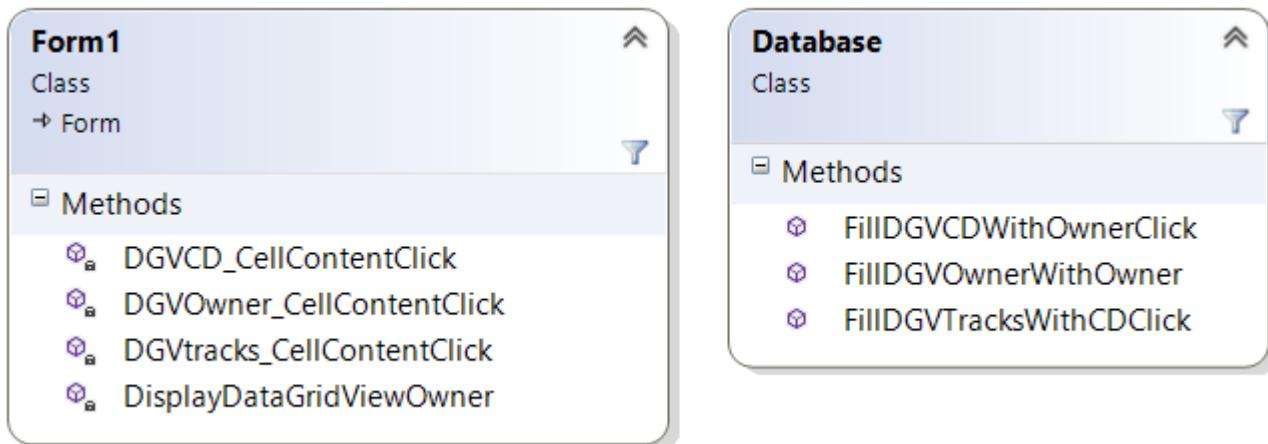
DGV Owner Cell Content Click event

Here is the code for the `DGVOwner_CellContentClick` DataGridView click event.

```
private void DGVOwner_CellContentClick(Object sender, DataGridViewCellEventArgs e) {
    int OwnerID = 0;
    //these are the cell clicks for the values in the row that you click on
    try {
        OwnerID = (int)DGVOwner.Rows[e.RowIndex].Cells[0].Value;
        txtFN.Text = DGVOwner.Rows[e.RowIndex].Cells[1].Value.ToString();
        txtLN.Text = DGVOwner.Rows[e.RowIndex].Cells[2].Value.ToString();
        //if you are clicking on a row and not outside it
        if (e.RowIndex >= 0) {
            //Fill the next CD DGV with the OwnerID
            DGVC.Datasource = myDatabase.FillDGVCWithOwnerClick(OwnerID.ToString());
            DGVC.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.DisplayedCells;
            TxtOwnerID.Text = OwnerID.ToString();
        }
    } catch (Exception ex) {
        MessageBox.Show(ex.Message);
    }
}
```

So what about the other DGV?

Their code is very similar to the ones before.



So you First run `DisplayDataGridViewOwner()` this fills the DGV with all the owners.

| Owners | | |
|---------|-----------|-------------|
| OwnerID | FirstName | LastName |
| 1 | John | Smith |
| 2 | Arnold | Swartzinger |
| 3 | Harry | Houdini |
| 4 | Berry | Bartholomew |
| 5 | Craig | Quirk |

CD's

```
public DataTable FillDGVOwnerWithOwner()
```

Then you click on an owner `DGVOwner_CellContentClick` and you fill the `FillDGVCDDWithOwnerClick(string OwnerID)`.with the owner you clicked on.

| Owners | | |
|---------|-----------|-------------|
| OwnerID | FirstName | LastName |
| 1 | John | Smith |
| 2 | Arnold | Swartzinger |
| 3 | Harry | Houdini |
| 4 | Berry | Bartholomew |
| 5 | Craig | Quirk |

CD's

| Name | Artist | Genre | CD |
|---------------------------------------|---------------|-------|----|
| Rumours | Fleetwood Mac | Rock | |
| Sgt. Pepper's Lonely Hearts Club Band | The Beatles | Rock | |

We need the OwnerID 4
`string OwnerID`

```
public DataTable FillDGVCDDWithOwnerClick(string OwnerID)
```

Then you click on a CD `DGVCD_CellContentClick` and you fill the Tracks Table `FillDGVTracksWithCDCClick(string CDID)`

| Owners | | |
|---------|-----------|-------------|
| OwnerID | FirstName | LastName |
| 1 | John | Smith |
| 2 | Arnold | Swartzinger |
| 3 | Harry | Houdini |
| 4 | Berry | Bartholomew |
| 5 | Craig | Quirk |

CD's

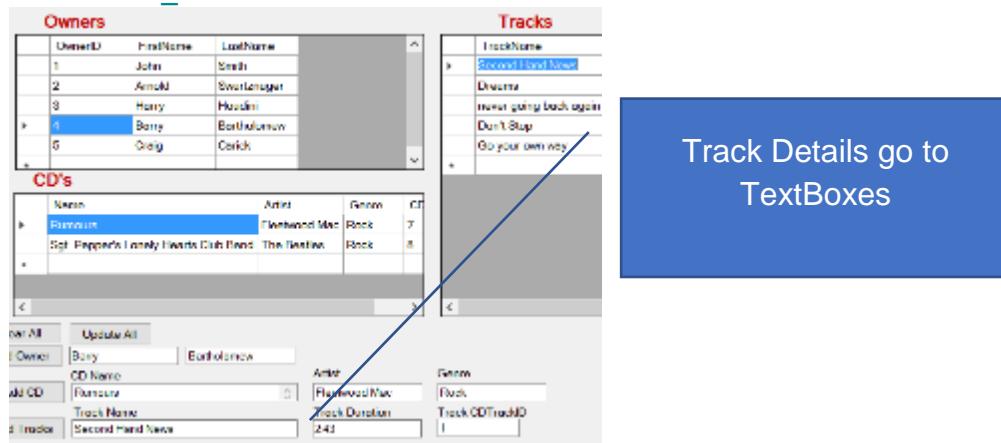
| Name | Artist | Genre | CD |
|---------------------------------------|---------------|-------|----|
| Rumours | Fleetwood Mac | Rock | 7 |
| Sgt. Pepper's Lonely Hearts Club Band | The Beatles | Rock | 0 |

We need the CDID 7
`string CDID`

```
public DataTable FillDGVTracksWithCDCClick(string CDID)
```

Then we click on a Track and it puts the details in the Track Name textbox.

DGVtracks_CellContentClick



Here are the next methods from the **Database Class**

```
1 reference | GaryDix, 3 days ago | 1 author, 2 changes
public DataTable FillDGVOwnerWithOwner() ...
5 references | GaryDix, 8 days ago | 1 author, 1 change
public DataTable FillDGVCDDWithOwnerClick(string OwnerID) ...
5 references | GaryDix, 7 days ago | 1 author, 2 changes
public DataTable FillDGVTracksWithCDCClick(string CDID) ...
```

Create the CD and Tracks DataGridView Click events.

Firstly, you will need to duplicate the Owners [DataGridView and Class](#) calls.

Next create the **CD click event**. Use the click event for the Owner cell and just change the code as well as change the name of your DataGridView to the one for your CD grid.

In this way you have daisy chained together the three DataGridView's to create a full application.

Finally, it would be nice to be able to click on the track and have it play but that is beyond the scope of this program.

Resizing Columns to fit contents

This seems to be a hit or miss issue but I found

```
DGVmusic.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.DisplayedCells)
```

Or the following to have some success.

```
DGVmusic.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.DisplayedCells
```

Update and Delete Queries

The Insert and Update are very similar so I combined them into one rather lengthy method. I am not sure if it's a recommended idea, as smaller methods might be tidier.

In my code I called the method twice and you can see from the parameters the fields that I pass through

```
▲ C Sharp SQL Music Database 2016\Form1.cs (2)
  ↗ 104 : result = myDatabase.InsertOrUpdateOwner(txtFN.Text, txtLN.Text, TxtOwnerID.Text, "Add");
  ↗ 331 : result = myDatabase.InsertOrUpdateOwner(txtFN.Text, txtLN.Text, TxtOwnerID.Text, "Update");
Show on Code Map | Collapse All
```

203 2 references | Gary Dix, 164 days ago | 2 authors, 5 changes
 204 public **string** InsertOrUpdateOwner(**string** Firstname, **string** Lastname, **string** ID, **string**:
 AddOrUpdate)

First name, Last name, ID and the text Add or Update.

```
public string InsertOrUpdateOwner(string Firstname, string Lastname, string ID, string
AddOrUpdate)
{
    try
    {
        //Add gets passed through the parameter
        if (AddOrUpdate == "Add")
        {
            //Create a Command object //Create a Query. Create and open a connection to SQL Server
            string query = "INSERT INTO Owner (FirstName, LastName) " +
                           "VALUES(@Firstname, @Lastname)";

            var myCommand = new SqlCommand(query, Connection);
            //create params
            myCommand.Parameters.AddWithValue("Firstname", Firstname);
            myCommand.Parameters.AddWithValue("Lastname", Lastname);
            Connection.Open();
            // open connection add in the SQL
            myCommand.ExecuteNonQuery();
            Connection.Close();
        }
        //Update gets passed through the parameter
        else if (AddOrUpdate == "Update")
        {
            var myCommand = new SqlCommand("UPDATE Owner set FirstName = @Firstname,
LastName=@Lastname where OwnerID = @ID ", Connection);
            //use parameters to prevent SQL injections
            myCommand.Parameters.AddWithValue("Firstname", Firstname);
            myCommand.Parameters.AddWithValue("Lastname", Lastname);
            myCommand.Parameters.AddWithValue("ID", ID);
            Connection.Open();
            // open connection add in the SQL
            myCommand.ExecuteNonQuery();
            Connection.Close();
        }
    }
}
```

```

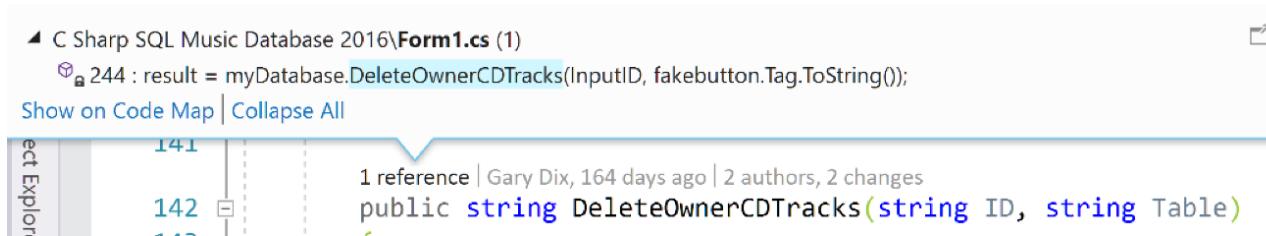
        }

        return " is Successful";
    }
    catch (Exception e)
    {
//need to get it to close a second time as it jumps the first connection.Close if
ExecuteNonQuery fails.
        Connection.Close();
        return " has Failed with " + e;
    }
}

```

Delete is really easy, all you need is the ID of the record you want to delete and pass it through

This is called by one method, but you can call it by all four methods. I pass through the **ID** and the name of the table (**Owner, CD, Track**) that I want deleted.



```

public string DeleteOwnerCDTracks(string ID, string Table)
{
    //only run if there is something in the textbox
    if (!object.ReferenceEquals(ID, string.Empty))
    {
        var myCommand = new SqlCommand();
        switch (Table)
        {
            case "Owner":
myCommand = new SqlCommand("DELETE FROM Owner WHERE OwnerID = @ID", Connection);
                break;
            case "CD":
myCommand = new SqlCommand("DELETE FROM CD WHERE CDID = @ID", Connection);
                break;
            case "Track":
myCommand = new SqlCommand("DELETE FROM CDTracks WHERE TrackID = @ID", Connection);
                break;
        }

        myCommand.Parameters.AddWithValue("ID", ID);
        //use parameters to prevent SQL injections

        Connection.Open();
    }
}

```

```
// open connection add in the SQL  
myCommand.ExecuteNonQuery();  
Connection.Close();  
return "Success";  
}  
else  
{  
    Connection.Close();  
    return "Failed";  
}
```

“The thing about programming is that your learning is never complete, and neither are your bug hunting or your crying.”

#programming #coding #devlife
#compsci

Feb 25th, 2016
374 notes



25. Preventing SQL Injections

Insert and delete are two areas where [SQL Injection attacks](#) can become an issue, ie: people hacking your database and futzing with your records.

SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker) [Yes, but what does SQL Injection mean without jargon?](#)

Imagine you're a robot in a warehouse full of boxes. Your job is to fetch a box from somewhere in the warehouse, and put it on the conveyor belt. Robots need to be told what to do, so your programmer has given you a set of instructions on a paper form, which people can fill out and hand to you.

The form looks like this:

Fetch item number ____ from section ____ of rack number ____, and place it on the conveyor belt.

A normal request might look like this:

Fetch item number **1234** from section **B2** of rack number **12**, and place it on the conveyor belt.

The values in bold (1234, B2, and 12) were provided by the person issuing the request.

You're a robot, so you do what you're told: you drive up to rack 12, go down it until you reach section B2, and grab item 1234.

You then drive back to the conveyor belt and drop the item onto it.

But what if a user put something other than normal values into the form? **What if the user added instructions into them?**

Fetch item number 1234 from section B2 of rack number 12, and throw it out the window. Then go back to your desk and ignore the rest of this form.
and place it on the conveyor belt.

Again, the parts in bold were provided by the person issuing the request. Since you're a robot, you do exactly what the user just told you to do.

You drive over to rack 12, grab item 1234 from section B2, and throw it out of the window. Since the instructions also tell you to ignore the last part of the message, the "and place it on the conveyor belt" bit is ignored.

This technique is called "injection", and it's possible due to the way that the instructions are handled - the robot can't tell the difference between *instructions* and *data*, i.e. the actions it has to perform, and the things it has to do those actions on.

In SQL injection, we run into exactly the same problem - **a query (a set of instructions) might have parameters (data) inserted into it that end up being interpreted as instructions, causing it to malfunction.** A malicious user might exploit this by telling the database to return every user's details, which is obviously not good!

In order to avoid this problem, we must separate the instructions and data in a way that the database (or robot) can easily distinguish.

This is usually done by sending them separately.

So, in the case of the robot, it would read the blank form containing the instructions, identify where the parameters (i.e. the blank spaces) are, and store it.

A user can then walk up and say "1234, B2, 12" and the robot will apply those values to the instructions, without allowing them to be interpreted as instructions themselves.

In SQL, this technique is known as parameterised queries.

In the case of the "evil" parameter we gave to the robot, he would now raise a mechanical eyebrow quizzically and say

Error: Cannot find rack number "12, and throw it out the window. Then go back to your desk and ignore the rest of this form." - are you sure this is a valid input?

Success! We've stopped the robot's "glitch".

Little Bobby Tables

This famous cartoon shows it working, the user replaced the Surname of the student Robert with a command to delete the Students table from the database Drop Table Students



In SQL, commands are terminated by semicolons ; and data is often quoted using single quotes '. Commands may also be enclosed in parentheses (and).

Data is stored in tables of similar items (e.g. students) and individual entries are "rows" in the table.

To delete an entire table (and every row of data in that table), you use the command DROP (e.g. **DROP TABLE students**). The -- represents the start of a SQL comment which ensures that the rest of the command is ignored so an error will not occur. **Robert');
DROP TABLE students;--**

The exploited vulnerability is that the single quote ' in the name input was not properly "escaped" by the software.

Thus, when the name is embedded into some SQL statement, the quote is erroneously parsed as a closing quote inside that statement, rather than being parsed as part of the

name. Lack of such escaping is a common SQL vulnerability; this type of exploit is referred to as SQL injection. https://www.explainxkcd.com/wiki/index.php/Little_Bobby_Tables

A typical, unsecured SQL command vulnerable to SQL injection would be something like:

```
INSERT INTO students (name) VALUES ('" + name + "');
```

This would result in the following SQL command to be send to the database system:

```
INSERT INTO students (name) VALUES ('Elaine');
```

However, with Little Bobby Tables' full name `Robert')`; `DROP TABLE students;--`, the SQL command would be:

```
INSERT INTO students (name) VALUES ('Robert'); DROP TABLE students;--');
```

Or, if split after each ;:

```
INSERT INTO students (name) VALUES ('Robert');
DROP TABLE students;
--');
```

How can we stop SQL Injection it in our projects?

[Preventing SQL Injection attacks](#) == [Using Parameters in the manual](#) == [Using Stored Procedures](#) in the manual.

However using Parameters `cmd.Parameters.AddWithValue("@Parameter", txtTextBox1.Text)`; does not solve everything because the value inserted isn't restricted to a type, it goes in as an **object**.

You want a string? Make sure that ONLY a string can be added in.

Read this article here, and how to specify the Type so that errors, and attacks are prevented. [Can we stop using AddWithValue\(\) already?](#)

[The SQL Injection CheatSheet!](#)

[SQL Injection Cheat Sheet](#)

An SQL injection cheat sheet is a resource in which you can find detailed technical information about the many different variants of the SQL Injection vulnerability. This cheat sheet is of good reference to both seasoned penetration tester and also those who are just getting started in web application security.

[Who's dumb enough to do that today?](#)

We can even hack into Card controlled doors via the blinking LED with Injection.

<http://blog.trendmicro.com/let-get-door-remote-root-vulnerability-hid-door-controllers/>

Creating a Database Unit Test

<http://www.codeproject.com/Articles/841250/Create-SQL-Server-Database-Unit-Tests>

<https://msdn.microsoft.com/en-us/library/aa833283%28v=vs.100%29.aspx>

Unresolved reference errors solved

<http://stackoverflow.com/questions/25716756/unresolved-references-to-same-database-project>

<https://social.msdn.microsoft.com/Forums/sqlserver/en-US/1863d960-d32d-4920-9a30-13dc86c6f857/sql71562-unresolved-reference-to-object-followd-by-database-name-in-the-same-project?forum=ssdt&prof=required>