

ECE 4180 Lab 5: C/C++ and C# GUI application development in Windows and interfacing mbed to a PC

Sections A & B: Get checkoffs in lab or upload a *.doc or *.pdf file to Canvas (Instructions coming soon from TAs) with Screen captures for Checkoffs by Tuesday, April 2nd. A video link can also be provided in the doc, if needed in addition to screen shots (for use on extra credit options).

Name: _____ Sect _____ Name: _____ Sect _____

Item	TA Checkoff
Basic C/C++ Helloworld	
Mbed I/O using USB Serial port	
Basic C# Helloworld	
Basic C# Helloworld with Mbed LCD	
Extra Credit(2%) - RPC-based mbed LCD Display	
Extra Credit(1%) - Com port drop down combo box	
Extra Credit(1%) – Analog sensor on Trackbar	
Extra Credit(1%) – 4 Trackbars for LED dimming	
Extra Credit(1%) – Ethernet RPC-based LCD Display	
Extra Credit(1%) – Bluetooth RPC-based LCD Display	
Extra Credit(3%) – Wi Fi RPC-based LCD Display	

Final TA Checkoff and Grading _____

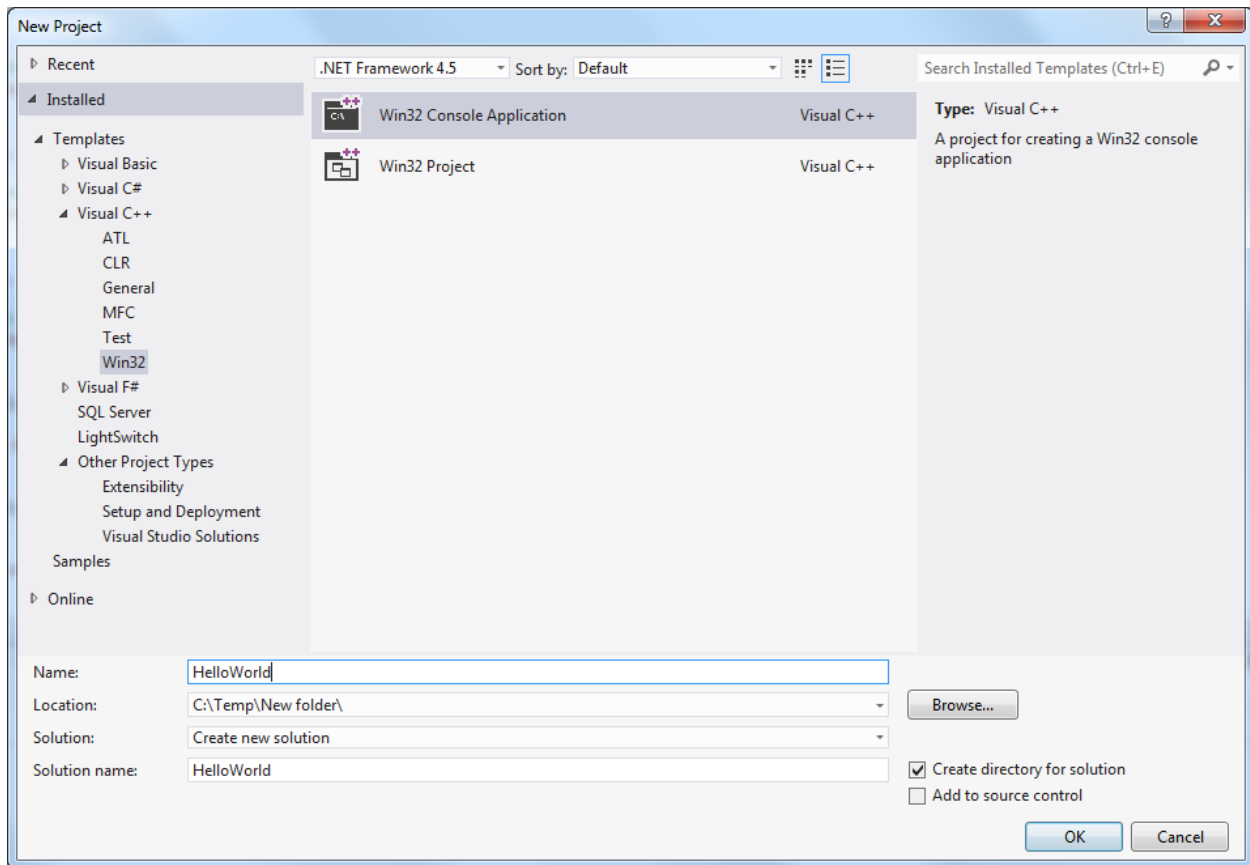
This lab has several short tutorials showing how to develop a few simple C/C++ and C# GUI and Serial I/O applications for PC-based hardware using Visual Studio. I/O devices include a USB virtual com/serial port on a PC. C# will then be used to setup a simple Windows GUI application. Next, a virtual com port will be used to transfer data to an mbed's LCD in response to a C# GUI button click.

Screen captures of the final output display can be used for the TA Checkoff for all parts other the mbed LCD Display Demos. Customize the data on your screen capture or video to include team member names to prove that your team did it.

A short Visual Studio tutorial on C/C++ application development

Start Visual Studio, select **File -> New Project** and then click C++ on the dialog window that opens and on the left first select **Visual C++->Win32** and then on the right **Win32 Console Application**. This sets up a project that reads and writes text to a command window (i.e., no Graphics or GUI).

Note: If you install your own version of Visual Studio and do not use the lab PCs older VS version, a few things are a bit different or moved around a bit in the GUI screen shots. Everything still works, but one big change is it is not **Win32**, but now shows up as **Windows Desktop** and then **Console Application** for a new project. The C++ windows first include file name also changes in the template code.

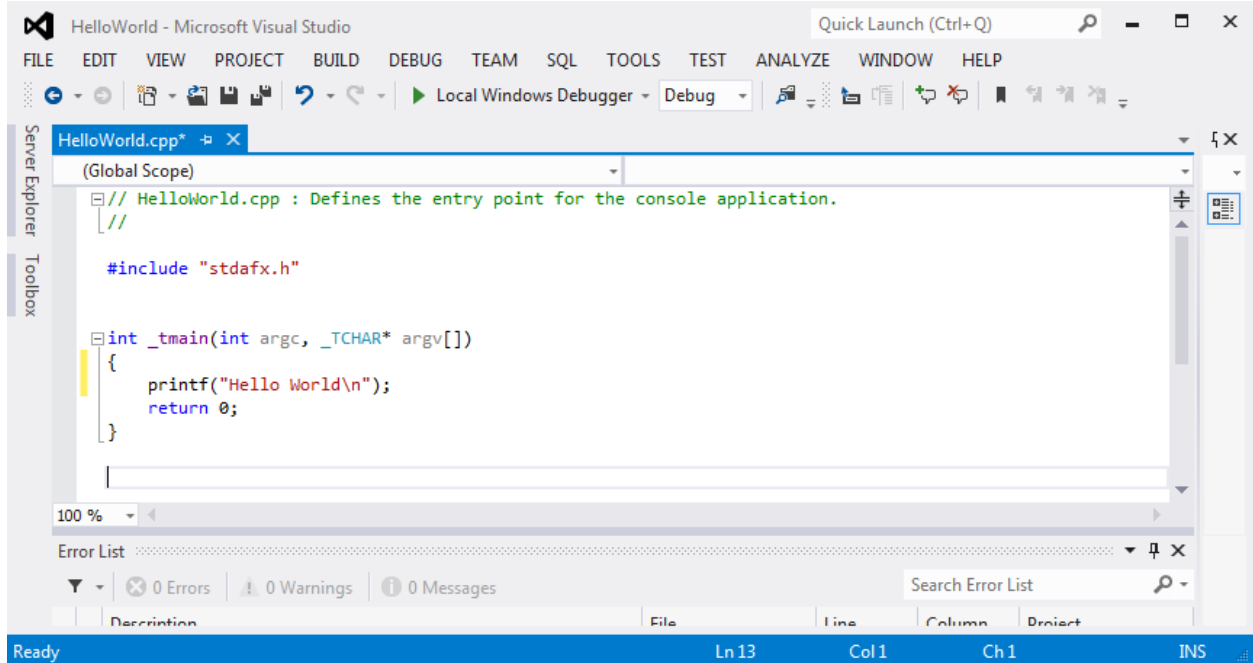


Setting up a Windows Text-only Console Application

Type **HelloWorld** for the project name and **click OK**.

Make sure you do not save the project in a net drive (prism and desktop) to avoid any building issue.

It creates a console application that automatically includes the basic C/C++ source code for HelloWorld. Type in the new **printf()** to the source code as seen below:



The screenshot displays the Microsoft Visual Studio IDE with a project named 'HelloWorld'. The main window shows the source file 'HelloWorld.cpp' with the following code:

```
// HelloWorld.cpp : Defines the entry point for the console application.
//

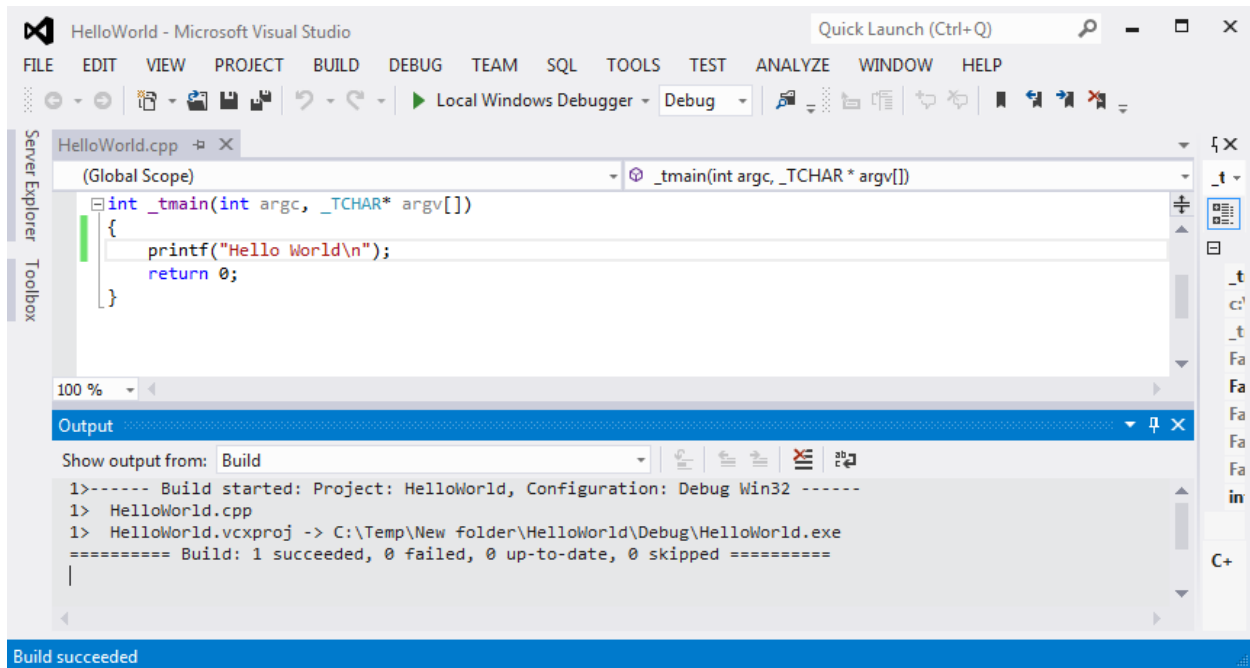
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    printf("Hello World\n");
    return 0;
}
```

The interface includes a menu bar (FILE, EDIT, VIEW, PROJECT, BUILD, DEBUG, TEAM, SQL, TOOLS, TEST, ANALYZE, WINDOW, HELP), a toolbar with icons for file operations and debugging, and a status bar at the bottom showing 'Ready', 'Ln 13', 'Col 1', 'Ch 1', and 'INS'. The 'Error List' panel at the bottom indicates '0 Errors', '0 Warnings', and '0 Messages'.

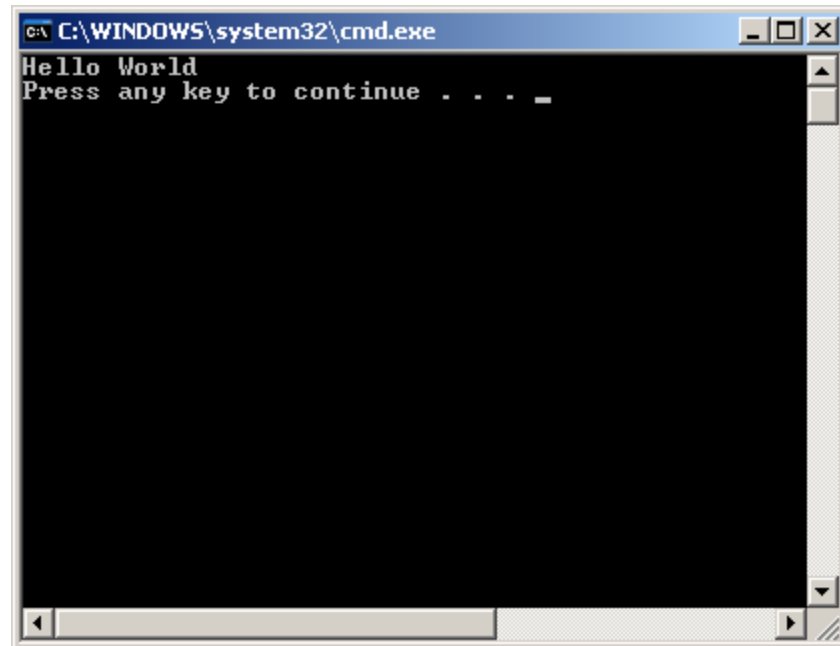
Source code for a C/C++ Hello World Console Application

Next to compile and link it, click **Build -> Build Solution**. It should compile and link (ie., build) with no errors (i.e, this is called succeeded in the output window) as seen in the output error log window at the bottom. In addition to breakpoints, there are a number of more advanced debug features that can also be used.



Compiling and linking a C/C++ Application Program

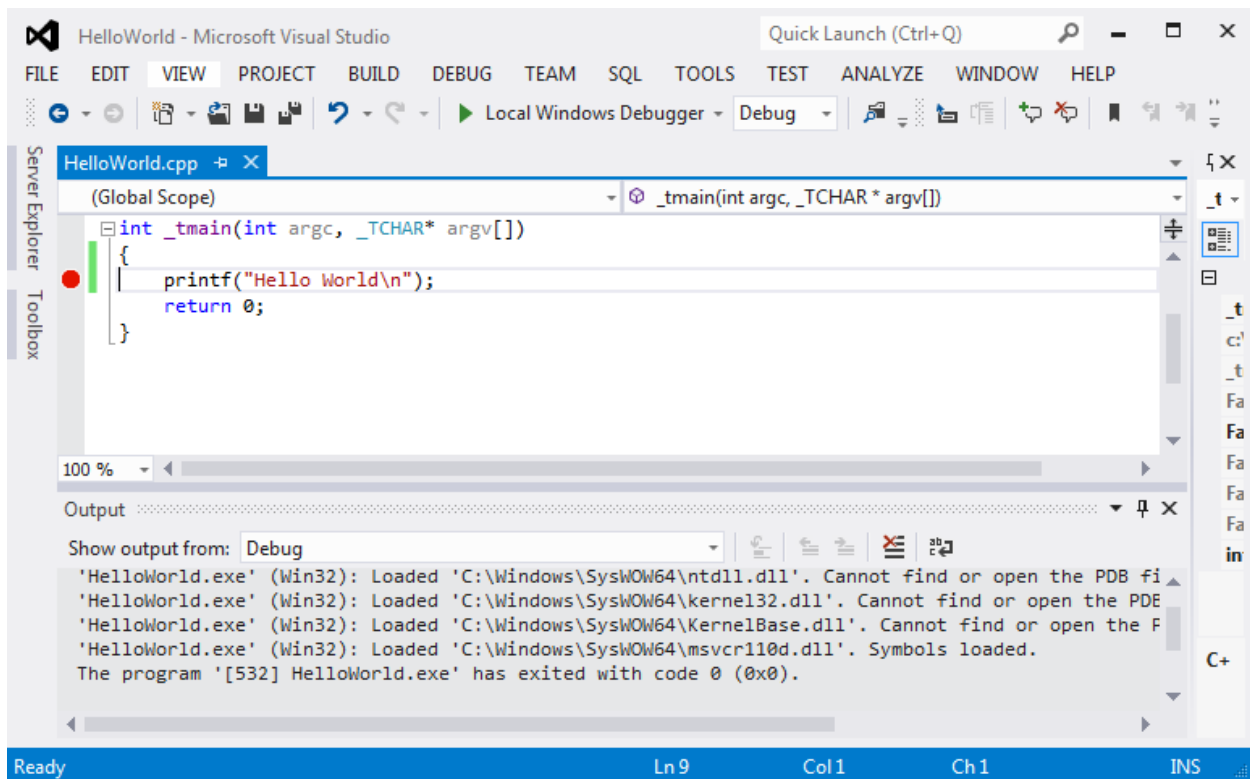
Now then program can be run and debugged. Select **Debug - > Start without debugging** to run the program. A command window with the HelloWorld message will pop up as seen below.



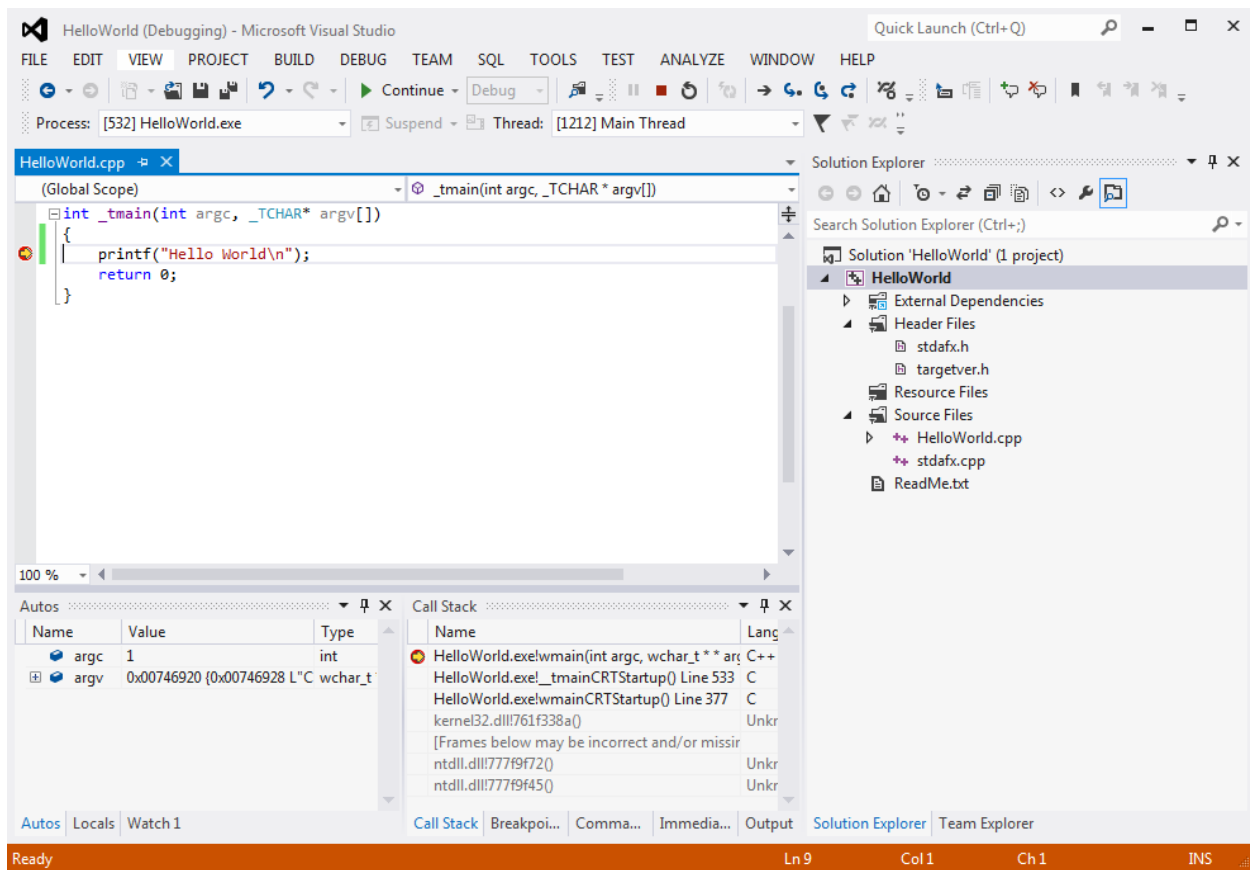
Console output window from the HelloWorld C/C++ application

Select the console window and hit any key to exit and return to the VS window.

Next back in the VS edit window set a breakpoint on a source line in the editor window by **clicking to the left of the line** with the mouse. A red dot should appear that indicates a breakpoint has been set as seen below.



Run the program again using **Debug -> Start Debugging** and the program should run until it hits the breakpoint and stops. When it stops, note that a yellow arrow appears in the red dot showing which breakpoint was encountered.

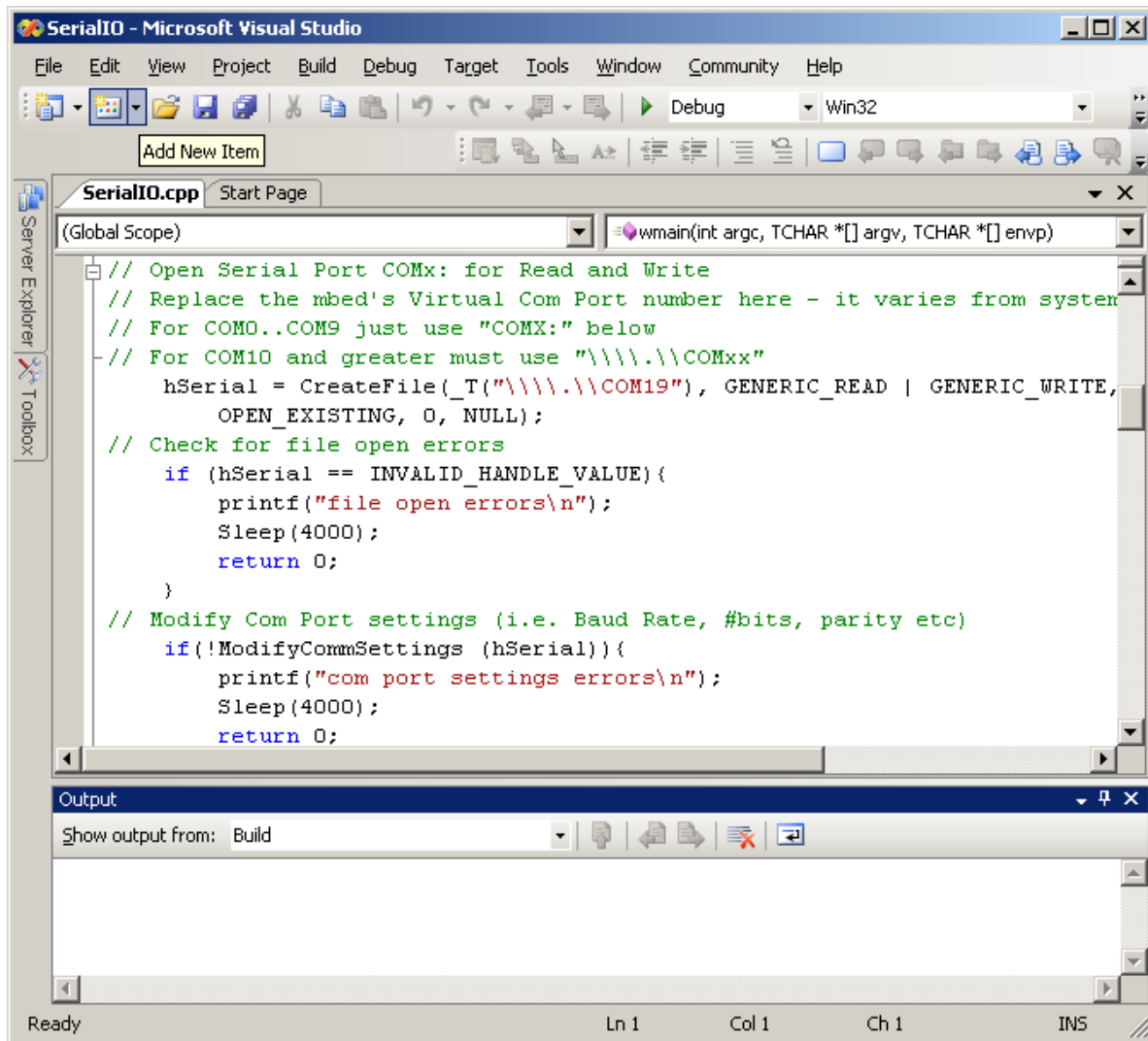


Using the serial port to communicate with I/O devices.

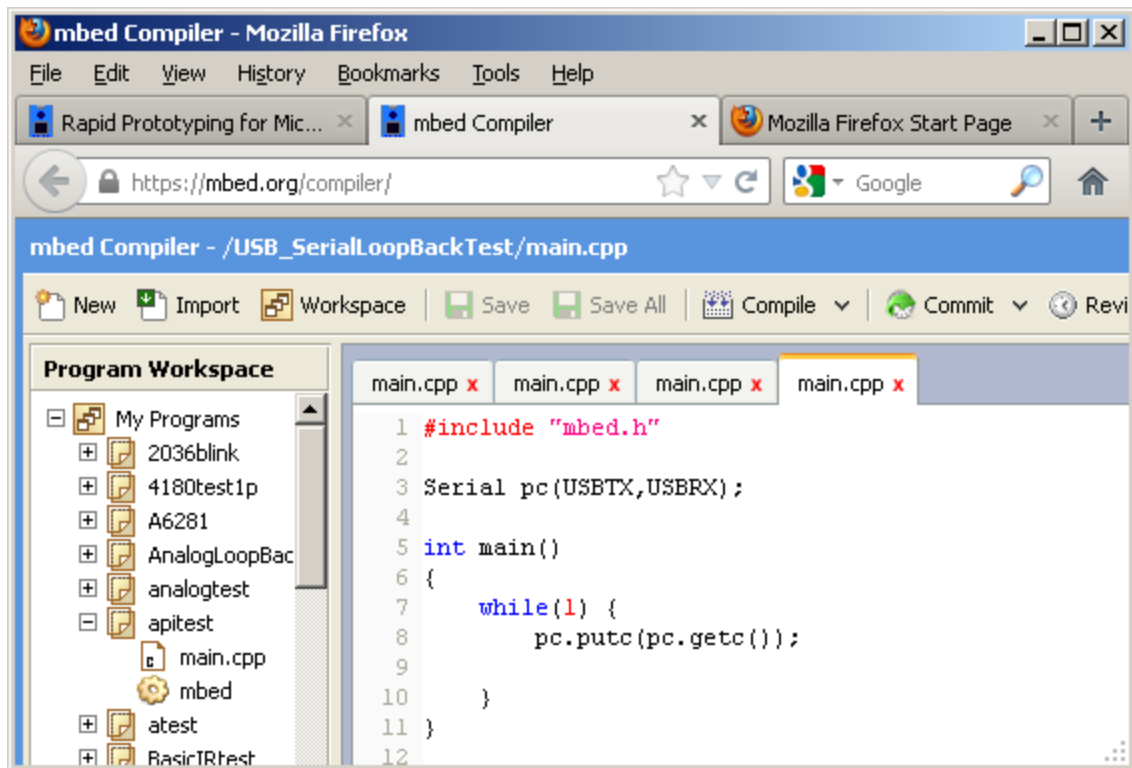
Many I/O devices interface using a serial port or COM port on the PC. Examples include GSM modems, GPS units, Magnetic card readers, Barcode readers and the mbed's USB com port.

Mbed also uses a virtual com port via the USB cable, but from a software perspective it operates the same as a real COM port. Note the COM port number that the mbed chip is using on the PC by checking it with a terminal application program while the mbed is plugged into a USB port. It will be needed before the next step and recall that the number changes whenever it is moved to a different PC.

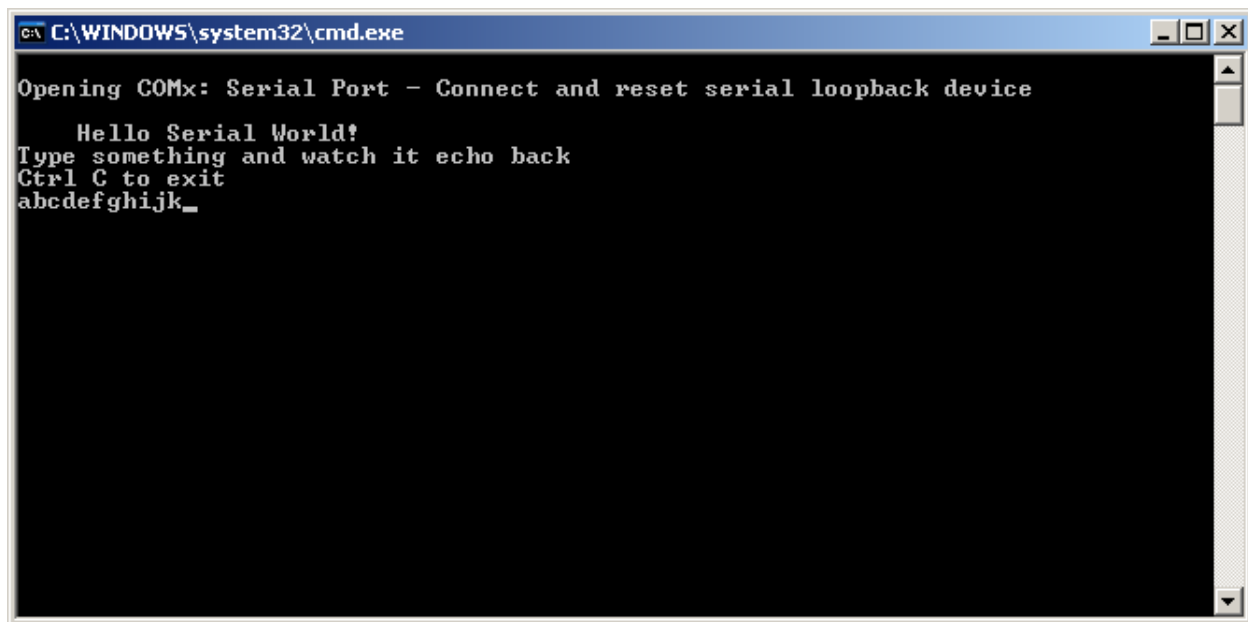
An example project for VS2005 called "Serial" has been setup that contains code to send data out on the serial port and read it back in. Download a copy of the project directory from www.ece.gatech.edu/~hamblen/489X/classmat/Labs/serial.zip and unzip it to the C:/temp directory. Open the project, and in the source code, edit the entry for the com port by typing in the number of your local mbed as seen below (with your COM number). Finally, rebuild the project. If the first build gets an error, it may be necessary to clean the project and then rebuild it. This sometimes happens when switching project files between different versions of VS. Going to a newer version typically works despite possible version warnings for something this simple, but moving from a newer to an older version likely will not.



For the test program to work, the mbed should be running code that reads in serial data and sends it back (i.e. called a serial loopback). The code needed is shown below:



After building the project with the COM port change. Start the program with **Debug->Start without Debugging**. If the correct COM port change has been made and the mbed is attached, the COM port should open with errors and wait for user input. Type a few test characters in the window as seen in the next screen shot.

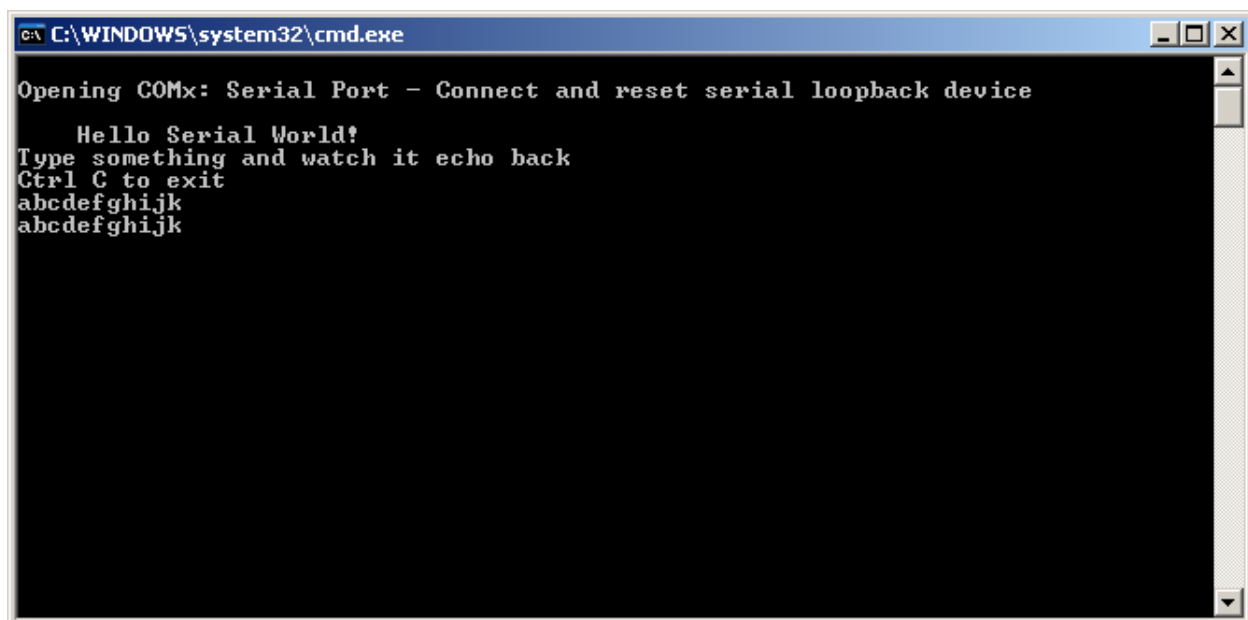


```
C:\WINDOWS\system32\cmd.exe

Opening COMx: Serial Port - Connect and reset serial loopback device

    Hello Serial World!
Type something and watch it echo back
Ctrl C to exit
abcdefghijkl_
```

Hit return, and the line will be sent over USB to mbed. The mbed program sends it back (echoes) and the same line should appear in the console window.



```
C:\WINDOWS\system32\cmd.exe

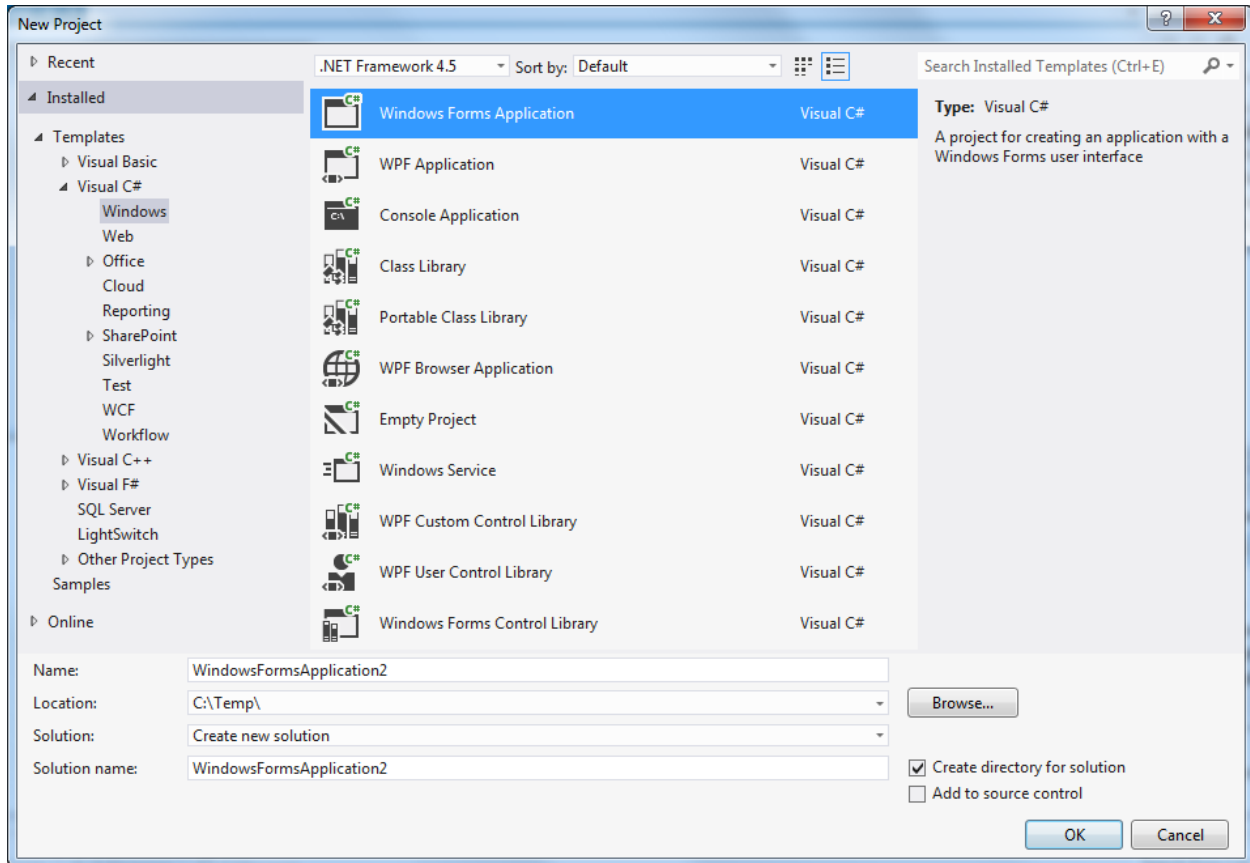
Opening COMx: Serial Port - Connect and reset serial loopback device

    Hello Serial World!
Type something and watch it echo back
Ctrl C to exit
abcdefghijkl
abcdefghijkl
```

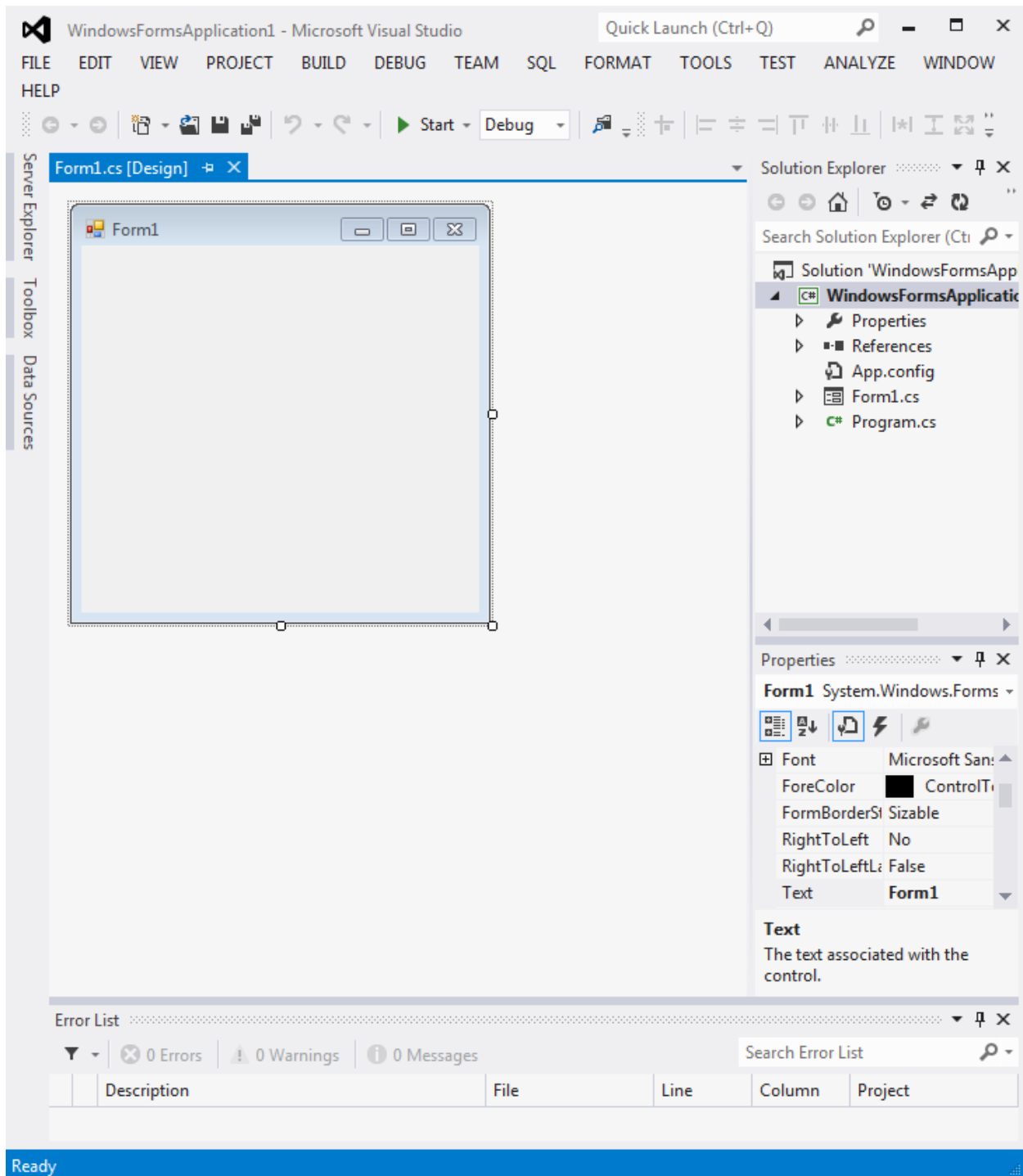
Type Ctl C and then another character to exit and close the window. This program showed how to use serial port to transfer data between a PC and mbed. It might come in handy for any serial devices in the final design project. For the screen capture, type in your team member names.

Developing a C# Windows Application in Visual Studio

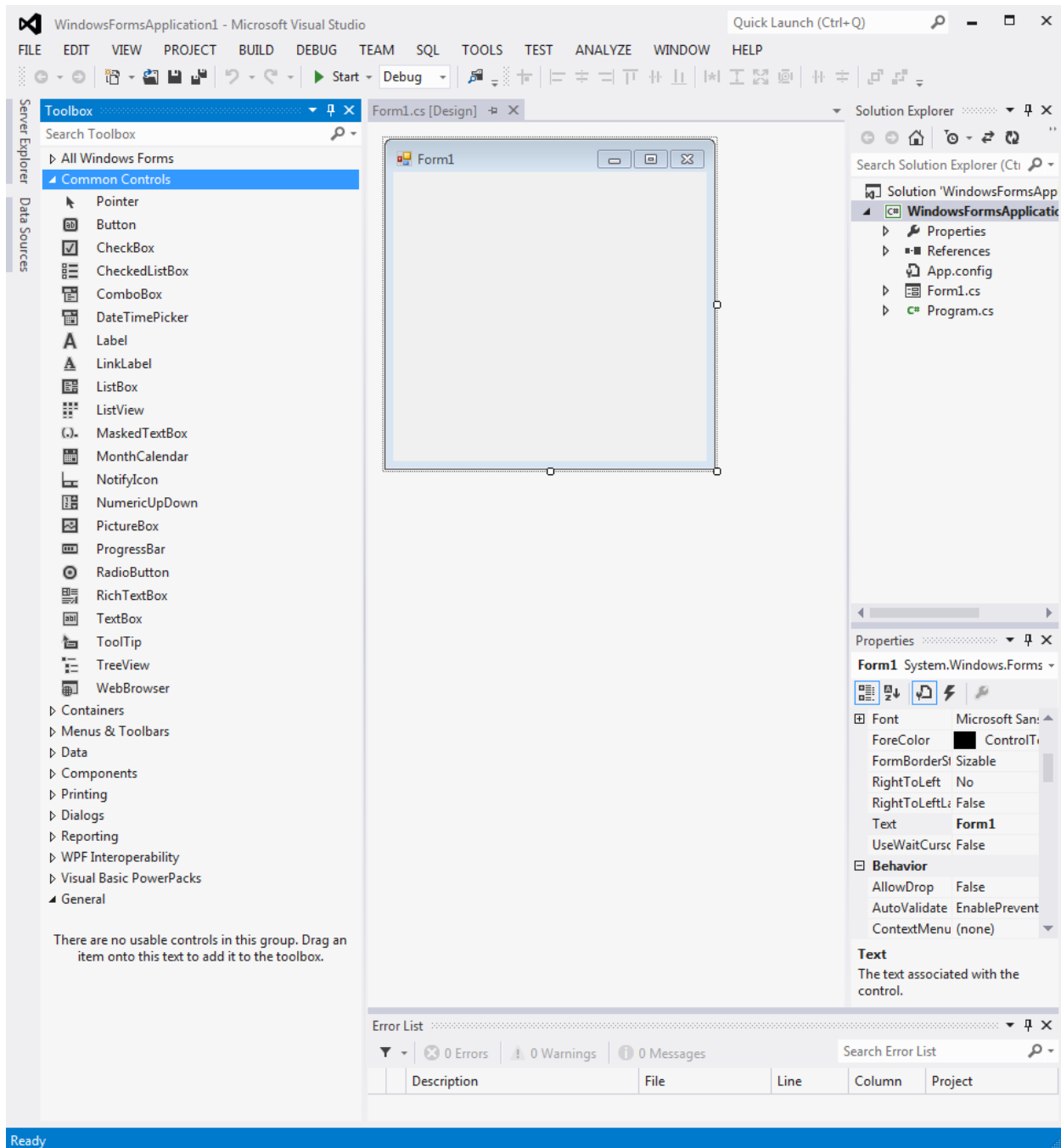
Start Visual Studio and as seen below select Visual C# Window project type (left) with a Windows Forms Application Template (right).



Click OK and a new project will be setup in the graphical forms editor with a blank window as seen in the next screen capture.

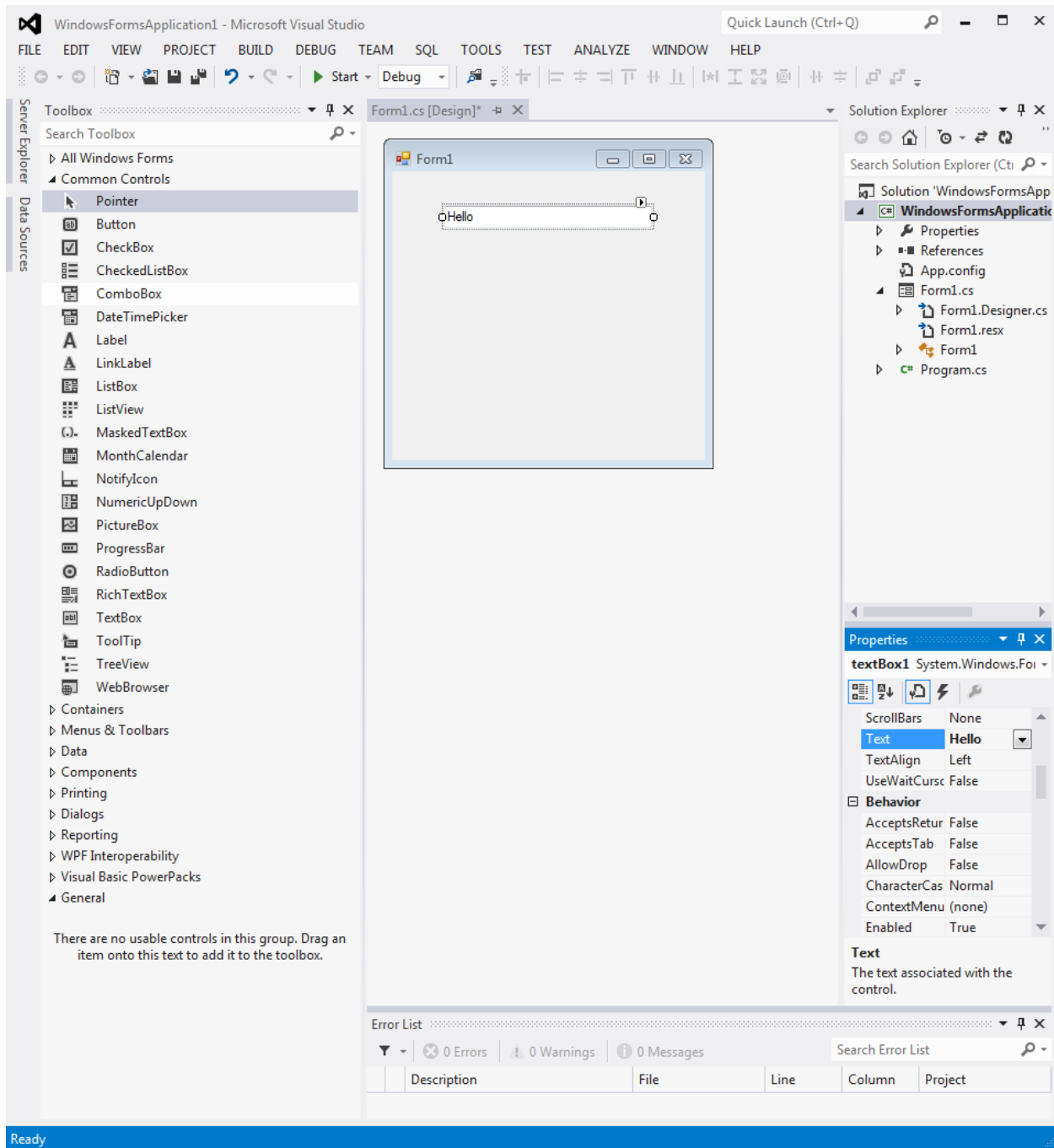


Select **View-> Toolbox** and a new window should appear as seen below



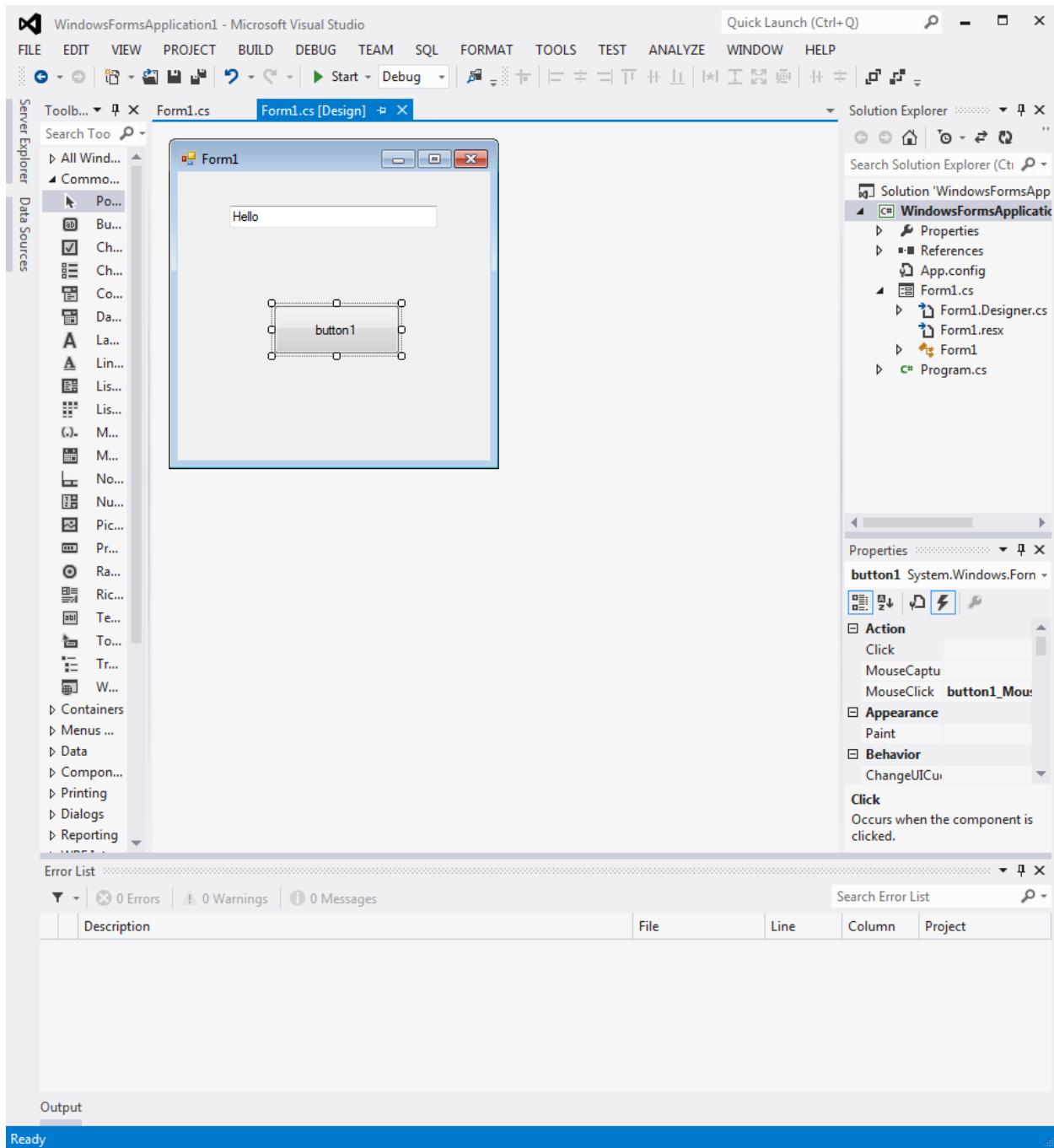
Select **Common Controls**, Scroll down and find **“textbox”**. Add a textbox to the forms drawing using the mouse. **Right click on the** textbox just drawn and select properties. Enter **“Hello”** for the Text property associated with the new textbox.

It should look like the image below.



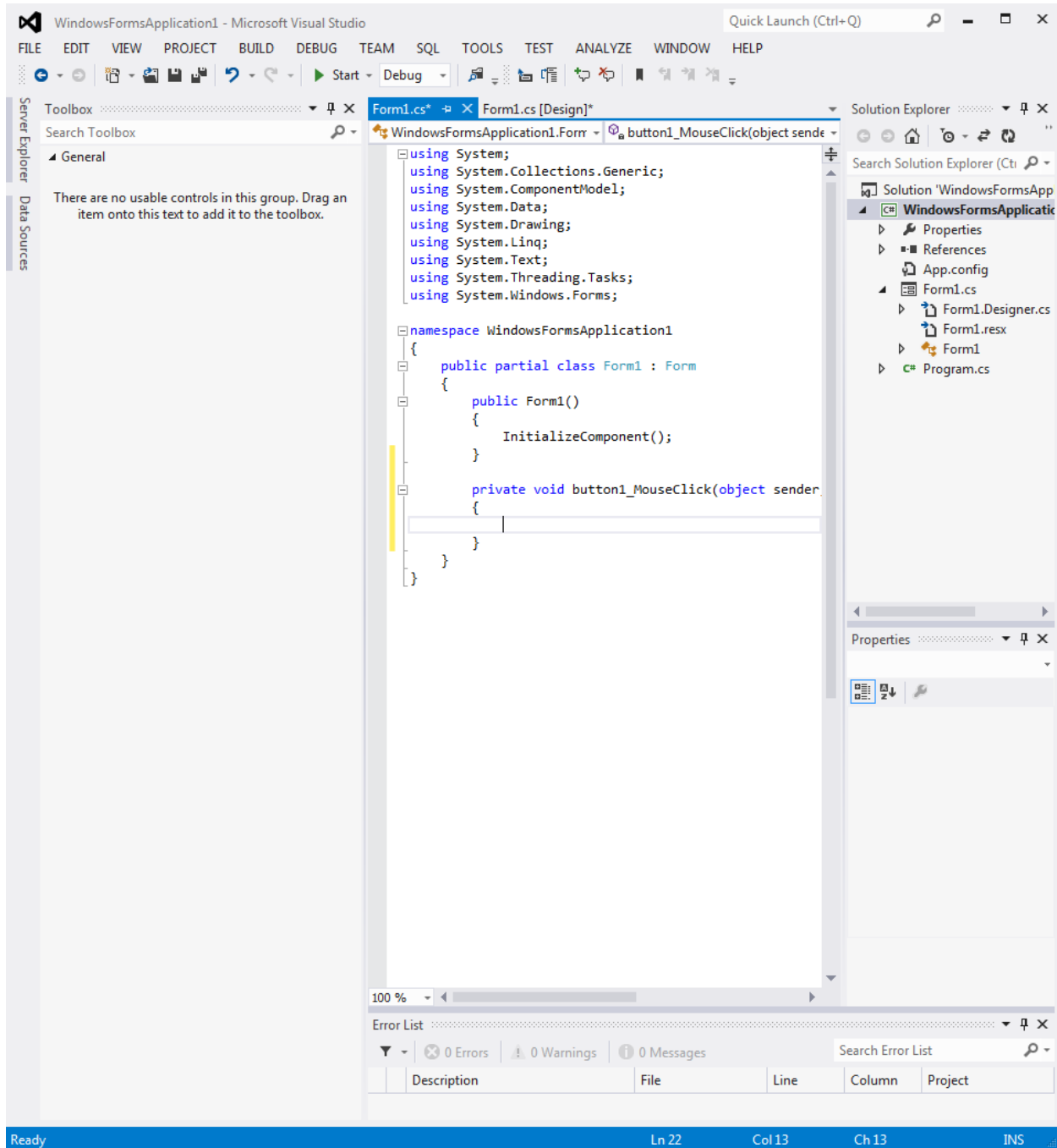
Click on the toolbox side tab and select button.

Add a button and it should look like the following image.

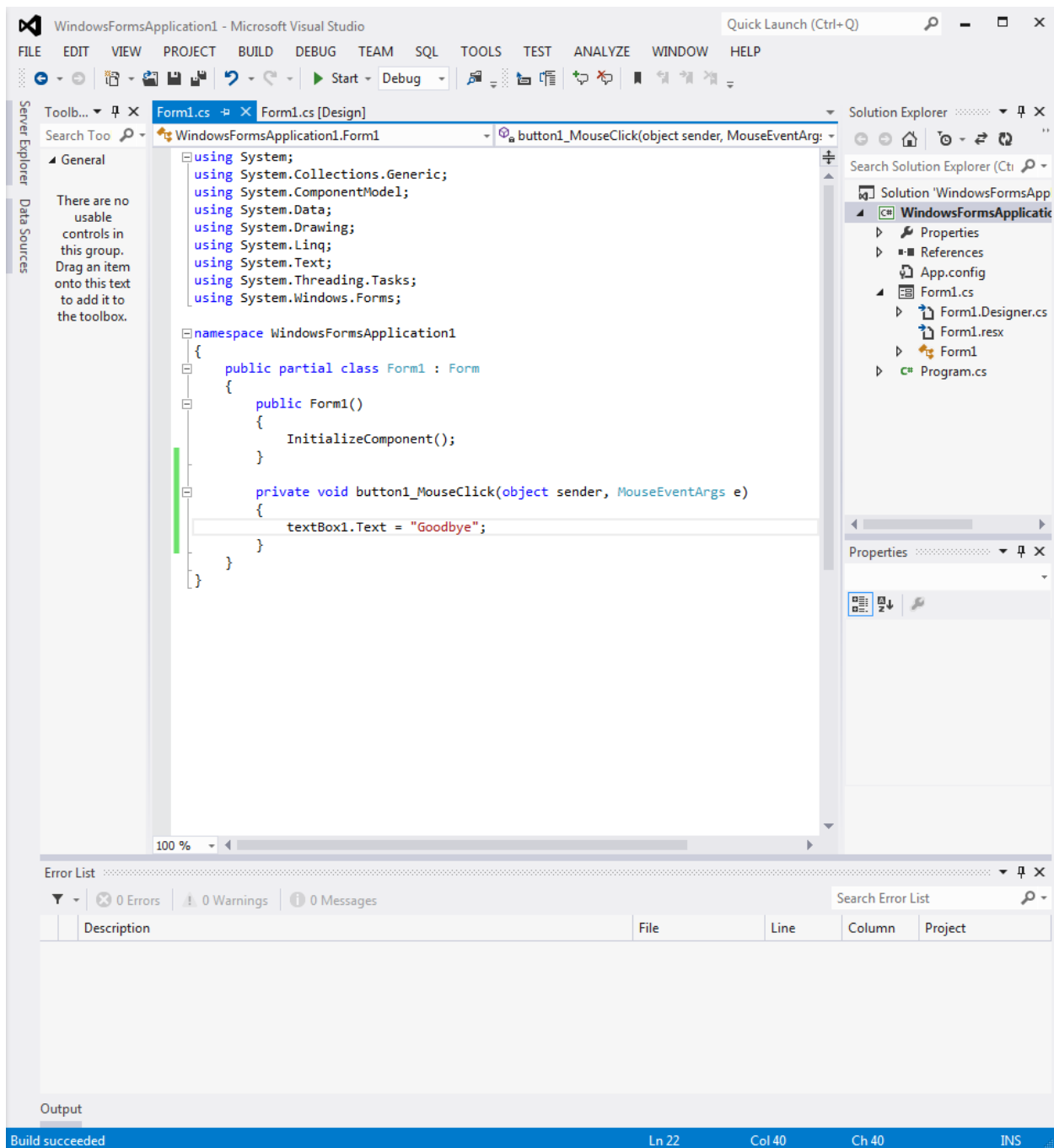


Next, notice the lightning bolt icon in the properties window above. **Click the lightning bolt, double click on “click” and watch out!**

The C# forms wizard automatically takes you to the code that executes when a button click occurs as seen below.

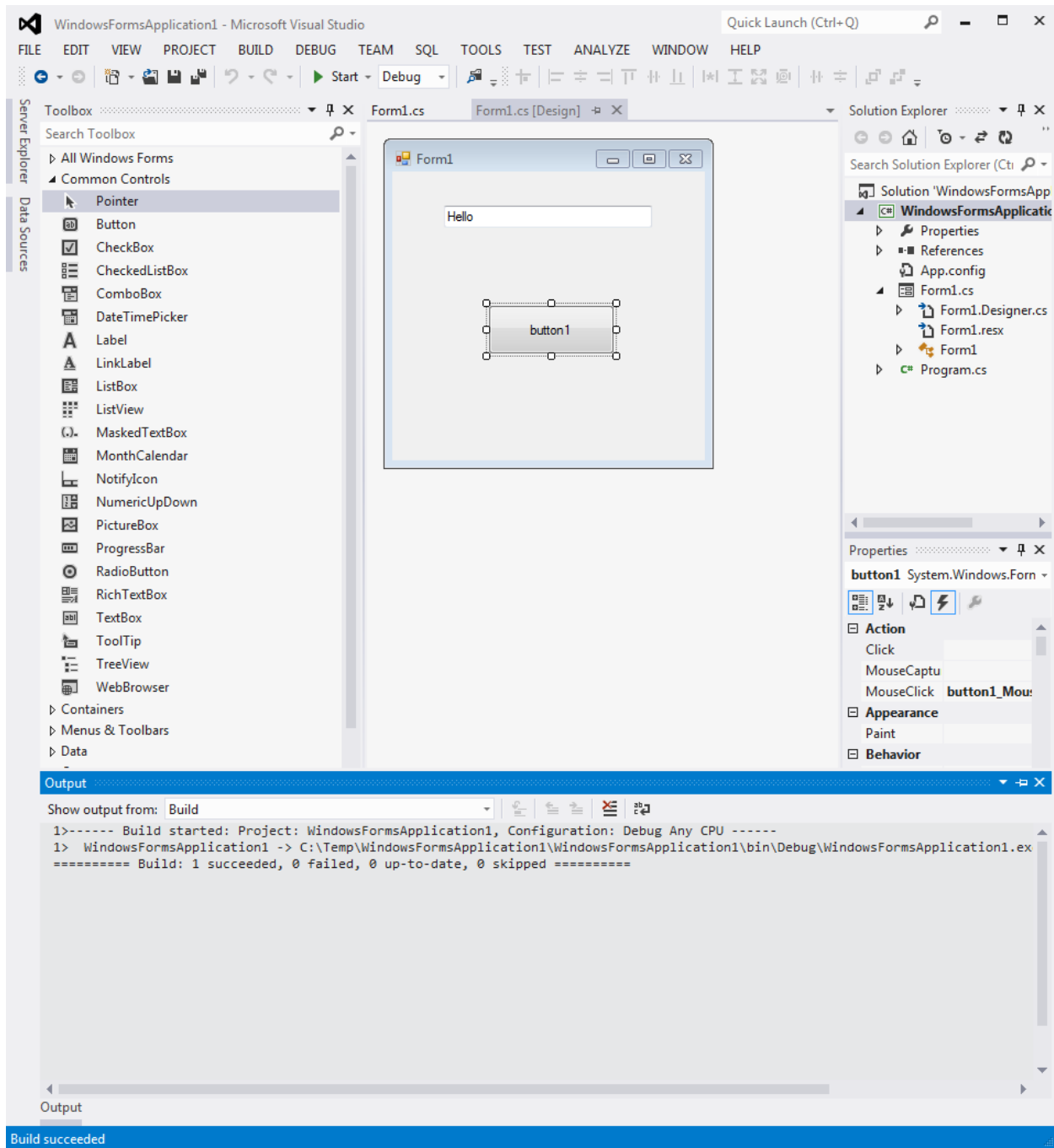


Type in a line of code to execute when the button is hit that changes the text in the text box as seen in the next screen capture



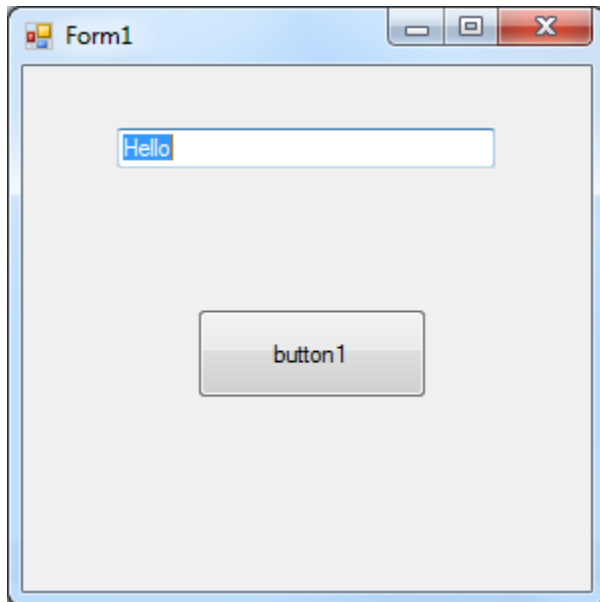
This code should change the textbox message to “Goodbye” when the button is hit. Next at the top menu. **Select Build->Build Solution.**

Your new C# windows application should build as seen below.

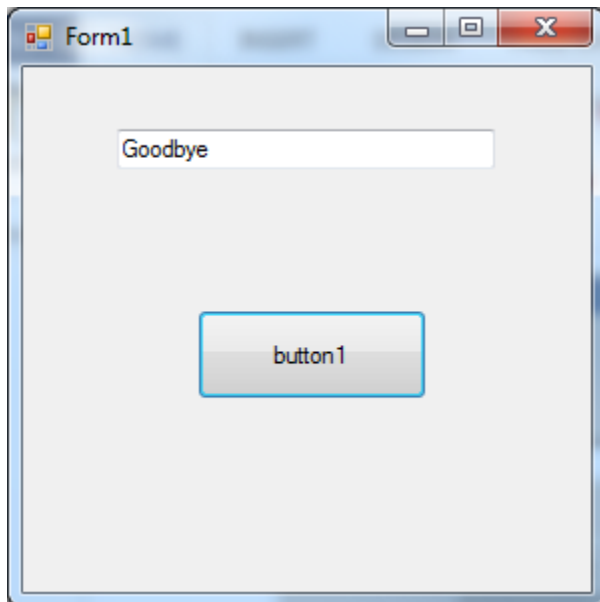


It is now ready to run. **Select Debug->Start without debugging.**

The window below should appear.



Click the **button** and you should see the text change to



Click on the **right corner X** to exit the application and delete the window.

Interfacing mbed to a C# GUI application using the USB virtual com port

Write an mbed C++ application that displays incoming ASCII data from the USB virtual com port and prints it out on the mbed's uLCD (default size and color is OK). Check for correct operation using a terminal application window. Make a note of the com port number needed. Leave this code running on the mbed.

Modify your C# code from the previous example, so that it also sends out the text box characters to mbed using the USB virtual com port for display on the uLCD whenever the GUI button is pushed. The USB virtual com port must be opened first at the correct baud rate. The tool box in VS contains a "SerialPort" tool that can be used to add a serial port to a C# application. Drop one anywhere in the GUI box. The properties of the serial port (click the serialPort icon below GUI image) can be edited to set baud rate and com port numbers. The C# serialport code from the 4180 text example can also be typed in manually in text mode, but it is a bit tricky to get it in the correct C# (*.cs) source files, so that it opens before a write and closes correctly. The code to write out the text to the serial port will need to be manually added to the button's click event code. There is another example in the text that uses a GUI to control a Roomba Create with serial character code that is sent out whenever a button is clicked, if you need additional C# serial I/O examples. Send out your team member names to the LCD and include a picture of the LCD and the C# GUI screen capture in your doc.

Once this application is working on the development PC, demo it to the TA. It is also possible to move the C# application's object files to a USB flash drive and then run it on one of the small embedded Atom PCs in the lab, if you need this for a small robot or GUI design project later.

It should also be possible to re-build projects with more or less the same source code for the Raspberry Pi 2/3 running Windows IoT (after installing the IoT add-on to Visual Studio and [Windows IoT on the Pi](#)) if Ethernet or Wi Fi is used for mbed communication. The mbed USB virtual com (serial) port driver needed for Windows IoT is most likely not available and would have to be written (a major task!).

This type of LCD setup can come in handy as a basic status display for embedded devices that do not have an LCD monitor (i.e., a "headless device"). Small robots are one such example (too small to carry around a monitor on the robot). For faster transfers, a higher baudrate than 9600 would normally be used.

This same approach can be used to add custom I/O hardware and sensors to the PC by writing an mbed application to get/send data to the new hardware and transferring it over the USB virtual com port. Ethernet networking or Wi Fi can also be used, if the mbed needs to be in a remote location.

There is also a more general approach that can be used that might come in handy for more complex projects. It is called a [remote procedure call](#) (RPC). There is a [RPC library for mbed](#). The mbed's class/object or member function name and arguments can be sent over a communication channel to mbed (USB or network) and mbed returns the value.

There is a somewhat out of date [example showing how this works in C#](#) (i.e., .NET), but it used an older version of the mbed RPC library and it does not work with the new official mbed RPC library. The older version of the library sends a bit different data for the constructor and returns and object number, while the new library uses the actual C++ object names (unless someone fixed the .Net wiki in the last six months!). To add to the confusion, the new and old RPC libraries have the same name and when a new project is downloaded libraries will sometimes auto update. To debug an application using serial RPC, there is a handy [free serial port monitor program](#) that will show all of the incoming and outgoing COM port traffic on a PC without adding a lot of debug printf's. There is a new version of code for serial RPC at <https://developer.mbed.org/users/MichaelW/code/RPCInterface/rev/bcc2e05e5da4> that works. There is a [new lab 4 student wiki page](#) just out with fixes for C# .NET RPCs using the new RPC library that should help a lot.

It would be possible to do just about anything with Internet RPCs on mbed from anywhere in the world (i.e., an Internet of Things (IoT) device approach), but keep in mind that it is a lot slower than doing everything locally in a program with all of the added round-trip communications delays needed for each RPC.

Extra Credit (2%) Use the new (or old) mbed RPC library running on mbed and an RPC call from C# to display the data on the uLCD instead of the earlier approach. Make an uLCD object with a printf member function/method that does this that would be easy to use anywhere in C# code. This could be called by serial port adhoc command code on mbed or the more general RPC feature that allows new objects to be [setup for RPC in a wrapper](#) (assuming the first extra credit RPC option is working). The C# examples at <https://developer.mbed.org/cookbook/DotNET> seem to be for the older version of the RPC library. Here is a new RPC tutorial which has been updated <https://os.mbed.com/users/nambvarun/notebook/rpc-tutorial/>. There is also a new version of Windows C# mbed RPC code setup for serial RPC at <https://developer.mbed.org/users/MichaelW/code/RPCInterface/rev/bcc2e05e5da4> that works and <https://developer.mbed.org/users/kennyainny/notebook/mbed-net-rpc-library---revised/> uses new version of RPC library. It has a demo that sets up a C# GUI on PC and controls mbed using RPC over USB virtual com port.

Extra Credit (1%) Short C# code examples can be found on the web that will scan for an array of available com port names. Add a drop down combo box that shows the valid com port options and have the user select the one to use (i.e., somewhat like TeraTerm or RealTerm works). This way the C# source code does not need to be edited and recompiled whenever the mbed com port number changes on a different PC.

Extra Credit (1%) Send the data from your favorite analog sensor from mbed to the PC. Use a “trackbar” with appropriate labels in the PC’s GUI to display the value of the analog sensor. See the C# Create remote project in the text and the text’s example project source files that can be downloaded to see how to setup a serial port DataReceived event handler (find start button and port_DataReceived code in Form1.cs). A “timer” tool could also be used to update the GUI trackbar value at a slower rate like the example, but this is optional.

Extra Credit (1%) To the previous GUI trackbar analog sensor input extra credit option, add 4 output trackbars that individually dim (control) one each of the mbed’s four built-in LEDs using PWM. Once several serial commands are needed, this will likely be easier if the RPC approach is used, but ad hoc is also allowed.

Extra Credit (1%) Instead of using RPC over Serial, repeat the first extra credit option using Ethernet with the RPC-HTTP example program from the RPC cookbook page at https://developer.mbed.org/users/sarahmarshy/code/RPC_HTTP/. The program can be demoed using a web browser (note main RPC wiki page has info on URL RPC syntax details). There are several ways to do simple HTTP Gets & Posts in C# with just a few lines of code since .NET has numerous web APIs. A [video demo](#) is even available.

Extra Credit (1%) Use RPC over Bluetooth to control the mbed LCD using the Bluetooth module in your kit. Use the Adafruit app UART text feature for the demo.

Extra Credit (3%) Do RPC to the LCD using the Wi Fi chip. Modify the RPC-HTTP example to work with the Huzzah Wi Fi chip in your parts kits. On your own on this option, don’t have any wiki pages yet on it.