CS253 Laboratory session 4

Part 1: Disassembling code, going backwards, converting an executable back to Assembly Language.

Preamble: Remember that whatever language you are using ultimately it runs as Machine Code on the processor. The Machine Code is just a list of binary numbers (bytes) contained in an executable file. We can convert this (or any) executable file back into Machine Code using a piece of software called a disassembler. We can't go back to the original source code for a language such as C++ as all the labels and variable names are lost at compile time. That said we can convert an executable created by C++ into Assembly Language. We could in theory edit this Assembly Language and recompile it. Disassemblers provide a useful tool for the forensic analysis of code where only the executable is available.

Note: Java is a bit of an exception; its code is converted to a form of "byte code" which is a virtual (and universal) machine code that runs on the java virtual machine (JVM). The JVM is a program that runs on the processor and is written in machine code. C# (pronounced see-sharp) is another language that runs on a virtual machine called the Common Language Runtime, CLR is part of the Windows .NET Framework. C# has a bytecode called CIL, Common Intermediate Language. Your knowledge of Assembly Language would give you the ability to understand the byte code contained in the Java Run Time file called a .Class file.

The use of virtual machines improves security at the cost of performance. The virtual machine can manage access to local resources on the computer and so provide a safe environment for the code to run. Malicious code would find it difficult to extend its reach beyond the virtual machine. Since it is a safe place to run code it is sometimes called the "Sand box".

```
public void whileLoop()      0: iconst_0
{                            1: istore_1
    int i = 0;               2: iload_1
    while (i < 2)            3: iconst_2
    {                        4: if_icmpge     13
        i++;                 7: iinc          1, 1
    }                       10: goto          2
}                           13: return
```

Figure 1: Java code and the corresponding bytecode in .Class file that runs on the JVM.

Note: Code can be either Interpreted (converted line by line to machine code and run line by line) or Compiled (converted to a single machine code program and run in one go).

If you are building a standalone application to process video from a camera running at high speed you might choose a language such as C++ as it compiles to an executable that has direct access to all hardware resources on the computer, the code loads into memory once at the start and then runs very fast as machine code.

If you are developing a camera based web application you might choose Java as it provides means to run your code on any platform (via the JVM) at the cost of performance.

To do: I have taken the executable version of the three programs you have written recently and renamed them A, B and C. These programs were “Hello world”, “Typing tutor” and “Floating point calculation”.

In MASM by default there are 100h bytes allocated at the start of the program for the data segment, 100h for the stack and 17h bytes for .STARTUP code, so the new code in a typical executable starts at 217h bytes into the executable address space.

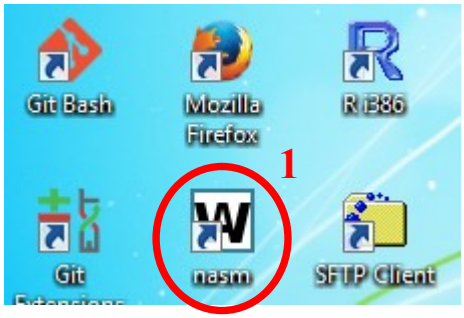
Copy the three files from the Zip file to a working directory (X:\CS253\DISAM or C:\TEMP) and then use the following command line to disassemble the code

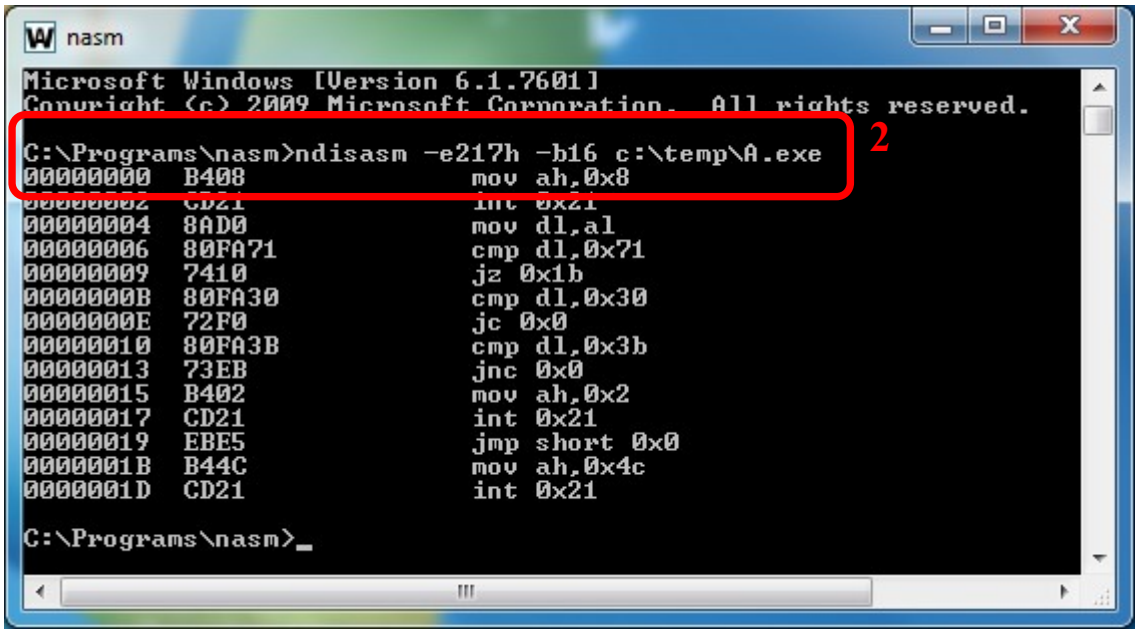
ndisasm –e217h –b16 c:\temp\A.exe

ndisasm –e217h –b16 c:\temp\B.exe

ndisasm –e217h –b16 c:\temp\C.exe

Look at each disassembly and identify which program is which, use the lab quiz to communicate your answer, **Q1**.





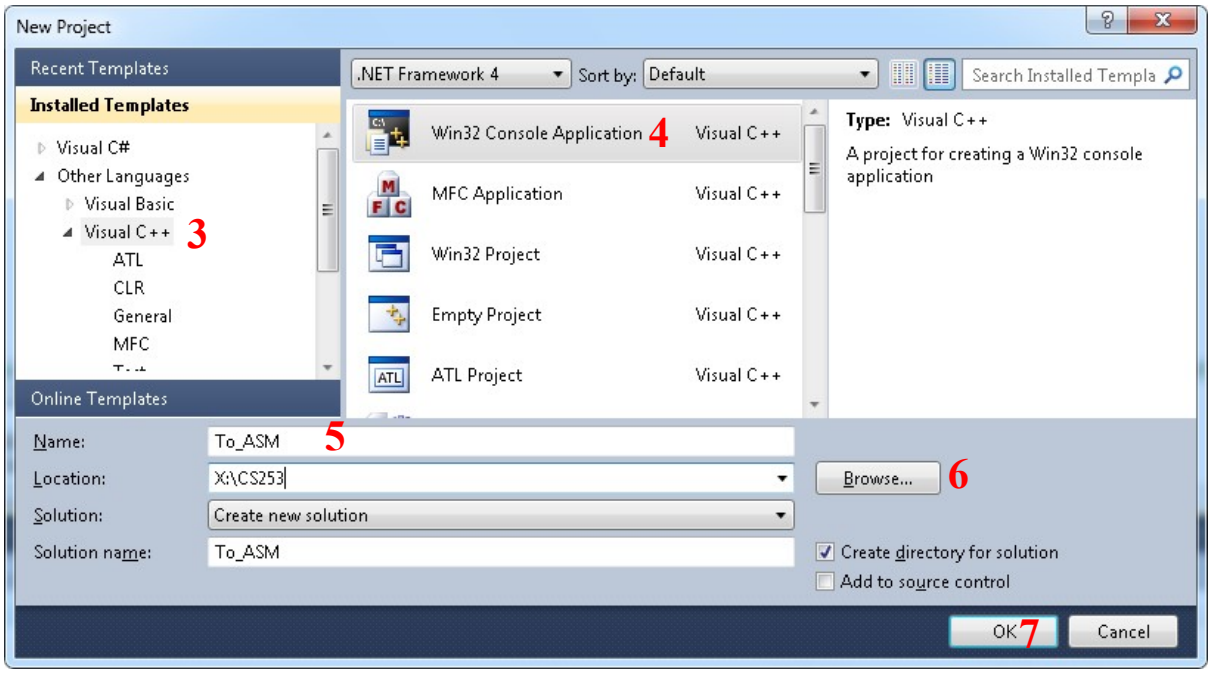
Aside: In this case you could cheat and just use DOSBox to run each program; you could try this to confirm your predication based on the disassembly.

A thought: You could take the disassembled code and recompile it. It may even be possible to change values in the executable file directly and resave it so as to change the properties of the program. We could perhaps change the typing tutor executable file so that it only echoed upper case characters rather than numeric characters by changing the operand values of the two *cmp* instructions, but that is for another day....

Part 2: Ask the compiler to generate Assembly Language as well as an Executable

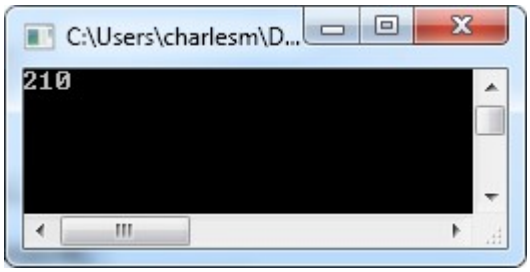
Preamble: Normally you ask the compiler to produce just an executable file of the code so that you can run it; however it is possible for you to tell the compiler to produce the Assembly Language file also. The aim of this part of the lab (and the previous section) is to help you see how assembly language, machine code, executable files and compiler output are related.

To do: Create a New Visual Studio C++ project called "To_ASM" and save it on either your X drive or C:\temp.

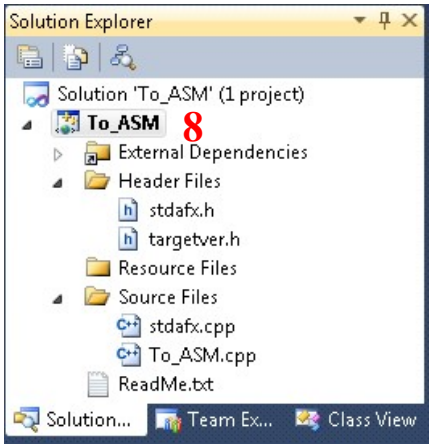


Modify the code provided so as to create a program that sums the integers from 0 to 20. Compile and run the code (in Debug not Release mode), it should display 210.

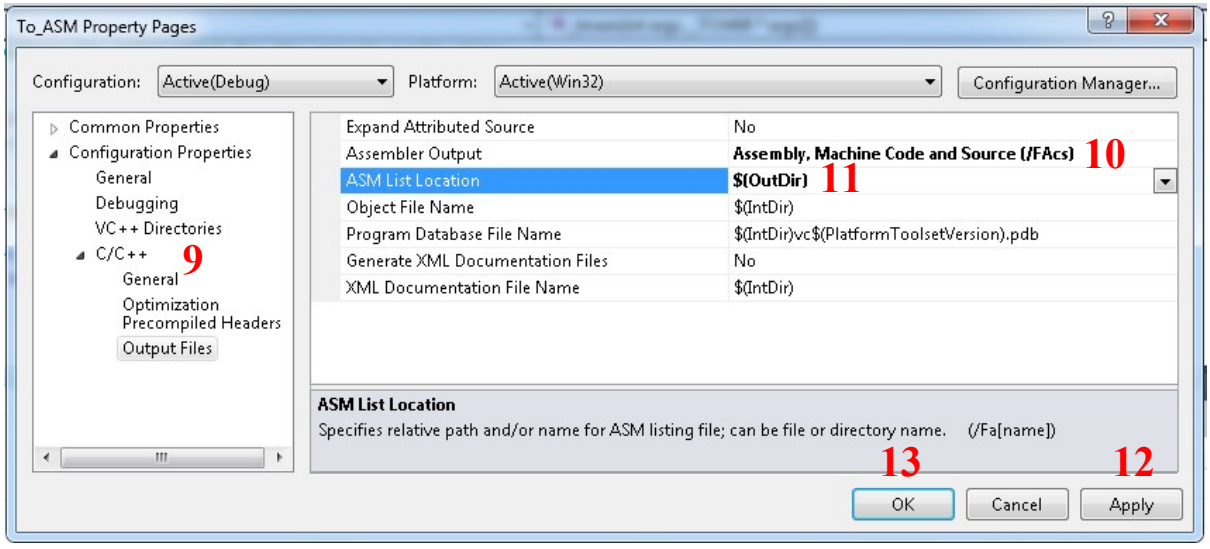
```
#include "stdafx.h"
#include <stdio.h>


int _tmain(int argc, _TCHAR* argv[])
{
	register int total=0;
	for(register int i=0;i<=20;i++)
	{
		total+=i;
	}
	printf("%d",total);
	getchar();
	return 0;
}
```



Right click on the *To_ASM* in the *Solution Explorer* and select Properties.

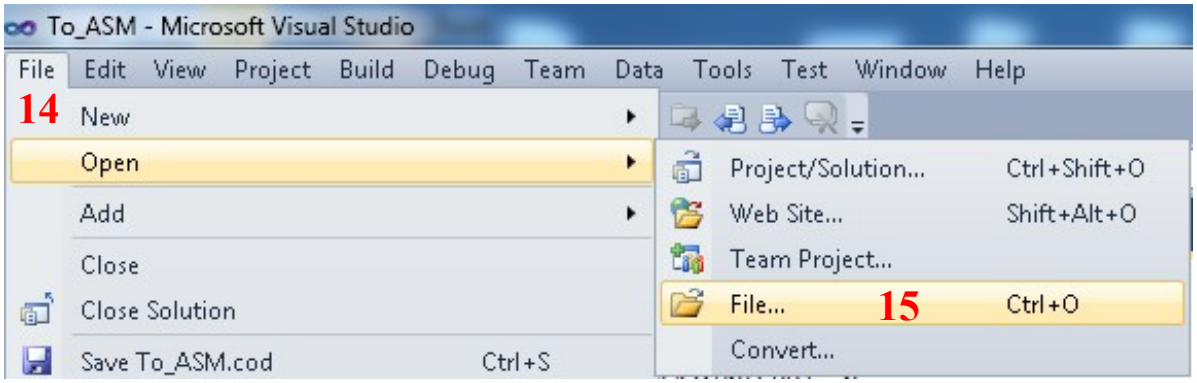


Change the Output files of the compiler to include Assembly, Machine Code and Source code and change the directory for this output to $(OutDir) so we can find it more easily after compilation.

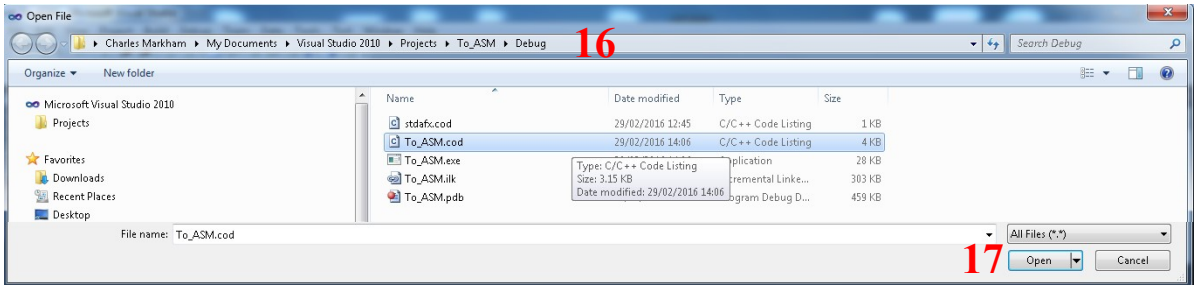


Run the program again so as to recreate the executable and also create an Assembly Language listing.

Use File on the Toolbar and select Open File and Navigate to the Debug folder (careful which one you go to as there are two in each solution folder), open the file *To_ASM.cod.*

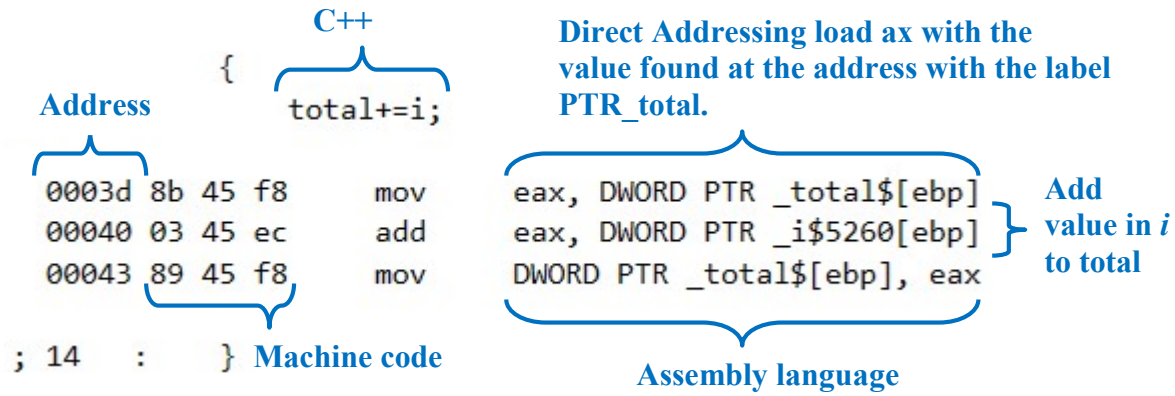


X:\CS253 \To_ASM\Debug\To_ASM.cod



This file contains the compiler output (Machine Code) alongside the original C++ source code. In addition an Assembly Language version of the source code is visible.

Look through this code and see if you can relate the high level language with the corresponding code created by the compiler. For example consider the code snippet shown below. Note EAX is a 32 bit version of the AX register.



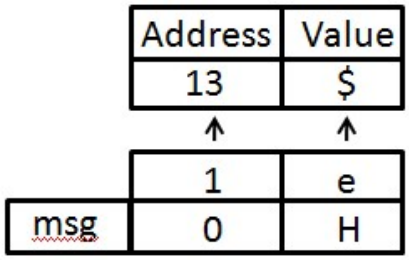
Comment: Well done, this is an unusual introduction to C++ and Visual Studio; we will use it in a more conventional way soon. See the quiz to record your understanding, **Q2**.

Part 3: Investigation of Pointers

Preamble: In assembly language we have used Direct Addressing (which works like a single variable in a higher level language, see part 1) and Register Indirect Addressing (which work like arrays) to access a value stored at a location in memory.

Consider the following comparison of x86 Assembly language Register Indirect Addressing with the equivalent higher level language equivalents, C/C++ and Java. This is very similar to how arrays are implemented; essentially the higher level language simplifies the addressing of data by using an index rather than an absolute memory address.

```
 msg   BYTE     "Hello world$",0
 Name  Type          Value
       .CODE
       .STARTUP
       mov bx,OFFSET msg
back:  mov dx,[bx]          ;dl=letters
```



in C++ the nearest equivalent statement would be one of the following,

```
char msg[]={"Hello world$"};

char msg[]={72,101,108,108,111,32,119,111,114,108,100,36,0};

char msg[13];
     msg[0]=72;   msg[1]=101;  msg[2]=108; msg[3]=108; msg[4]=111;
     msg[5]=32;   msg[6]=119;  msg[7]=111; msg[8]=114; msg[9]=108;
     msg[10]=100; msg[11]=36;  msg[12]=0;
```

or perhaps in Java

```
String msg;
msg = "Hello World$";
char a0 = msg.charAt(0);
```

In C/C++ a variable is declared in a very similar way to Java, for example.

int x=1, creates a variable with name x, value of 1 and type integer.

However C++ allows you more access to the variable, for example we can declare a pointer which stores the memory address of where the value of a variable is stored.

int *a creates a place in memory with the name "a" that stores the location in memory of an integer variable, (it doesn't store the value it stores where it is),

The **&** operator provides a means of obtaining the address of where a variable is stored in memory,

The ***** operator returns the value found at the address provided.

To do: Consider the following program and talk through the various operators with a friend so as to make a prediction of the final run time value of the variable y.


```
#include "stdafx.h"
#include <stdio.h>

int _tmain(int argc, _TCHAR* argv[])
{
int *address; // Create a pointer that can store an address (location in memory)
int x=10;            // Create an integer variable called x containing value 10
int y=0;             // y is an int with an initial value 0
int z[3]={5,7,11};   // Create an array of three integers

address=&x;           // Let address store the value of the address of x in memory
y=*address;           // y is assigned the value of whatever is pointed to by address
address =&z[0];       // address is set to the location in memory of the first value
y+=*(address+2); //y is incremented by the value pointed to at two items above address

printf("%d\n",y);    // prints y to screen (y=????)

// Wait for enter to be pressed before terminating
while(getchar()!=10); // Clear buffer of previous <ret>
while(getchar()!=10); // Wait for a new <ret>
return 0;
}
```

Run the code in Visual Studio and see if your prediction is correct, again record your findings in the quiz, **Q3**.

Finally you often see two more operators associated with accessing classes in C++. I think you will have done classes in Java and if so the following code example should explain itself.

The **.** (pronounced "dot") operator allows you to reference individual members of a Class the **->** (arrow operator) allows you to reference individual members of a class by reference of its pointer.

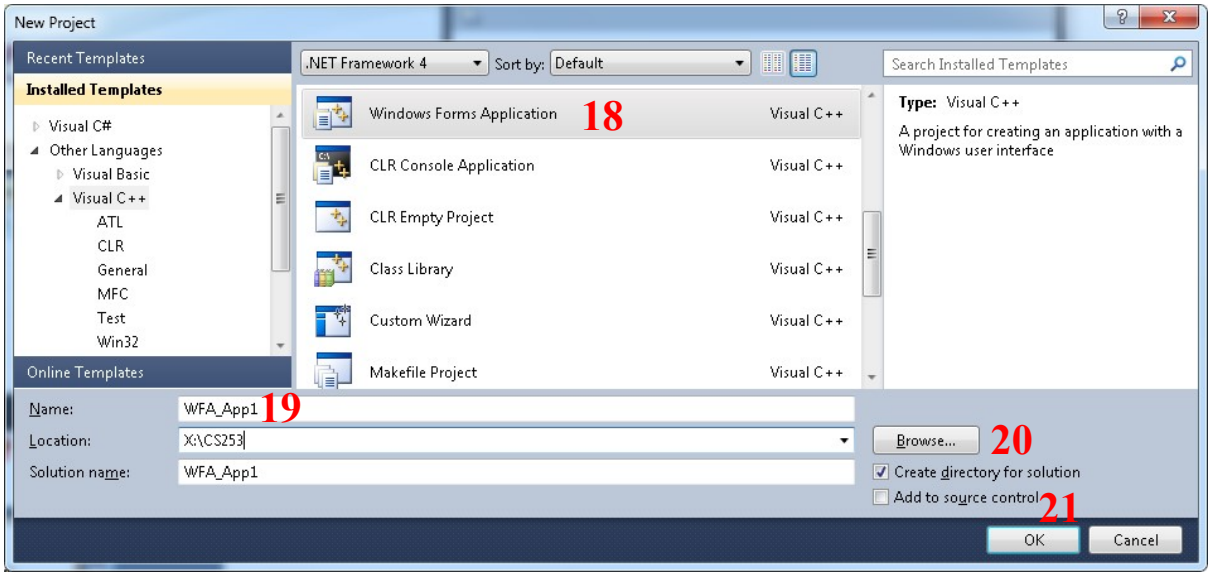
Investigation of the Dot and Arrow operators used to access member functions and variables contained in an instance of a class.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

// Create a class (object) called Sphere, this is a
// description of the object, not an instance of the object.
// It is the cookie cutter not the cookie.
class Sphere
{
public: float radius;
public: float surface_area();

    // Class constructor runs when a new instance of "sphere" is created.
        Sphere()
        {
                radius=1.0f;
        }

        // In line function belonging sphere
        float diameter()
        {
           return(2.0f*radius);
        }
};

// External method used by sphere
float Sphere::surface_area()
{
        float area=4.0f*3.14159*(radius*radius);
        return(area);
}


int _tmain(int argc, _TCHAR* argv[])
{
        class Sphere ball1;   // Create an instance of Sphere called ball1 (a cookie!)
        class Sphere *ball2;  // Create a pointer to store the location of an instance of Sphere

        ball1.radius=10;      // Make the radius of ball1 = 10

        ball2=new Sphere();   // Create a new instance of a sphere and assign its location to ball2

        ball2->radius=10;     // Make ball2 radius equal to 10

        (*ball2).radius=10;   // This is the same as the line above, you can dereference a
                              // pointer and then use the dot to access members, (*p). same as ->

        cout <<  ball1.radius << "," << ball2->radius;  // Print value of radius in ball1 and ball2

        int x; cin >> x;      // Wait for keypress
}
```

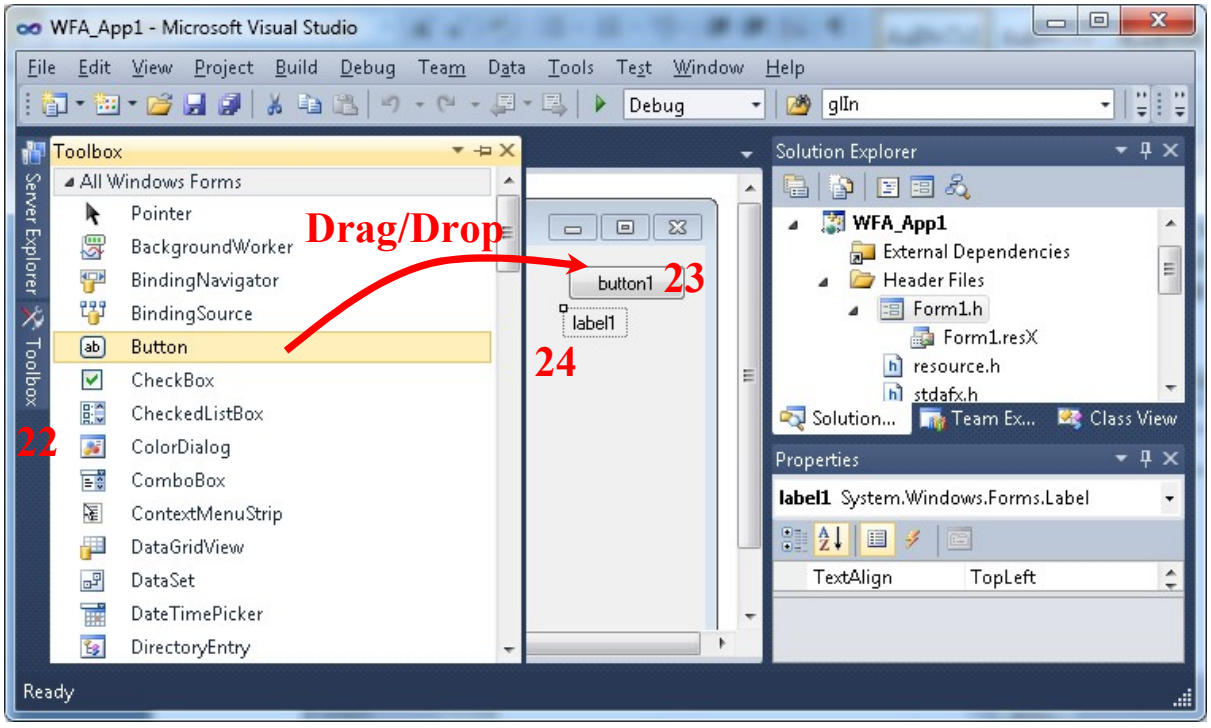
See quiz for a test of understanding, **Q4**.

Part 4: Creating a Windows Forms Application using C/C++

Finally we are ready to create a new C++ Windows Form Application project within the Microsoft Visual Studio 2010 IDE. You can call it what you like and save it where you want (either X:\CS253 or C:\temp), limit the name to no more than 8 characters and never include <space>, ‘,$&~.’ or similar characters. I called mine WF_App1.

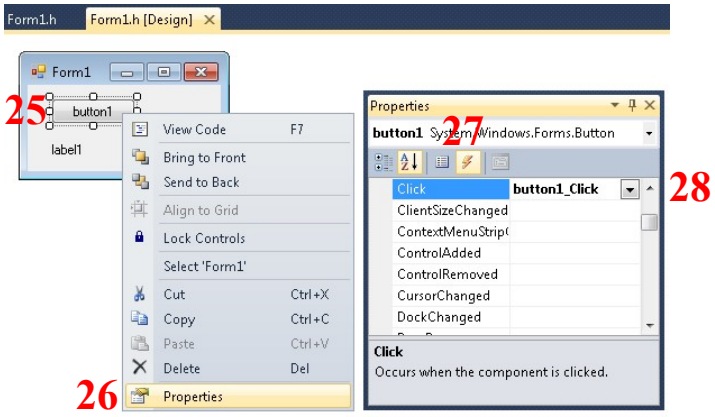


Drag and drop a button and a label onto the Form using the Toolbox. If the Toolbox disappears then recover it using Windows-Reset Windows Layout or Run-Debug-Stop.



Double click on button1 on the form so as to create the “event handler” for a button click.

Note: Alternatively you could left click on the button1, select properties and then click on the lightning bolt (events associated with button) and add the mouse click handler by clicking on the relevant box.

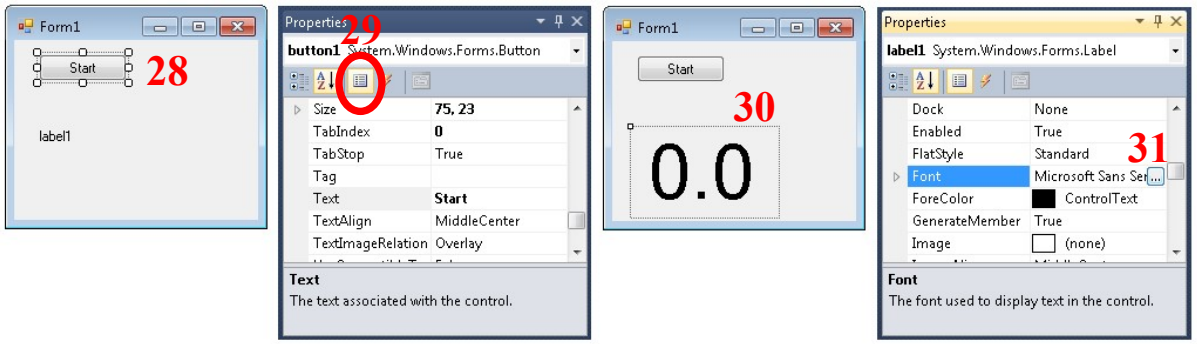


If you have double clicked on button1 (or have clicked on button1_Click in Properties) you will have been brought to the event handler for the event added to Form1.h. Add a line of code so that when a click event occurs, the text associated with the label is changed to “Hello”.

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
                     {
                              label1->Text="Hello";
                     }
```

Run the program and verify that when you click on the button the text changes to “Hello”. Note the ^ is a .NET form of the * operator (i.e. “points to”).

Return to the Form1.h[Design] window and right click on button1 and use properties to change the text to read, “Start”. In a similar way change the font used for the label to be 48 point, Arial and read 0.0. You may need to click on the properties tab to the left of the event tab, see 29.



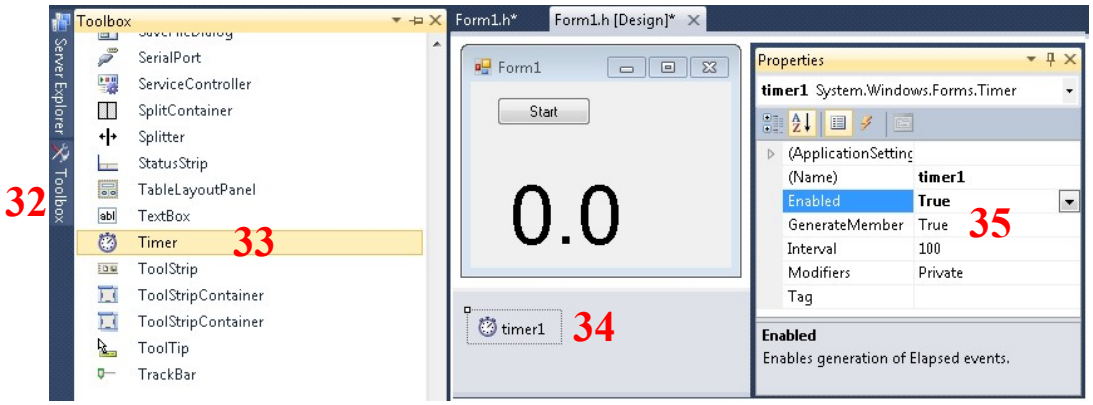
If you look through the code for Form1.h you will find that each of your changes has been added to the code. For example, the following code was added when you created the event handler for a button click and changed the button text. It is easier to use the form editor to create this code rather than type it in directly yourself (you could if you wanted to).

```
this->button1->Text = L"Start";
this->button1->UseVisualStyleBackColor = true;
this->button1->Click += gcnew System::EventHandler(this, &Form1::button1_Click);
```

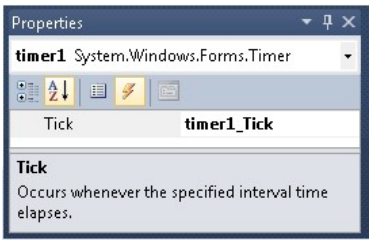
Note the L in front of the string converts it to wide string 16 bits per character.

You could run the program again to verify your changes have take effect.

Return to the Form1.h[Design] and drag and drop a Timer onto the form. As this is not a visible control it will appear below the form. If the Toolbox is missing then use Windows-Reset Layout to make it return (or you may have just left the programming running which prevents the Form being edited also).



Create a timer1_Tick event handler by double clicking on the timer (or you could use the lightning bolt in properties to do this). The 100 refers to (apx) 100 milliseconds between calls to event handler. Then identify the "Tick" event handler in the code.



Add the following two lines to the Form1.h code in the event handler for the timer

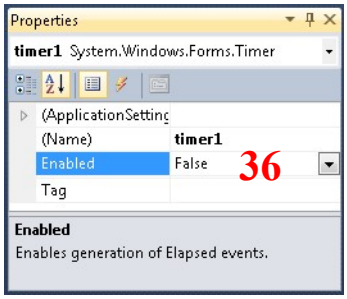
```
private: System::Void timer1_Tick(System::Object^  sender, System::EventArgs^  e)
        {
                count++;
                label1->Text="Time:"+Convert::ToString((float)count/10.0f);
        }
```

Remember, you will need to declare the variable count at the start of the program.

```
using namespace System::Data;
using namespace System::Drawing;
int count=0;
/// <summary>
/// Summary for Form1
```

Run the program and you should see the timer display increment by 0.1 each time the timer event handler is called.

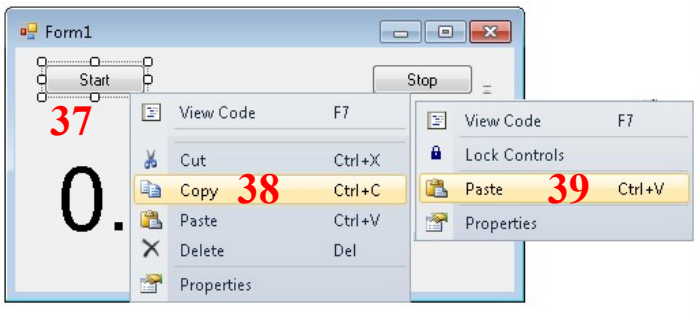
Finally right click on the properties of the timer and disable the timer so that it does not run when the program is started.

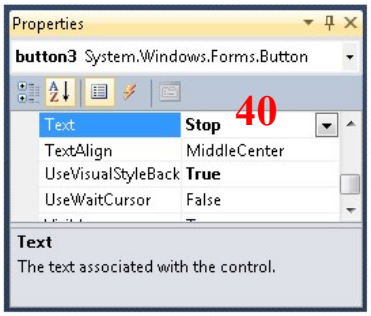


Add the following code to the button1_click event manager so that “Start” enables the timer.

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
        {
                //label1->Text="Hello";
                timer1->Start();
        }
```

Right click on the copy button and paste so as to create a second button, change the text (not name) on this button to “Stop”. Add an event manager for the new button and then add the line of code to stop the timer.






```
private: System::Void button3_Click(System::Object^  sender, System::EventArgs^  e)
                {
                        timer1->Stop();
                }
```

Finally you will notice that this is a fairly poor timer as the event happens about every 100 milliseconds.

You could improve your timer by using the system clock, the following code snippets should help with this,

At the start of the Form1.h, add the following two lines....

```
#pragma once
#include<time.h>
namespace WFA_App1 {
using namespace System;
...
using namespace System::Drawing;

int count=0;
clock_t start_time, current_time;
/// <summary>
/// Summary for Form1
/// </summary>
public ref class Form1 : public System::Windows::Forms::Form
{
```

In the event handlers, add the following three new lines and comment out three lines....

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
{
        //label1->Text="Hello";
        timer1->Start();
        start_time=clock();  // Record a start time
}

private: System::Void timer1_Tick(System::Object^  sender, System::EventArgs^  e)
{
        // count++;
        // label1->Text="Time:"+Convert::ToString((float)count/10.0f);
        current_time=clock();
        label1->Text="Time:"+String::Format("{0:0.00}",(float)(current_time-start_time)/1000.0f);
}
```

To do: Use this code base to create your own stop watch application. Submit the Form1.h source code and a screen capture image of the application running; show your running code to a demonstrator before you leave.

Note: We could continue this laboratory by adding a panel and drawing lines on it using the mouse events. You could also investigate the use of other controls such as the slider bar.

Windows forms can be created in either c++ (as we have done today) or in C# which is very similar to Java with which you are already familiar.

After the break we will do one more laboratory session where we do similar work but use C# instead of C++. C# is my favourite way of building Windows Forms Applications, the auto-complete (Microsoft call it Intellisense) and more intuitive language (C# is very similar to Java) make it very fast for development.

A Final Thought: Today we have looked at event driven programming, interrupts are the computers way of detecting events, the use interrupts is the theme of next week's laboratory session.