Before the lecture begins …

- Install Microsoft Visual C++ Compiler 6.0+ and TeamSpeak (version 7.0+, current version 9.0b) – see the Technical Guide on our site if you need help or don't have Visual C++.
- Download ITCC_HW0.zip, then ***Build*** and execute the zipped projects FirstProgram.cpp and HelloGraphics.cpp (see these slides for instructions).
- Do all of the TODO sections in each project's main CPP file (these do not have to be submitted)
- Review the Lecture 1 slides.

Intro to C++

Lecture 1

Compiler, Includes, Analysis of Basic Programs, if Statements

Notation

- Source code keywords are highlighted or boxed in Ms Sans Serif font.
  - `void main( int argv )`
- Conceptual Keywords are in italics.
  - *Implementation.*
- Important items are in bold.
  - **something important here**

Programming

- Why is it hard to learn?
  - Little influence from outside. Anything takes work.

- How can you learn?
  - Practice, Practice, Practice
  - Make up challenges for yourself and implement them
  - Check out a few C++ books to reinforce concepts
  - Consult with friends who know programming
  - Positive surroundings
  - Know your goals, final goals are always exciting
  - Look at what others have made

C vs. C++

- C++ can be thought of as a superset of C.
- C++ is different because programmers can use ***Object-Oriented Programming*** (***OOP***) rather than traditional programming methods to design software. (Discussed in later lectures.)
- It has been shown that C++ drastically enhances productivity and program maintainability over C (*Code Complete*).
- If you learn C, then most of what you know can be directly used in C++. This course will attempt to use OOP as much as possible from the beginning to ease the learning curve later on.

SDKs

- The compiler comes with *source code* that specifically allows you to program for Windows which is NOT standard code to the C language. Generally these are referred to as *SDKs* (Software Development Kits) or *APIs* (Application Programming Interfaces).
- Hence, the DirectX SDK or Windows SDK contain code necessary to develop applications that use DirectX or Windows.

The Anatomy of the EXE

- An *EXE* (executable) *file* is written in *machine language*, largely unintelligible to humans but very compact. Every few bytes of machine code tells the processor to do a single, very specific action.
- Every line of C or C++ code may correspond to several lines of assembly code (on average, a 2:1 ratio according to *Code Complete*)

C Code to Machine Language

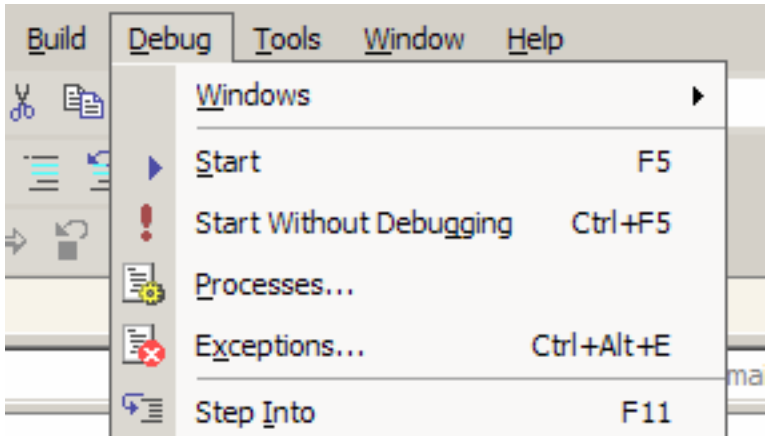
- First, each *source file* (a file containing C/C++ code) is ***compiled***, converting it into an *object file* (a half-done state which defines all of the objects present).
- Second, each *object file* is ***linked*** *together* to form machine code. Why do they need to be linked? Some *source files* may require unknown code in other files which cannot be immediately retrieved when a single file is compiled.
- The process of ***compiling*** then ***linking*** is called ***building*** a project.

Visual Studio: Building software

- Let's go over how to use your compiler.
- These are meant to be very quick, because we want to get to the fun stuff.
- If you have a question at any point **ask.**

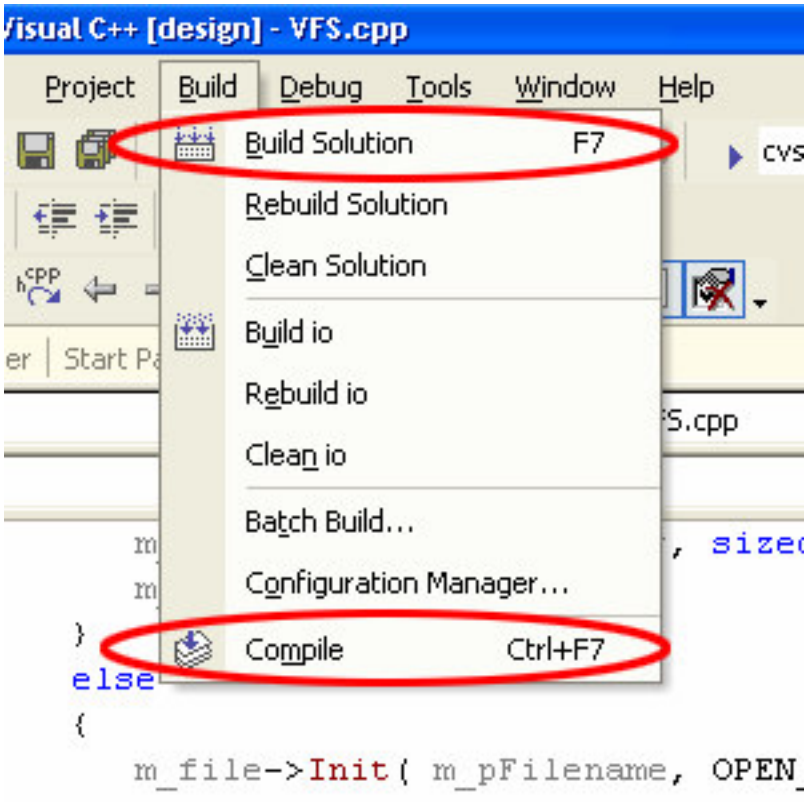
How to make an EXE

- Load a .dsw (workspace, VC6) or .vcproj (project, VC7) file by double-clicking on it or by selecting File->Open Workspace …
- Press F5 to ***build*** the project then run it in Debug mode (usually the best option), or press Ctrl+F5 to ***build*** the project and run it without debugging.



Compiler

- Build (F7)
  - ***Compiles*** and ***Links*** all of the files in a project.
- Compile (Ctrl+F7)
  - Takes a file and converts it to executable form.
- Clean
  - Remove intermediate files used in compilation.

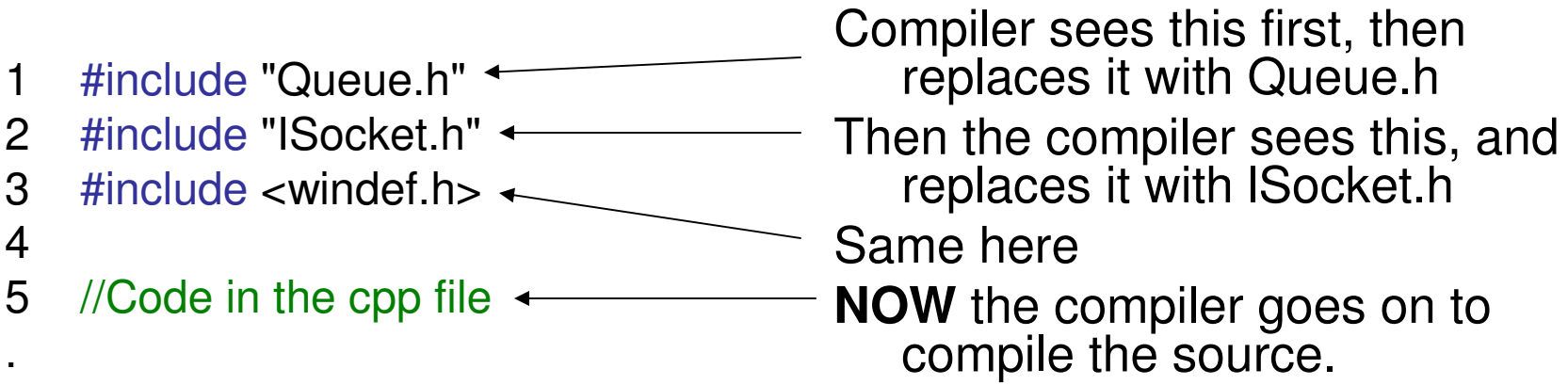


Errors

- Many types of errors exist:
  - *Compile-Time errors*, which occur when you ***Build/Compile*** a file and the compiler tells you about it, often refusing to make an EXE.
  - *Linkage-Time errors*, which typically occur because a symbol used was not defined in the given files.
  - *Run-Time errors*, which occur when a program has successfully compiled, but generates an error when it runs.
  - *Logical errors*, a subset of *Run-Time errors*, these are errors the programmer makes – they don't have any technical flaw so the compiler can never catch them, but your program just doesn't run as you expected.

#include

- ***Preprocessor command/directive***, a line that starts with #. These commands are meant to be compiler commands – they do NOT directly translate into machine code.
- #include “<filename in quotes>” – Tells the compiler to replace the #include line with the contents of the filename.
- Example: In the following code fragment “Queue.h”, “ISocket.h”, and “windef.h” will be inserted into the file in that order.
- Compilation in a file is **top-down**, starting at the top.



Header Files

```
#include "ISocket.h"
#include <windef.h>
```

- What is the difference between these two statements? Notice that neither of the .h files has a path associated with it.
- “” indicates that the .h file can be found in the **same directory as the file that contains the #include**
- <> indicates that the .h file can be found in a **standard, pre-specified directory for .h files**
- Why .h? .h stands for ***header file***, and these files typically declare symbols, not logic, that will be used in your .cpp file.

Analyzing FirstProgram.cpp

What are all of the //s?

- All text following a // is a ***comment***, meaning that the text will be ignored by the compiler
- This allows programmers to write meaningful thoughts into their code
- Good programmers write comments fairly often by commenting on groups of commands that have a purpose not immediately obvious from merely looking at the source code

Analyzing FirstProgram.cpp

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

int main(void)
{
    cout << "Hello, world!" << endl;
    getch();

    return 0;
}
```

- Initial observations:
  - Every line (excluding ***preprocessor directives***, braces, and ***function definitions***, explained later), has a semicolon that terminates it.
  - When the EXE is run, the processor starts code execution inside the braces of the ***main function***.

Analyzing FirstProgram.cpp

```
#include <iostream>
#include <conio.h>
```

- Files that are part of the *C++ Standard Library* should not include a .h (although the physical file still has one). This is to distinguish these files from deprecated C++ standards.

```
using namespace std;
```

- To avoid naming conflicts, the *C++ Standard Library* places its commands inside the ***namespace*** std. By stating that we are using this ***namespace***, this saves us typing by not having to prefix standard library commands with 'std::'. In general, it is a bad idea to use the using namespace command, but here its use may enhance learning. (Will be covered in later lectures.)

Analyzing FirstProgram.cpp

```
int main(void)
```

- This is a ***function definition*** (to be analyzed later on in the course). For now, know that this function represents the entry point for all C++ programs – the place where code is first executed.

```
cout << "Hello, world!" << endl;
getch();
```

- 'cout' stands for 'character out'. The '<<' writes the "Hello, world!" text to the standard output stream. Since it points toward cout, this typically writes the text to the terminal display. 'endl' (end line) sets the cursor to the beginning of the next line.
- 'getch' stands for 'get character'. This provides a pause before exiting the program.

```
return 0;
```

- This sends a code of zero to the operating system. A zero indicates that our program terminated normally.

Console vs. Graphics

- Note that FirstProgram.cpp is a ***console application***. Most beginning programming courses emphasize how to program console apps because graphics are usually more complicated.
- However, we are going to abandon the console window and delve into graphics programming.
- Graphics are complicated to set up, so we have provided a *common* ***framework*** that you must link with your code. You must also link some DirectX *library files* (.lib) with your project.
- If you open one of our .dsw or .vcproj files, then all of the setup necessary to correctly build applications is done automatically, provided that you have installed the DirectX SDK.

Graphics

- Welcome aboard – now is when it starts to get exciting!
- The ***framework*** which we've provided you with emulates programming the video card in DOS before *SDKs* such as DirectX and OpenGL were available.
- Because of this, every ***pixel*** (the smallest dot that can be lit on your monitor) that is displayed on the screen has to be manually lit by our code.
- Throughout these lectures, we'll not only increase our C++ knowledge, but we'll grow to understand how SDKs like DirectX actually work by drawing graphics in software.
- Don't worry – even as a beginner to C++, you'll start to understand how graphics programming works, and at a very fundamental level!

Analyzing HelloGraphics.cpp

```
#include "../Framework/ITCC_Framework.h"
#pragma comment (lib, "../Framework/ITCC_Framework.lib")

// the (fake) entry point for our application
int Main(ITCC *itcc)
{
        itcc->InitWindowed(640, 480);

        while (itcc->Flip() != NULL)
        {
                itcc->Clear();
                itcc->Text ("Hello World!", 0, 0, RGB(255,0,0));
        }

        return 0;               // terminated successfully
}
```

Analyzing HelloGraphics.cpp

```
#include "./Common/ITCC_WinMain.h“
#pragma comment (lib, "../Framework/ITCC_Framework.lib")
```

- ITCC_WinMain.h defines symbols specific to our ***framework*** so that we can call ITCC_* functions without errors. ITCC_Framework.lib provides pre-compiled code written by us that make it easier to develop Windows graphical applications. The .lib was provided instead of the source so that you don't need to download DirectX SDK to compile our source code.

```
int Main(ITCC *itcc)
```

- Not the entry point of the app, but it is for our purposes (graphics initialization is done in the real main function, WinMain(…) for Windows apps).

```
ITCC_Windowed(640, 480);
```

- Makes a window appear with its ***client area*** (the area under the title bar) having a width of 640 pixels and a height of 480 pixels.

Analyzing HelloGraphics.cpp

```
itcc->Clear();
itcc->Text ("Hello World!", 0, 0, RGB(255,0,0));
```

- Clears the ***client area*** to black, then prints the upper-left point of the text “Hello World!” at (x,y)=(0,0) on the monitor. The three numbers inside RGB correspond to the intensities of the red, blue, and green light mixed together (note that 255 is the highest allowed intensity).

```
while (itcc->Flip() != NULL)
```

- The Flip() function is required to display any graphics that have been written to the display. With the while command, we continually clear the display and display text for the entire life of the program.

```
return 0;
```

- This sends a code of zero to the operating system. A zero indicates that our program terminated normally.

The if Statement

- Used to test whether a certain statement is true. If it is true, then the code inside the curly braces following the if is executed; if not, this code is skipped.

```
if (KEYDOWN(VK_UP))
{
    itcc->Text (“Up key down!", 0, 0, RGB(255,0,0));
}
```

Here, text at (0,0) is only drawn if the up arrow key is depressed.

Tip: Placing ‘!’ in front of ‘KEYDOWN’ will test whether the key is NOT depressed.

if/else

- What if we wanted some code to be executed if a condition is true (e.g. when a key is pressed), else execute some other code (e.g. when the key isn't pressed)?

```
if (KEYDOWN(VK_UP))
{
    itcc->Text (“Up: down!", 0, 0, RGB(255,0,0));

}
else if (!KEYDOWN(VK_UP))
{
    itcc->Text (“Up: up!", 0, 0, RGB(0,255,0));
}
```

Method 1

```
if (KEYDOWN(VK_UP))
{
    itcc->Text ("Up: down!", 0, 0, RGB(255,0,0));
}
else
{
    itcc->Text (“Up: up!", 0, 0, RGB(0,255,0));
}
```

Method 2

SetPixel

Call the following inside the rendering loop:

```
itcc->SetPixel(0, 0, RGB32(255,255,255));
```

- This will light one white pixel on the screen in the upper-leftmost corner.
- The first two ***parameters*** are where to place the pixel in the ***client area*** – its x- and y-coordinate, respectively. x is left-right, y is up-down.

SetPixel: RGB

- We seem to use the term ‘RGB’ a lot in our code, always requiring three numbers.
- This stands for ‘Red Green Blue’. Every visible color can be created from a combination of intensities of these three lights.
- When we input the three numbers, 0 is minimum intensity and 255 is maximum intensity.
- Here are how to make common colors:

```
Red   :   (255, 0, 0)   Yellow  : (255, 255, 0)
Green :   (0, 255, 0)   Magenta : (255, 0, 255)
Blue  :   (0, 0, 255)   Cyan    : (0, 255, 255)
Black :   (0, 0,   0)   White   : (255,255,255)
```

SetPixel: Color Depth

Note the ***macro*** inside the SetPixel ***subroutine***:

RGB32(255,255,255));

- This will only create the right color for 32-bit color modes, determined by what your desktop resolution is set at.
- If this does not work, try the other two ***macros*** RGB16a(…) or RGB16b(…) for 16-bit desktops.

Full Screen

- Instead of itcc->InitWindowed, try typing:

```
itcc->InitFullScreen(640, 480, 32);
```

- This will initialize a full-screen application, physically changing your monitor's display resolution like computer games do.
- The first two ***parameters*** are the width and height of the screen.
- The third ***parameter*** is the ***bit depth*** of the desired resolution. Remember to use the right RGB* ***macro*** if you're drawing pixels on the screen!
- If you pass numbers that your video card cannot handle, then your application will become unstable. We will talk about resolving this in the next lecture.

TODO

- Download ITCC_1HW.zip from the site, http://www.pclx.com/itcc/, and complete the homework exercises, emailing them (FOR THIS WEEK ONLY) to itcc_teachers@pclx.com. Please do not resubmit solutions, even if they are revised. All homework must be submitted by 6:00am PST Monday, July 5.
- Look over the slides for the second lecture before Monday, July 5.
- If you finish with the homework, experiment!