Programming in C/C++

Lecture 1 – Chapter 1: Introduction to C++

Kristina Shroyer

Objectives

- Introduction
  - Software Developer Skill Set
  - Rules for Programmers
- C++ Concepts and Background
  - ANSI Standard C++
    - We will be using the SECOND STANDARD AND LATER – we will NOT use anything from the pre-1990 FIRST standard
      - I am teaching the second standard and some things from the third standard and beyond
      - I am NOT teaching the FIRST standard – the REALLY OLD ONE FROM BEFORE 1990 – the OLD FIRST STANDARD is NOT OK to use in this class – only the second or later (and only use what is learned in class)!! (very important for tests)
    - Most of the basic/intermediate concepts we are learning in the second standard have not changed much in the standards beyond the second one(the third standard (and the ones beyond that) have changed some of the more advanced concepts and added some bells and whistles).
      - For example they have a new type of tweaked pointer but the old one will still be used and you must understand the old one before the new one will make sense
      - The point is everything we do will have value in the new standard as well
      - We will do at least one pretty different thing from the new standard near the end of the class which is the new Array data type (different than built in arrays)
        - I will make sure and point this out when we get to it
    - I focus more on the new standard in CSIS 137 (the second level class which is intermediate and advanced concepts where this fits better)
- Understand the Compiler and the Compilation Process
- Why Program in C/C++?
- Understand the Basic Rules for C++ Statements
- Review algorithms and pseudocode
- Learn to do Basic Output
- ***EITHER SET UP MICROSOFT VISUAL STUDIO COMMUNITY 2013 AT HOME OR X-CODE (if you have mac) – THIS IS REQUIRED…not having software is not an excuse for not doing homework***
  - ***For Mac I want you to use Xcode at home and Visual Studio Professional at school (the version on the school computers which is 2013) – YOU NEED TO KNOW BOTH X Code and Visual Studio so you can do the exams***
  - ***For PC I want you to get Visual Studio Community 2013 at home and use the school's Visual Studio Professional 2013 at school (they work almost exactly the same)***
- Review what an Identifier is and the learn the C++ rules for using one

Introduction

- C++ is widely used in today's programming environment
  - Game Programming
  - Controllers for Robots
  - User Interfaces
  - Graphics for Games
  - Anything where the programmer wants more lower level control over thing like memory is suited for C++, it really depends on the project and the company.

- Skills needed to be a software developer/engineer:
  - Programming
  - Problem Solving – Can you actually solve a problem without "copying" another one?
  - Design Skills (Software Design)
    - You must know/research the field you are designing the program for
    - Object Oriented Design
  - Documentation (code should be readable and self documenting as well)
    - Your software may be used for years to come
    - Someone should be able to read your code and easily understand it
  - Communication/No Ego
    - Team members, customers, end users
  - Quick Learners

Introduction

- **10 Rules For Programmers**

1. Keep Your Cool

2. Work When You are Rested

3. **KEEP IT SIMPLE**

4. Give help/Get help
   - On Homework – **NOT** on tests
   - What is help?
   - What is **NOT** help
     - Copying code
     - Copying code and only changing variable names or the order of certain statements
     - People who copy homework almost always do very poorly on the programming tests
       - They think I don't know they did it, but I do – I just don't say anything about it and I remember it when they ask me for letters of recommendation (I don't give those to people who give their code to others or people who do the copying)
   - ***Ask before class, come to Office Hours or Email***

5. Study and Know the Rules for the Language
   - Syntax

Introduction

- **10 Rules For Programmers (continued)**

6. Learn the Development Environment and Tools
   - ***Microsoft Visual Studio Community 2013 OR LATEST VERSION OF XCODE (Do NOT use Visual Studio "Code" – it's not the same thing – and I recommend Visual Studio 2013 instead of 2015)***
     - We'll use the Microsoft Visual Studio Professional 2013 in class and you can use either of the following at home
       - Microsoft Visual Studio Community ***2013*** (which you can download for free – try the 2013 version first if that doesn't work you can try 2015 – I found 2013 to be less buggy)
       - Xcode if you have Mac
     - ***YOU MUST USE Microsoft Visual Studio Professional 2013 (the one on the school computers) FOR TESTS IN THIS CLASS – at home you can choose between Microsoft Visual Studio Community 2013 or XCode***
       - Part of the goal of this class is for you to go out of it knowing a real world, widely used IDE for C++
       - If you have a Macintosh: You should use XCode
       - ***For Homework technically you can use whatever IDE you want (however I will only support the ones I've recommended above) BUT FOR EXAMS YOU WILL BE REQUIRED TO USE VISUAL STUDIO PROFESSIONAL 2013 (part of the class is knowing a professional IDE and part of the test will be knowing how to use it)***

7. Understand the Problem You are Trying to Solve

8. **BUILD AND TEST YOUR SOFTWARE IN STEPS**
   - Compile Often

9. Save Early/Save Often

10. Practice and Study
    - Know both how to program and the programming concepts we discuss in lecture
      - Objective Exams
      - Programming Exams

C/C++ - Brief History and Overview

- **Brief Overview of C/C++**
  - Originally developed by Bell Laboratories
    - In the 1970s **C** was originally used for operating systems (Bell's UNIX)
    - **C** quickly grew in popularity as a general purpose programming language
      - Offered the tools for writing many types of programs
      - In 1978, in addition to the UNIX versions of C, Honeywell, IBM, and Interdata also offered application software for the C language
  - C kept growing in popularity and hardware became more affordable
    - There were many version of C created
    - **Rumor:** ***At one time there were 24 Versions of C!***
    - **PROBLEM:** ***There was no standard for the language***

C/C++ - Brief History and Overview

- **Brief Overview of C/C++ (continued)**
  - ***American National Standards Institute Committee (ANSI)***
    - Formed in 1982 to create a standard for the C language
    - The ANSI standard was adopted by the ***International Standards Organization (ISO)*** in 1989
      - Think of the ISO as the governing laws for C/C++
        - These laws dictate the correct form of C/C++ statements and how aspects of the language must work
  - ***ANSI Standard C++***

C/C++ - Brief History and Overview

- **Brief Overview of C/C++ (continued)**
  - In the mid 1990s extensions and corrections to the ISO C Standard were adopted
    - BIG PART OF EXPANSION: The ability to build object oriented software
      - The birth of C++
    - Other parts of the extension
      - Additional library support for foreign character sets, multibyte characters and wide characters
  - A working draft of the ISO C++ Standard was created in 1994
    - In 1997 the ANSI committee published the final draft of the C++ language
      - It was approved by the ISO making it an international standard
  - Programs written according to ISO C++ standards can be run (or modified and run) on any computer system that has ISO Standard C++ software (meaning a ANSI/ISO Standard Compiler)
  - ***NOTE:*** The new standard ***C++11 (the one before this was C++3/both will just be called standard C++)*** came out in August 2011 that books and compilers are just now starting to adapt to, most of the things we learn in this class have not changed with the new standard and anything that has should be compatible but you should know the new standard exists (our book definitely has not caught up with it)
    - Everything is forward compatible so we don't have to worry about anything we learn not working in the new standard
    - The problem with standards is it takes everyone a while to catch up – some people are using the PRE-1990 (REALLY OLD) standard still – we will NOT do that in this class…now that is TWO behind instead of one

- ***So there are MORE THAN THREE C++ Standards***
  1. **The OLD one from BEFORE 1994!**
     - DO NOT USE! TOO OUTDATED
  2. **The SECOND one that was created in 1994**
     - We will be using this combined with some of the newest stuff – the basic and intermediate level concepts we learn were not affected by the third standard – some were and I'll point that out as we go though
  3. **The THIRD one from 2011 AND THERE ARE AT LEAST TWO MORE SINCE THEN!!!**
     - The newest standards haven't changed the basics and intermediate things we will be using in class so most of what we learn will be from the second standard
     - We will NOT USE ANYTHING FROM THE FIRST STANDARD (points off for using such old stuff)
     - The third standard and beyond is small tweaks and advanced concepts – I will incorporate maybe a few things but most of this will be in CSIS 137 – Advanced C++

C/C++ - Brief History and Overview

- **Since the C++ language was an expansion of the C language:**
  - **When you learn C++, you are learning C as well**
  - The entire C language is wrapped up in C++
    - `if` and `for` loops are essentially identical in C and C++
    - C has structures (structs), C++ has structures and classes (used with objects)

C++ Concepts/Background

- **C++ source program:** Set of instructions written in C++ language
- **Program Translation:**
  - How a program goes from your C++ source code to machine language that can be read by the computer
- **Machine language:** Internal computer language
  - Consists of a series of 1s and 0s – Are they really 1s and 0s? What does the computer really read?
  - Source code in any language cannot be executed until it is translated into machine language
- **Computer languages are either interpreted or compiled**
  - Interpreted language translates one statement at a time and executes it
  - Compiled language translates all statements together – the program can then later be executed all at once
  - ***For people who have taken CSIS 112 what type of language was Java? (trick question)***
- **C++ is a Compiled Language**
  - A compiler is a computer program that reads source code, and if that source code is grammatically correct translates it into machine code
  - Note, that we say compilation of a program results in a machine language program in C++, we do NOT yet have an executable file

C++ Concepts/Background

- ***Two Levels of Computer Languages*** **– Low Level Languages and High Level Languages**

  - **Low Level Languages**
  - **Machine Language:** The language made up of binary coded instructions that is used directly by the computer
    - In the beginning all programming was done in machine language
    - Unique to each computer (different binary codes for each instruction)
    - Hard to read and modify
  - **Assembly Language:** developed to make the programmer's life easier
    - Uses mnemonics to represent each of the machine language instructions in a particular computer
      - ADD 100101
    - Is easier to work with but a computer cannot directly translate the instructions – An assembler is required
    - ***Assembler:*** A program that translates an assembly language program into machine code
    - Assembly language is a step in the right direction but still requires that programmers think in terms of individual machine instructions
    - Assembly language is a ***low level language***
  - **Low Level Language characteristics**
    - Usually Involve Instructions unique to the processor around which the computer is constructed
    - Permit the programmer to use special instructions that are tied to the particular type of processor the language is for
    - Closer to machine language than to the natural language of humans
    - Faster to execute than high level languages

C++ Concepts/Background

- **High Level Languages**
  - Easier to learn than assembly language or machine code because they are closer to English and other natural languages
    - One word of code represents many machine language instructions
  - Examples: C++, C, Java, FORTRAN, C#, Pascal etc.
  - The ***translation process is more involved*** in a high level language so programs may take longer to execute than those written in a low level language

- **Intermediate Level Language**
  - Sometimes C++/C are referred to as intermediate level languages because they are closer to machine language than a language like Java is
    - However they are still much higher level languages than assembly language
  - ***C++ gives the programmer the ability to work with and manipulate memory which Java does not***
    - Pointers (used often in graphics and game programming)
    - In game programming this is really important (speed reasons) and C++ memory management is usually re-written even to make code even faster

C++ Concepts/Background

- **Libraries**
  - C++ provides libraries containing classes and functions that a programmer can use in their programs
  - One C++ library we will use in our very first program is a library that provides tools to receive input from the keyboard and write output to the monitor

- **Linker**
  - After source code is compiled successfully, your C++ source code has been translated into machine language
  - However a C++ program may include many C++ source code files as well as C++ library code
  - The next step in building an executable C++ program is to link all of the machine code and library code together and bind it into an executable file
  - You only have an executable C++ program (file) after your program has been through both the compilation and linking process

C++ Concepts/Background

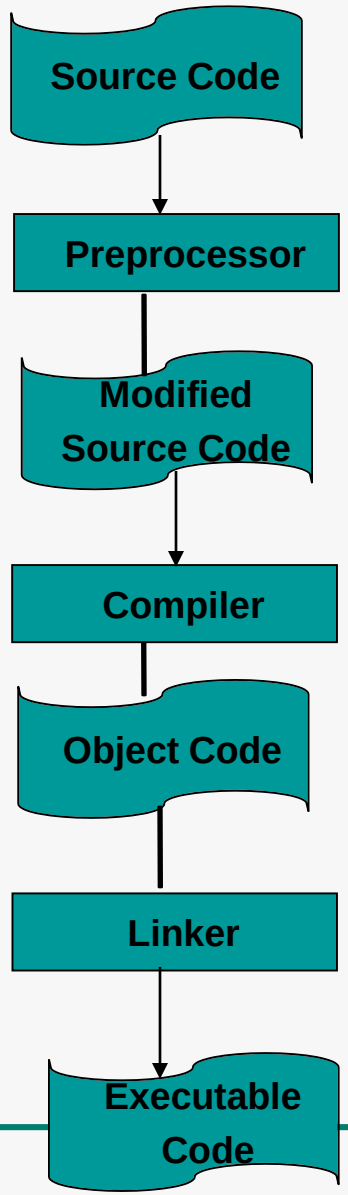
- **The C++ Translation Process**

- **Source Code:** When a C++ program is written the first step is to type it onto a computer and save it into a file (.cpp). After source code is saved into a file the translation process begins

- **Preprocessor:** The preprocessor searches in the source file for special lines of code that begin with the **#** symbol. **Lines of code beginning with this symbol cause the preprocessor to modify the code in some way *before* it is translated into machine language**
  - Example:

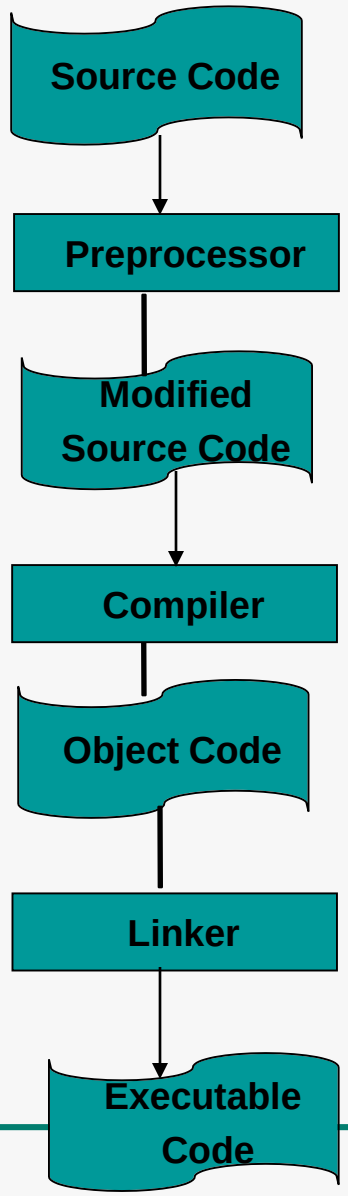
    `#include <named file>` causes the processor to include the contents of the named file to be inserted wherever the `#include` command appears in the program

- **Modified Source Code:** Source code after modification by the preprocessor



C++ Concepts/Background

- **The C++ Translation Process (cont.)**
- **Compiler:** The compiler translates the source code into machine language.
  - The Compiler creates an **object code file (machine code)** that contains machine code for operations in source code file. The object code file only knows the location of functions defined in the source code file
  - You do NOT yet have an executable
- **Linker:** A program may be stored into several source code files, each of which is compiled separately. Additionally, a program may use run-time libraries that contain commonly used functions such as input/output functions and mathematical functions.
  - **It is the job of the linker to combine all of these files into a single program so they can be executed together.**
  - **Only after the Linker has run do you have an executable C++ file**
- Once the linker is run and completed the code is in executable form and can be run



C++ Co

- **The steps required for the programmer to compile, link, and run a C++ program**
- ***This diagram does not include the preprocessor step***
  - This step would go right before the C++ Compiler Step

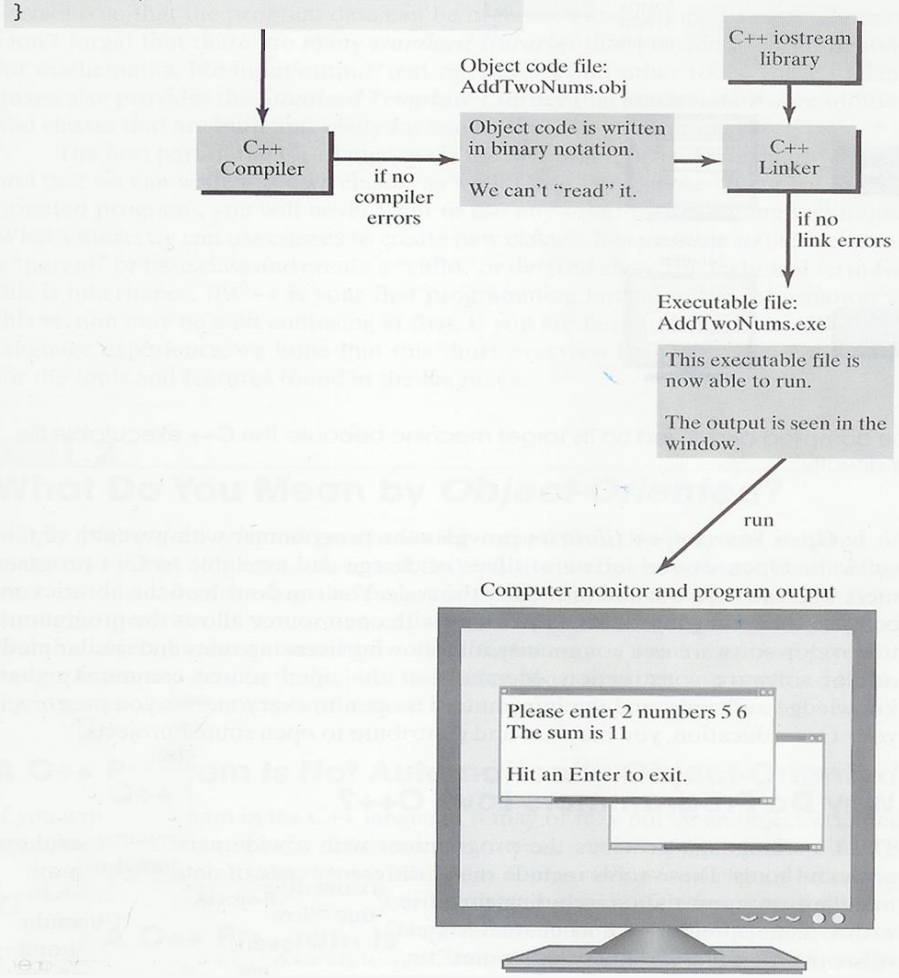
```
// File : AddTwoNums.cpp
// Program that adds two numbers.

#include <iostream>
using namespace std;

int main()
{
  int A, B, C;
  char Enter;

  cout << "\n Please enter 2 numbers";
  cin >> A >> B;
  C = A + B;
  cout << "\n The sum is " << C;
  cout << "\n Hit an Enter key to exit.";
  cin.get(Enter);

  return 0;
}
```



C++ Concepts/Background

The General C++ Programming Process

1. Algorithm Selection and Design (we will use pseudocode)
2. Use a text editor or IDE (integrated development environment) to enter the C++ source code
3. Save the source code as a .cpp file (later on you will have more than one file)
4. Tell the C++ compiler to convert the program into low-level instructions (machine code)
   - When you tell the compiler to run, the preprocessor will be run before the compiler as part of the process
   - **We are going to use the Microsoft Visual C++ IDE for this class**
   - **Since Microsoft Visual C++ is a common IDE used in the industry you must use this IDE for the class**
     - Later in the lecture I'll go through how to set this IDE up and how to use it both in class and at home
5. Tell the C++ compiler to link the source code file with supporting libraries and/or other source code files the program needs and produce an executable file
6. Run the program
   - the operating system loads your executable file into memory and passes control over to it
   - Program runs

C++ Concepts/Background

- ***Portable Language:*** A language in which the **source code** does not need to be changed when the program is moved from one computer to another

- **Example:** You write a program in ISO ANSI Standard C++
  - To run the program on a PC, it must be compiled and linked on a PC.
    - The executable file issues commands directly to the processor when the program runs
  - To run the program on a Sun Microsystems Computer with a SPARC processor, the code must be compiled and linked on this type of machine
  - NOTE: The ISO/C++ source code does not need to be changed
  - This makes C++ a portable language since the source code does not need to be changed when moved from one type of machine to another
    - **You only need to re-compile to run the program on a different type of machine**

- (***NOTE:*** I will be teaching ANSI Standard C++ (some will be the second standard and some will be the newest third), do NOT use oldest Style C++ from the 1970's, 80's and 90's (this is the FIRST standard and is WAY OUTDATED to the point of being obsolete…no one wants to learn to be a dinosaur type programmer), your programs will be marked down significantly if you do this – it is even more wrong now with the new standard, I know some old school teachers teach this and they shouldn't be)

C++ Concepts/Background

- **CAUTION REGARDING THE PORTABILITY OF C++ PROGRAMS:**
  - **If a C++ programmer uses custom libraries in their code that are specific to a certain operating system or machine, this code will NOT be portable across machines**
    - For example
      - Visual C++ provides customized libraries for developing Microsoft Windows specific programs (we don't use those in this class but in some of the more advanced classes we use them)
      - Apple has libraries for building Mac Specific Programs
      - If a program uses any system-dependent classes and functions it will not be portable to other types of machines
  - In addition, you will notice certain subtle difference in IDEs ***depending on the compiler and how well it adheres to ISO C++ Standard***
    - The programs we write in this class should be simple enough they will run easily on most C++ compilers

- However, some cross platform libraries are being/have been developed that allow programmers to write code that can be compiled and linked on various types of machines
  - *wxWidgets* is an open source C++ graphical user interface framework fro building cross platform C++ programs
    - http://www.wxwidgets.org/

Object Oriented Programming

- **Programming languages are also classified by *orientation***
  - **Procedural Languages**
  - **Object Oriented Languages**

- C++ is considered an ***object oriented language*** since it has the tools available for the programmer to write an object oriented program with

- HOWEVER, Programs written in C++ are NOT automatically object oriented
  - Everything in C++ is NOT an object
  - For a C++ program to be object oriented it must be written using *objects and classes*
  - Before the C++ extension was added to the C language programs in C were written as procedural programs rather than object oriented programs
    - ***Procedural programs*** are designed in a linear manner and focus on a specific sequence of events that solve a problem – procedural programs are less reusable than object oriented programs (***with procedural programs the data is separate from the procedures/methods that act on the data***)
    - Common design scenario for a procedural program:
      - Start the program
      - Find and open the data file
      - Use the data as needed and calculate the needed output
      - Output some sort of report
      - Close the data file
    - You would design methods (called functions in C++) that perform various tasks on the data, the methods would accept both input and output data…so in procedural programs we have methods that act on data
  - It is possible to write a procedural program in C++
    - As we learn basic concepts we will start with procedural programs that utilize objects
    - In the second half of the class you will create your own objects

- NOTE: WE are NOT trying to be procedural programmers in this class. I teach procedural programming in a way to gear you towards objects when we learn them in the second half of the class. Try to follow the methods I give you even if you are an experienced procedural programmers
- ***Some rules for the class: NO GLOBALS, NO GOTOS***
  - If you don't know what these are that is a good thing.

Example: **Program to calculate various information about Circles**

Object Oriented Programming

- An ***Object Oriented Program*** defines an object as a conceptual type of data. This object contains both the general characteristics of the data object and set of operations that operate on the data object's characteristics

- Think of how we describe an objects in the real world. We categorize everything into objects. For example a car is an object. A car has characteristics such as a color, a model, and an engine. A car also has certain operations that can be performed on it such as acceleration and braking.
  - Object oriented programming breaks down programming operations by object rather than by procedure

- Think of common data types some different games might have (Asteroids for example)

- Object Oriented Programming is based on the idea of Classes. A class is a definition for an object that contains the general characteristics of an object and the operations that manipulate that object
  - Think of a Rectangle object
    - Some of its characteristics would be length and width
    - Some operations that could be performed on it would be calculating its area and perimeter

- Also keep in mind that an object oriented program can have multiple objects (classes) that interact with each other
  - We will go through an example design of an object oriented program with multiple objects in it when we get into object oriented programming later in the course

Object Oriented Programming

- ***Object Oriented Programming is one of the topics students find harder in this course***
  - Most of you probably have more experience writing procedural (step-by-step) programs than object oriented programs so far
  - By the end of this course I want you to see why object oriented programs are better and be able to design and write simple (to intermediate level) object oriented programs

- Even though you are writing procedural programs at the beginning of this course I want you to keep in mind that the object oriented programming we do in the second half of the class is what you will need in the real world
  - Why learn procedural first?
  - It tends to be easier to grasp basic programming concepts (selection/repetition statements) when programming procedurally
  - In order to program using object oriented programming you need to have these basic concepts down cold
    - C++ Syntax, Data types, Selection Statements, Repetition/Looping Statements, Functions
    - These basic concepts were covered in Java but there will be some differences in C+ which I will point out
    - These basic concepts will be in every language, so it's imperative you get them down

- ***Every programming concept in this class builds on what has been learned before it***
  - If you don't understand a concept ask questions because it will be necessary to understand that concept in order to get the next one

Object Oriented Programming

- Why Learn it?
- Most real world jobs will use object oriented programming
  - Why?
    - It saves software costs (we'll learn why & how later)
- Employers usually test you before they hire you for a programming job
  - I guarantee that test will include object oriented programming principles
- Object Oriented programming is especially useful when programming GUIs
  - Windows are objects that have certain characteristics: color, size, location etc.
    - Different operations can be performed on a window's characteristics
      - Resize, move, change color etc.
  - Windows can contain other objects such as menu's, tool boxes, panels etc.
  - Without an understanding of object oriented programming, understanding GUI programming would be very difficult
- In addition in order to program games you MUST understand object oriented programming
  - Ask any of the students who took my Python class

Approach for this Class

- **Since C++ is built on the procedural language C it is important to understand both the procedural and object oriented aspects of it**
  - You cannot write a C++ program without relying on some procedural code
  - In this class we will start by learning the procedural aspects of C++
    - Since you already have experience in a different language many of these concepts will be familiar
      - It's important to note the differences between the language you already know and C++
      - Since many of you already know Java, I will point out some of these differences as we go along
    - One you have a firm grasp of procedural programming in C++ we will expand that knowledge into object oriented programming
    - For now just know the basic differences between procedural and object oriented programming
    - Keep in mind at the beginning that you are learning procedural programming
    - We will go into detail and expand on the object oriented programming as the class proceeds

Review: Programming Fundamentals

- **What is programming?**
  - **A *program* is a set of instructions a computer follows in order to perform a task – it should work DIFFERENTLY on different data (think back – Gross Pay example from Java)**
    - ***Programming*** is the process of writing a computer program in a language that the ***computer*** can respond to and that other programmers can understand. These instructions to the computer and the associated rules are called a ***programming language***.

- **The Big Picture**
  - Programming requires **problem solving skills**
  - **You will usually be presented with a problem or some sort of desired end result**
    - "Can you make the computer do this?"
    - Most of your clients will not be programmers and many won't be intermediate or advanced users
  - As a programmer you need to understand the problem/goal and figure out a plan of how to solve the problem/reach the goal
    - ***Your first plan is not always the best one***
  - You will have two types of homework in this class
    - Homework where I give you the structure of the program
      - These are designed to make sure you understand syntax and basic concepts
    - Homework where I give you a problem and you write a program to solve it
      - There will be more than one correct solution to these type of homework assignments
        - Your goal will be to think about the problem and come up with the best and most efficient solution to the problem (you want your program to not just work but to work efficiently)

Review: Algorithms & Pseudocode

- As a software developer you must plan/design, code, build and test your programs.

- ***Algorithm development*** is an essential part of the programming process

- What is an algorithm?
  - **An *algorithm* is:** A step by step sequence of instructions for performing a task or solving a problem in a finite amount of time
  - ***It's imperative that you create a set of steps for a program before you sit down and start entering code***
  - There is (except with the most basic program) more than one way to solve a problem
    - So there is more than one algorithm that solves a problem
    - The first algorithm you think of is often NOT the best one
  - ***ORDER MATTERS IN YOUR ALGORITHMS – remember from CSIS 112 computers are not like humans and they must get instructions in the exact logical order needed***
    - If you didn't take CSIS 112 you can email me for the CSIS 112 Chapter 1 lecture slides which go over algorithms and why order matters in more detail

- ***Why is it not ok to "just have the code work"?***

Review: Algorithms & Pseudocode

- **Example:** Suppose you had to write a program that reads text from a data file and determines how many times the word sheep is found in that data file. How would you solve this problem?
  - At this point we're not thinking about C++ coding at all, just how the computer would solve this problem
  - There are several approaches (and several algorithms) that would solve this problem).
  - We will go over two different algorithms to solve the problem

- ***Algorithm #1***

1. Create a variable named *sheepCount* to keep track of the number of 'sheep' found and set it to zero
2. Read the first unread line of the data file
3. As the line from step 2 is read, search that line letter by letter in order for the letter 's'
4. If the letter 's' is found, check the character before the 's' to make sure it is blank .
   - If the character before the 's' is not blank
     - if the end of the line has NOT been reached return to step 3.
     - if the end of the line has been reached return to step 2 and read the next line of data
   - If the character before the 's' is blank check the first character after the 's' to see if it is an 'h'. If it is an 'h' check the first letter after the 'h' to see if it is an 'e'. Keep this process up until you find an letter that doesn't fit the pattern or you reach the letter 'p' in the correct sequence
   - If a match is found increase the *sheepCount* by 1
   - If a match is not found go back to step 3 if the end of the line is not reached, else go back to step 2
5. Repeat steps 2 and 4 until the end of the data file is reached
6. Display the number of time the word 'sheep' was found to the screen

- **QUESTION:** What do we call the above series of statements?
- Answer: ***PSEUDOCODE*** for an ALGORITHM

- **PSEUDOCODE IS NOT REWRITING YOUR ASSIGNMENT INSTRUCTIONS** (you will get points off for rewriting instructions)
  - Ask this question: How am I going to instruct the COMPUTER to SOLVE THE PROBLEM – what do I need to tell the computer to do?
  - Try starting each line with a verb or using some code like words (if, else, while) but not actual code
  - ***This is the style of pseudocode I taught in my CSIS 112 class – please follow this style even if you had a different teacher – I will just have you do pseudocode for one or two homework and if everyone is getting it we will stop after that***

Review: Algorithms & Pseudocode

- **Example:** Suppose you had to write a program that reads text from a data file and determines how many times the word sheep is found in that data file. How would the computer solve this problem?
  - At this point we're not thinking about C++ coding at all, just how we would solve this problem
  - There are several approaches (and several algorithms) that would solve this problem).
  - We will go over two different algorithms to solve the problem

- Algorithm #2
  1. Create a variable named *sheepCount* to keep track of the number of 'sheep' found and set it to zero
  2. Create a String type variable (a memory location) named *testWord* and have that variable be equal to "sheep"
  3. Read the data file one WORD at a time
  4. Check to see if the word read in is equivalent to the *testWord* variable

     (notice there is no mention of HOW this will be implemented with code – Why? Pseudocode should be able to be used with any programming language)
  5. If the words are equal increase the *sheepCount* variable by 1
  6. Repeat steps 3-5 until the end of the data file is reached
  7. Display the number of time the word 'sheep' was found to the screen

- **QUESTION:** What do we call the above series of statements?
- Answer: ***PSEUDOCODE*** for an ALGORITHM

Review: Algorithms & Pseudocode

- ***You must develop an algorithm before you start your code***
  - Only after an understanding of the problem, the data needed for the problem, and the algorithm that is going to solve that problem is acquired can a program be written

- Algorithms can be written or described in various ways
  - In this class we will use pseudocode to describe our algorithms
  - ***Pseudocode:*** **English like phrases or descriptions to describe the algorithm**
    - **PSEUDOCODE DOES NOT NEED TO BE PAGES LONG – A SIMPLE STEP BY STEP SET OF ENGLISH INSTRUCTIONS IS SUFFICIENT**
    - **PSUEDOCODE DESCRIBES AN ALGORITHM NOT A PROGRAM**
      - ***If your pseudocode starts like this "This program does….." you will not get credit for the pseudocode – this is repeating the instructions***
      - Why?
        - We need to understand the difference between pseudocode and a program

- ***What is the difference between a pseudocode and a program?***
  - Pseudocode describes an algorithm (a finite series of steps to solve a problem)
  - A program implements an algorithm in a specific programming language
    - All programmers have a different programming style so even if the same algorithm is implemented to solve a problem, the resulting programs will still be different

Review: Algorithms & Pseudocode

- Pseudocode for a problem involving how to calculate someone's Gross Pay
  - High Level Pseudocode (not what I'm looking for but still more detailed than "human" instructions)
    - Get Payroll Data
    - Calculate Gross Pay
    - Display Gross Pay
  - Detailed Level Pseudocode
    - Create variables: hours, rate, pay
    - Display *How Many Hours Did You Work?*
    - Store user input in the *hours* variable
    - Display *How Much Did You Get Paid Per Hour?*
    - Store user input in the *rate* variable
    - Store the value of *hours* times *rate* in the *pay* variable
    - Display the value of the pay variable

Programming Fundamentals

- Coding an Algorithm
  - Only after an algorithm has been selected (more than one algorithm can solve a problem) and understood by the programmer (using pseudocode or a flowchart) can the program be coded
    - There are different algorithms that can be used to solve problems in a program.
      - ***It is the job of the programmer to determine the algorithm that:***
        1. **Solves the problem correctly**
        2. **Solves the problem efficiently**
      - **THE ALGORITHM MUST DO BOTH OF THE ABOVE – Just compiling is NOT enough!**
  - Coding: the writing of a program using computer language statements
  - Look back to our translation process on slide # 14, we are on the FIRST step, keep in mind the computer doesn't understand C++ until it is translated to machine language and the linker is run to create the executable code

C++ Program Structure

- Every C++ program consists of several basic components whether the program is complex or consists of just a few lines of code

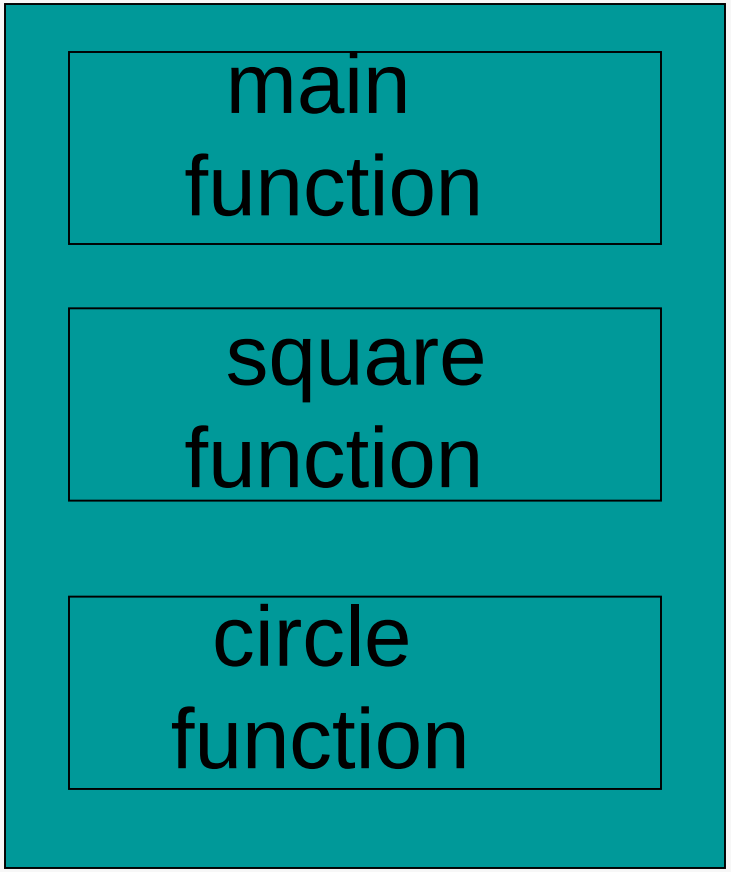
- A couple of fundamental C++ concepts

1. C++ is ***case sensitive***
   - This means C++ knows the difference between upper case and lower case letters
   - Every character in C++ is important, so pay attention to every symbol and every letter (whether uppercase or lowercase) – DETAILS MATTER
   - The language will view the following names as three separate things:
     - total
     - Total
     - TOTAL

2. A ***function*** is a block of code that has a specific format with an entrance point and an exit point
   - Functions are an essential part of procedural programming and allow us to build code blocks that do program tasks
   - We'll start by writing procedural programs that are just one big function
   - Better definition of a function:
     - A miniature program that optionally takes some input data processes it in come way and optionally gives some output data
     - For the CSIS 112 people – this is the C++ equivalent of a METHOD

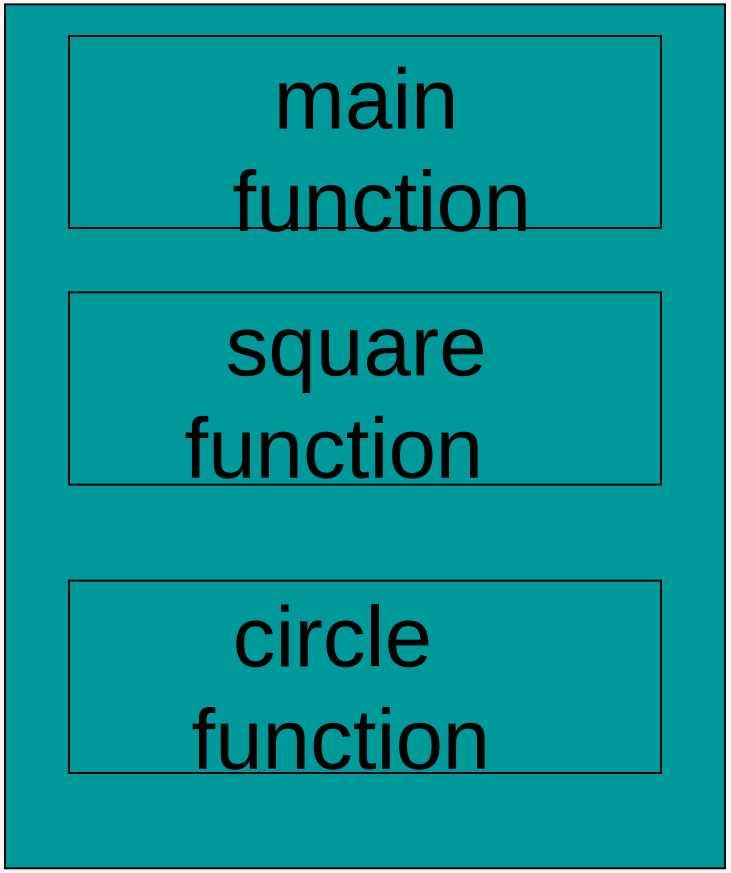
C++ Program Structure

- In C++ a ***function*** is a set of C++ statements that perform a ***particular task*** (in Java this is called a method)
  - Each function is similar to a small machine that transforms the data it receives into a finished product
    - Another key aspect of procedural programming
  - Formally, a function is a self-contained block of code with a unique name that performs a task or set of tasks
    - For all functions other than main, a C++ program executes a function by calling it (using a specific command code to make it execute)
    - A function is called by using it's name
- In ***procedural programming***, think of a collective group of functions as subprograms that work together to perform the overall task of a program
  - All of the functions work together collectively to form the entire program that solves the problem
  - A C++ program is a collection of one or more functions



C++ Program Structure

- Every C++ program must have a ***main function***
  - A C++ program can have ***one and only one*** main function
  - Think of the main function as the master function of the program and the other functions as the servants
    - The main function not only drives all of the other functions but in object oriented programming the main program is also in charge of the class data types used in a program (when they are instantiated and when they are used)
  - The main function can also be thought of as the driver function because it drives the other functions by telling them when and in what sequence they will execute
  - So the main function is automatically called (executed) in a C++ program but the other functions (circle and square in this example) are only executed when they are called by the main function



First C++ Program

- **A Simple C++ Program**
  - **Example: (helloWorld.cpp – Example 1)**

```
//A first C++ Program
//PSEUDOCODE:  Display Hello World on Screen

#include <iostream>
using namespace std;

int main()
{
  cout << "Hello World!";
  system("PAUSE");
  return 0;
}
```

Comments

- The first line of this program is a ***comment***
  - `//A first C++ Program`
  - ***Question:*** Is this a useful comment? Why or why not?

- Comments are explanatory remarks written within program
  - Clarify purpose of the program
  - Describe objective of a group of statements
  - Explain function of a single line of code or section/lines of code

- Computer ignores all comments
  - Comments exist only for convenience of reader

- A well-constructed program should be ***readable*** and understandable
  - Comments help explain unclear components
  - Why is it important for code to be readable? Why might "tricks" like Gotos and globals make a program less readable? (if you haven't programmed before ignore the second question)

- ***As a rule you should always comment your program comprehensively but at the same time do not make unnecessary comments***
  - Your comments should be written so that another programmer or you at a later date can glance at the comments and understand how a particular piece of code works and what your goal was in writing that particular piece of code
    - Multiple programmers will often work on a project and it saves a lot of money if your code is easy for the new programmer to understand and continue working on
    - It's also important for you to be able to return to your program at a later time and quickly and easily understand your intentions for each piece of code in the program

Comment Structure

- **There are two types of comments: line comments and block comments**

1. ***Line comment:*** Begins with 2 slashes(**//**) and continues to the end of the line
   - Can be written on line by itself or at the end of line that contains program code

```
//  this is a line comment
```

2. ***Block comment:*** Multiple line comment begins with the symbols **/*** and ends with the symbols ***/**

```
/* This is a block comment that
   spans
      across three lines */
```

- To embellish or highlight certain comments above functions, many programmers use a frame around block comments

```
/**************************************************************
 A function and it's description
**************************************************************/
```

The `#include` Directive – Header Files

- The second line of the example program is:

```
#include <iostream>
```

- This is called a ***directive*** (to the preprocessor) because it directs the compiler to do something
  - The `#include` directive directs the compiler to include the contents of the file in **<>** ***before*** compilation
    - Remember the compilation process (Slide 14 – remember step 2, ***preprocessor***)
    - The preprocessor performs an action ***before*** the compiler translates source code to machine code
  - In this example: `#include <iostream>` Causes the `iostream` file to be ***inserted*** (basically copied into the spot) where the `#include` command appears

- `iostream` is part of the C++ standard library
  - Included in `iostream` are two important ***classes***:
    - A class is a piece of code that contains data and actions to be performed on that data
    - `istream`: Declarations and methods for data input
    - `ostream`: Declarations and methods for data output
  - If we didn't include the contents of `iostream` into this program it wouldn't compile because `iostream` contains the code definitions needed to use input and output statements
    - `cout` is an object of class `ostream` that represents the standard output stream
- Libraries are very important, the code for input/output is complex

Note: **there ARE more preprocessor directives than just `#include`**

The `#include` Directive – Header Files

- The **`#include`** directive is a type of ***Preprocessor Directive***

- Peprocessor directives are not C++ statements, they are signals to the preprocessor that run prior to the compiler
  - #include type directives (and other preprocessor directives) make life easier for the programmer
  - Example: Any program that uses the **`cout`** object must contain extensive setup information contained in **`iostream`**. It would be very time consuming for the programmer to have to type or even cut and paste all of this information into their program

- The **`#include`** preprocessor directive must always contain the name of a file
  - The preprocessor inserts the entire contents of a file into the program at the point it encounters the **`#include`** directive
  - The compiler itself doesn't actually see the **`#include`** directive, instead it sees the information that was inserted by the preprocessor as if the programmer had typed it there themselves

- The **`iostream`** file is referred to as a ***header file*** because a reference to it is always placed at the top of a C++ file using the **`#include`** command
  - The information in this header file is C++ code
  - Later you will learn out to make your own header files

Namespaces

- The next line in the Hello World program is:
  - `using namespace std;`

- The ***standard library*** in C++ (it is part of the C++ ANSI standard) is an extensive set of programming routines that have been written to carry out common tasks – basic to complicated programming tasks that as a programmer you need to perform over and over again
  - From Wiki: In C++, the **C++ Standard Library** is a collection of classes and functions, which are written in the core language and part of the C++ ISO Standard itself[

- Since there are a large number of routines and files in the standard library that use many different names it is possible when you are writing your program you will accidentally use one of the names defined in the standard library for your own purposes (Example: `cout`)

- A namespace in a program is used in C++ to avoid problems that can occur when duplicate names are used in a program for different things.
  - A *namespace* does this by associating a given set of names (such as the ones from the standard library) with a group (family) name such as which is the namespace name. **The `iostream` libraries are associated with the standard namespace (`std`).
  - Every name that is defined in the code that appears in a namespace also has the namespace name associated with it.
    - In the example program the name `cout` is associated with the namespace `std`
    - So the true name for `cout` is actually: `std::cout,`
      - `USE NAMESPACE in your programs! It is ANSI Standard, do not use std::cout!`

Namespaces

- Every name that is defined in the code that appears in a namespace also has the namespace name associated with it.
  - In the example program the name **`cout`** is associated with the namespace **`std`**
  - So the true name for **`cout`** is actually: **`std::cout`**
- The two colons that separate the namespace name **`(std)`** from the name of the entity **`(cout)`** are called the **scope resolution operator**.
  - We'll discuss the scope resolution operator later in the semester
- So technically the Hello World program could have been written as follows (***However this is NOT OK for this class and is NOT ANSI Standard***):
  - **Let's rewrite our original program like this**

```
//Hello world program with no using namespace std statement
#include <iostream>

int main()
{
  std::cout << "Hello World!\n";
  return 0;
}
```

- Notice that in this revised version of the program no **`using namespace std;`** statement appears at the top of the program
- In this OUTDATED program with no **`using namespace std;`** statement, every time **`cout`** or any name associated with the standard library (which is defined by the standard namespace was used) the name of the namespace std followed by the scope resolution operator would have to precede it
  - As you will see in this class we will use the standard library a lot, particularly the `iostream` file for input and output so using **`std::`** with all of the different names associated with input and output could lead to having some cluttered code.

Namespaces – The Using Directive

- So why didn't I use the `std::` in front of `cout` in the original Hello World program?

- The using declaration:

  ```
  using namespace std;
  ```

  - is used to tell the C++ compiler that the program intends to use all of the names from the standard library without specifying the namespace name.
  - Once this command is used in a program, the compiler assumes that every time `cout` or any other name in the namespace `std` is used that the intention is to use the name from the standard namespace rather than a name defined in the program itself

- In this class we will use this using directive in almost all our programs since we are going to be using the standard library for input and output extensively
  - ***AGAIN, the using directive is STANDARD C++ - you must use the directive, do NOT use the OLD style in your programs***

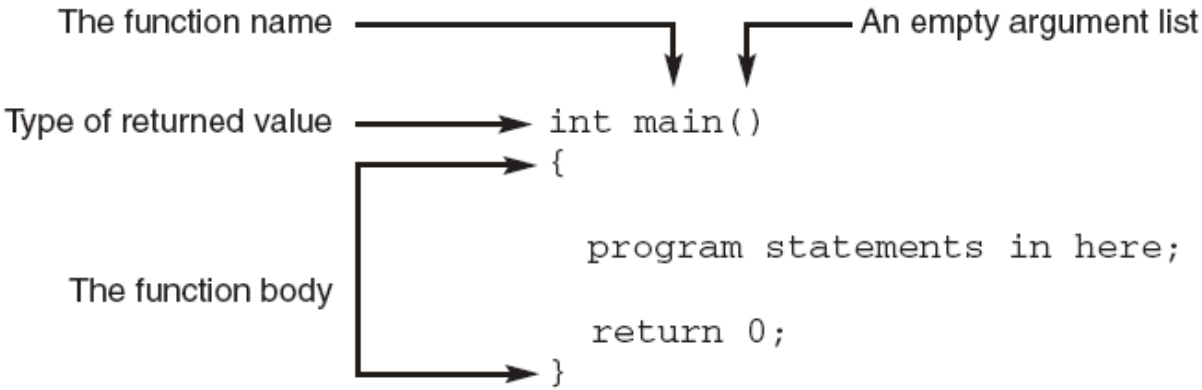
The `main` Function

- The next line in the Hello World program is: **`int main()`**
- **`int main()`** marks the beginning of a function, in this case the main function
  - Each C+ program must have one and only one function named **`main`** **(CASE SENSITIVE!)**
  - **`main`** is the starting point of the program
  - Another way of thinking of a function is as a group of one or more programming statements that collectively has a name – ***a function is a set of code that takes input (optional), processes it in some way and produces output (or a result)***
  - The name of this function is **`main`** and the set of opening parenthesis that follows the name indicates that it is a function
  - The word **`int`** stands for integer and indicates that the function sends an integer value back to the operating system when it is finished executing
    - Remember a function takes in data input (optional) does something and then returns data output (optional)
- The next line of the program is an open curly brace **`{`**
  - This is called a left brace or opening brace and is associated with the beginning of the function main
    - It marks the beginning of the function main's function statements
      - These function statements are also known as the function body
    - **All function statements that make up a function are enclosed in a set of curly braces (be very careful about the difference between curly braces and parenthesis)**
      - The second brace, right brace in our program indicates the end of the main function
      - Every function must have both an opening and closing brace (right/left)
        - Everything between the two braces are the contents of main (main's function body)
- **Curly braces are used to enclose the body of all functions.**
  - When any function executes, ***it starts executing right after the beginning curly brace and continues executing sequentially until it reaches the closing curly brace*** (this means the code can't look ahead and can't look back)
  - Make sure and make the opening and closing curly braces of EVERY function easy to identify
    - See my format and how they line up and are easy to recognize

The `main` Function

- First line of function is called ***header line***
  - What type of data, if any, is returned from function
  - The name of function
  - What type of data, if any, is sent into function
    - Data transmitted into function at run time are referred to as ***arguments*** of function
- Keep in mind that a procedural program is a set of functions working together
  - Every program has a main, but you can also design your own functions to work with main
- The parts of the program we have talked about so far are a framework for your program
  - The only part of the basic framework we haven't talked about is the return zero which we'll get to in a minute

FIGURE 1.10 *The Structure of a* `main()` *Function*



The `cout` Object

- The **`cout`** object sends data to the standard output display device
  - Remember **`cout`** is defined in the **`iostream`** library
    - Basically when you use **`cout`** you have an instance of an ostream class that defines the data and methods you need to use **`cout`** for simple input and output in your programs
  - The display device is usually a video screen
  - Name derived from ***Console OUTput*** and pronounced "see out"

- Data is passed to **`cout`** by the insertion symbol **`<<`**

```
cout << "Hello World!";
```

- Notice that the message **`"Hello World!"`** is printed without the quotation marks
  - The quotation marks are there as a signal to the compiler that is ***String type data*** (strings are sequences of characters), ***the*** **`cout`** ***object requires that you send it String type data***

- **`cout`** is classified as a stream object which means it works with streams of data
  - To print a message on the screen you send a stream of characters to **`cout`**
  - The **`<<`** is called the ***stream-insertion operator***.
    - The information immediately to the right of the operator is sent to **`cout`** and displayed on the screen

- Note that this is the only line in our program that causes anything to display on the screen
  - The other lines are necessary for the framework of the program but do not cause any screen output
    - It is the line that actually does something in our very simple program

system("PAUSE");

- This command is what makes the command prompt (DOS) screen pause when your program is finished running
  - Without this line of code your program will run really quickly and close and you won't be able to see the results

The `return` Statement

- The `return 0;` statement at the end of the program sends the integer value `0` back to the operating and indicates the successful completion of the main function
  - Remember the function header said that the main function returned an integer value upon its completion
    - Any function that has a header indicating a value must be returned from it must have a return statement, otherwise a compiler error will occur
      - A keyword immediately before the function name defines the data type of the value returned by the when it has completed its operation.

- For now, the important thing to understand is that the main function must always end with a `return` statement (`return 0` for now) and a closing curly brace `}`

- So let's write our simple program framework on the board

Program Statements

- Remember the body of the one function in the Hello World is the code enclosed in the curly braces after the function header `int main()`

- The lines of code that make up the body of the main function are called ***program statements***
  - A program statement is a basic unit of code defining what a program does
  - One or more program statements make up a program
  - The action of a function is always expressed by one or more program statements each ending in a semi colon

- **It is a *semicolon* that marks the end of a program statement NOT the end of a line**
  - A program statement can then be spread over several lines if it makes the source code easier to read
  - Multiple statements can also appear on one line if this makes the source code easier to read
  - Remember the goal is code that other programmers can easily interpret and understand

Whitespace

- ***Whitespace*** is the term used in C++ to describe blanks, tabs, newline characters, form feed characters, and comments

- One purpose whitespace serves is to separate one part of a statement from another part of a statement
  - This enables the compiler to determine where one element of a statement begins and another ends
    - **For example:**
      - `return 0;`
        - compiles – legal statement
      - `return0;`
        - does not compile, compiler can no longer determine that return and 0 are separate statements

    - **Notice that whitespace is not necessary between `cout` and `<<` or between `<<` and the `"` in `"Hello"`**
      - This is because these characters are not combinations of alphanumeric characters together
      - However for readability, you should include spaces between these types of code

- ***Other than for the purpose described above (to separate elements in a statements that might be confused) whitespace has no effect on the way the compiler reads code.***
  - Whitespace is used by programmers to make source code **more readable** by other programmers so it can be reused, debugged, and easily modified.
  - In this class you MUST write readable code, I should easily be able to look at your program see where a function begins and ends and see what's going on in the program (what your intentions were)

Statement Blocks

- When one or more program statements are enclosed between a pair of curly braces they become a ***block*** or ***compound statement***

```
{
        program statement;
        program statement;
        program statement;
        ….as many more as needed;
}
```

- The body of a function is an example of a block.
- Wherever you put a single statement in C++ you **could** also put a block of statements between curly braces `{ statement(s); }`
- Statement blocks can be placed inside other statement blocks to any depth
- As you will see later in the course ***statement blocks have important effects on both variables and selection and repetition statements as well as on functions – curly braces define the scope (visibility) of statements***

C++ Basic Formatting Summary

- **A semicolon marks the end of a complete statement in C++**
  - Comments are ignored by the compiler so they do not require a semicolon
  - Preprocessor directives do not end with semicolons. Since preprocessor directives are not C++ statements, they do not require them. In many cases an error message will result if you terminate your preprocessor directive with a semicolon
  - The phrase `int main()` is the beginning of a function, not a complete statement and you should not use a semicolon after it
  - Function bodies `{}` also do not end with semicolons
- **C++ is a case sensitive language.**
  - `int MAIN()` is not the same as `int main()`
- **Every opening brace `{` in a program must also have a closing brace `}`**
  - You should write your code so one can EASILY identify the beginning and ending curly brace for a function (or other code block) – one way is to line them up like I do
  - Curly braces denote the beginning and end of a block of code (a function in our case right now) – think of it like a beginning must have an end
  - Notice this rule holds for the smooth braces `()` as well, they denote input (arguments), you must define where the arguments begin and where they end
- **Also note that a function header or any part of a function's structure is not followed by a semicolon – structure is not a statement**
- **You'll get used to the semi colons more by practicing code**
- **It is important to have useful comments in your program.**
  - Makes the program more readable and easier to understand

Running the Program using Visual C++

- What do we mean by IDE?
  - **IDE stands for Integrated Development Environment**
  - It's important for a programmer (software developer) to understand the tools of their development environment
    - Think of a construction worker that knew how to build a house but didn't know how to use the tools
  - Think of the IDE as a "workbench" for programmers

- IDEs provide an integrated set of tools for writing, editing and debugging programs
  - Editor
  - Compiler
    - Usually a window lists compiler errors
  - Linker
    - Hooks the object code(s) and the library code together
    - Builds and executable file to run the program
  - Debugger
    - Allows the programmer to single-step through the program, providing the ability to watch variables and follow the flow of the program
    - Breakpoints can be set in the program so the program will stop at certain points the programmer wants to focus on
  - Help

Large Project Size Issue

- Unlike in Java we're going to be creating professional projects in this class – a project is made up of your .cpp and .h files (C++ files) as well as a debugger and various other links – so it's a bunch of files combined together
  - You'll see what I mean as we go through how to create our first Hello World project
  - It will seem hard at first but once you do HW #1 (you'll get to practice making 6 small projects) you will get it and it will be easy

- In Microsoft Visual Studio 2013 and beyond the developers did something different that made all standard projects link to a large number of libraries and have a large number of external dependences
  - The did this to try to make the program more convenient (and it may have been good for professional developers)
  - For us, it caused a problem because it made project more than THREE TIMES as large as they used to be (even projects with only one .cpp file!!! – even when compressed they are too large)

- As a result I can't keep example projects on Moodle – HOWEVER, what I can do is put the .cpp and .h files that make up the projects on Moodle and then you can recreate them at home (you can recreate them with Xcode or Visual Studio)
  - After tonight and HW #1 you will understand how to do this
  - What I would recommend is going through the examples for the Chapters we are covering for the week BEFORE class and creating projects, that way it will be easier to follow along
    - The other option is to just look at the individual files as I run the projects up front

- ***NOTE:*** Even if you are going to use XCode you are REQUIRED to use Visual Studio C++ for programming exams, so make sure you are following along

Running the Program using Visual C++

- We're going to use Microsoft's Visual Studio IDE (2013 Community Version)
  - ***At school (for labs and exams) you will use the Microsoft Visual Studio Professional Edition (2013) installed on the lab computers – at home you will have the Community version – they are very similar and compatible do not worry***
    - I DO NOT RECOMMEND THE 2015 version (and definitely NOT the 2017 version) – try the 2013 version first on your PC and if that doesn't work then try 2015 as a last resort
- AT HOME, you should have:
  1. **Microsoft Visual Studio COMMUNITY 2013**
     - Can be downloaded at home for free on the web (EVERYONE WITH A PC IS REQUIRED GET THIS!) – see slides.- keep going forward
     - NOTE: If you have Windows 10 and the 2013 gives you an issue try the 2015 version but do try 2013 first
  2. **OR: X Code (get it free from the App Store – Everyone with a Mac is REQUIRED to get this)**
- So you may be using the Community Version at home and the Professional version at school, OR you may be using Xcode at home and Visual Studio Professional at school
  - ***This is NOT an issue, the two Visual Studio versions ARE compatible, just make sure at home you have the 2013 Community Edition – they are compatible – if you end up with 2015 – you can still do it just bring your .cpp and other files to school and re-create the project here***
  - ***Also if you're using Xcode you simply can bring your .cpp file or files to school and then create a visual studio project here***
- If you have a Mac I do NOT recommend trying to install bootcamp or Parallels and using Visual studio UNLESS you already have Windows – I recommend Xcode instead – I cannot provide support for bootcamp or Parallels
- **THE SLIDES SHOW THE PROFESSIONAL VERSION OF VISUAL STUDIO – remember if you have the Community Version it's similar enough you can follow along – NOTE the colors may look a bit different but that's shouldn't matter the functionality should be exactly the same**

Running the Program using Visual C++

- Why Microsoft Visual C++?
  - While there are many good IDEs out there this is one that is used often in the real world, so whether you're a Microsoft fan or not, it's important to know how to use this tool
  - It can be used to compile any C++ program that adheres to the ISO Standards
- Supported Operating Systems – NOTE YOU DO NEED Windows 7 or Higher
  - Windows 10 is a free upgrade from Windows 8 I believe
- If you have a Macintosh again I want you to use Xcode (download and install from App store) unless you already have parallels and Windows and want to do it that way (if you don't already have it it's too much work and you won't get set up in time so use Xcode)

For PC Users:
Installing Visual Studio Community 2013

- ***THIS IS THE FREE VERSION OF VISUAL STUDIO – ALL PC Users need this at home for this class***
- **Setting up *Visual Studio Community* at home (everyone with a PC should do this ASAP)**
  - ***Go to the website and download Microsoft Visual Studio Community 2013 Version***
    - https://www.visualstudio.com/en-us/news/releasenotes/vs2013-community-vs
    - USE THIS LINK!! Only try the 2015 version if the 2013 version doesn't work (2013 is less buggy and has better compatibility with the school computers0
    - Choose the "Install Now – English" button
      - *Choose English as your language – if you don't I won't be able to help you with compilation errors (this is not good)*
    - For the install/download
      - ***Registration is free but required after 30 days***, just follow the instructions and register within 30 days
      - It's a fairly easy install assuming you have Windows 7 or higher
  - ***DO NOT FOR ANY REASON INSTALL VISUAL STUDIO "CODE" it is NOT the same thing!!!***
  - ***You definitely need Windows 7 or higher to install this software***
    - ***Remember I will get you access***

Visual Studio Community 2013

Last Update: 9/26/2016

November 12, 2014

Visual Studio Community 2013 is a new edition that enables you to unleash the full power of Visual Studio to develop cross-platform solutions. Create apps in one unified IDE. Get Visual Studio extensions that incorporate new languages, features, and development tools into this IDE. (These extensions are available from the Visual Studio Gallery.) Find out more details about Visual Studio Community 2013 here.

Download Visual Studio Community 2013.

What's in Visual Studio Community 2013

- Professional-grade editing, code analysis, and debugging support

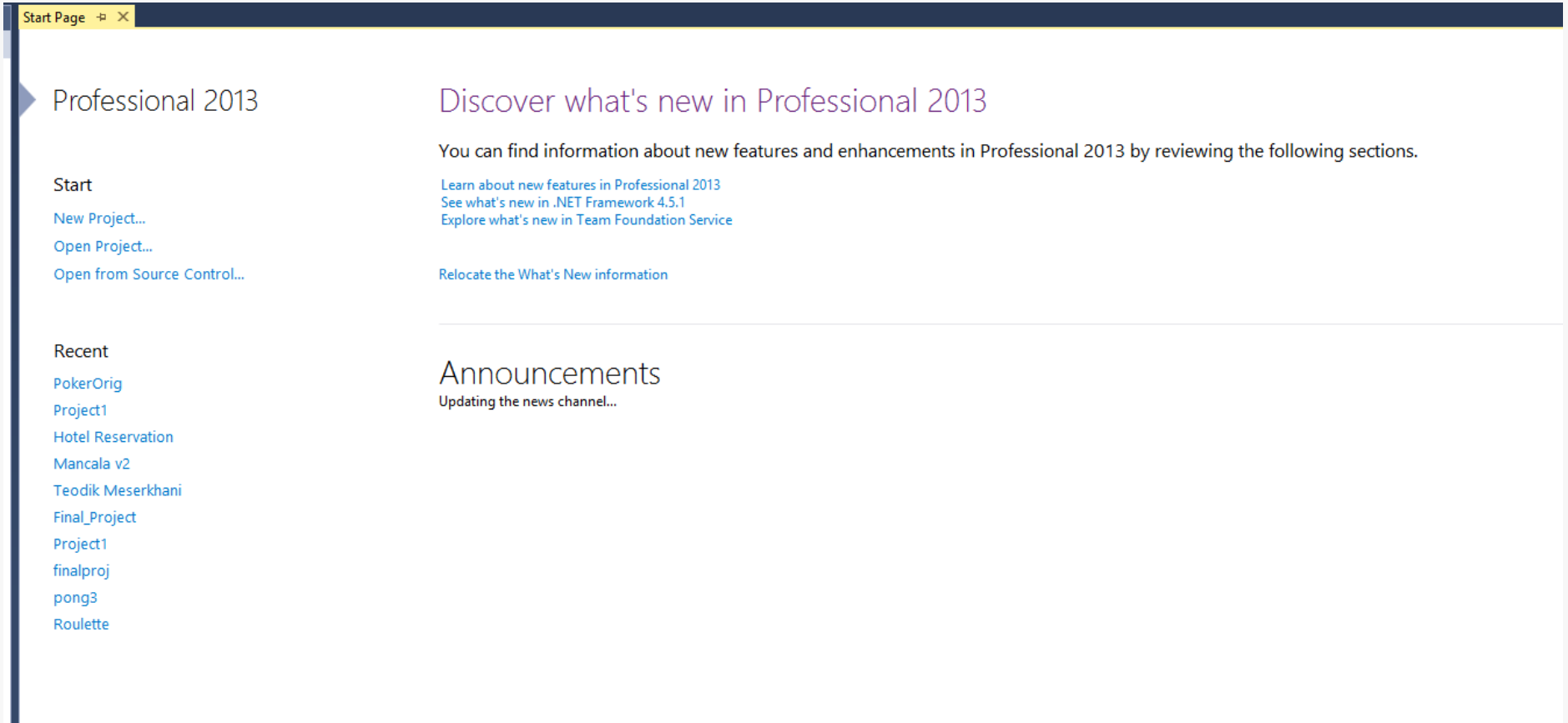
Running the Program using Visual C++

- **Using Microsoft Visual C++**
  - Projects and Solutions
    - When you build programs in Visual C++ you work on a project and build a solution
    - For the programs we will create, the C++ source code file(s) is/are located in the project folder
    - The majority of C++ programs consist of many source code files rather than just one (at the beginning of the class we will use just one file to learn with)
      - This is why the Visual C++ software requires the programmer to create a project and to add or create source code files into that project
        - The IDE keeps track of the entire set of program files and the compile, link and run steps are also performed in the project environment

- Now I'm going to go over how to use the ***Microsoft Visual Studio 2013 Professional (what we have here at school – the 2013 Community Version works almost exactly the same – the differences are minimal)*** software to create a project, create a file into that project (we're going to create *HelloWorld.cpp*), and compile, link, and run the resulting program
  - do not panic these instructions will work the two versions are similar enough

Running the Program using Visual C++

- **Using Microsoft Visual Studio 2013 Professional**
  - **Step 1:** Start the program – you can use the ICON or navigate from the start menu depending on your operating system
    - It may take a minute to load to start the first time you use it
    - You can just “x” out of the start page you don’t need this (unless you like it and want to use it – I don’t)



Running the Program using Visual C++

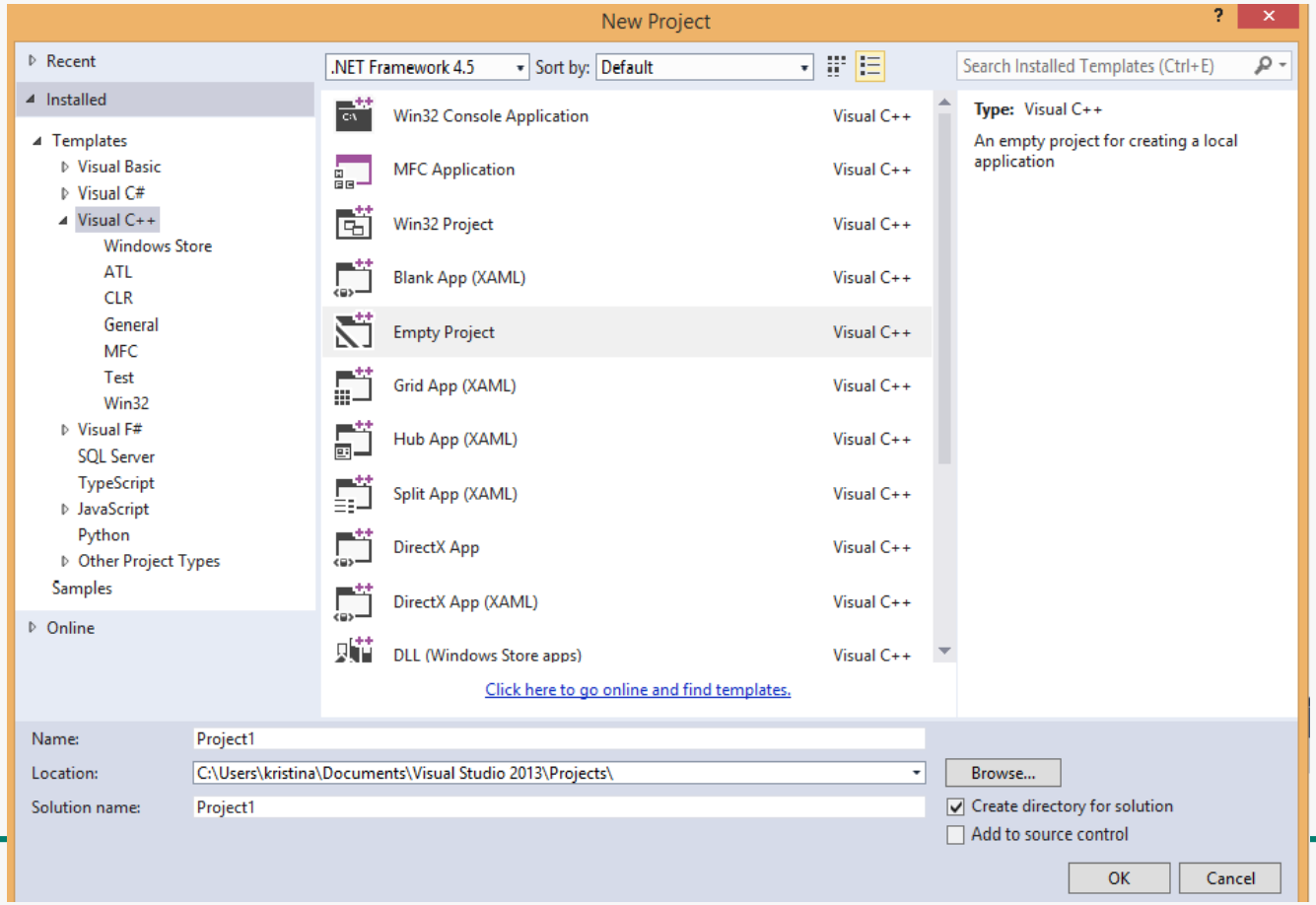
- **Using Microsoft Visual C++**
  - **Step 2: Create a New Project**
    - In order to code and run the HelloWorld.cpp program it must be contained in a project
      - A project must be set up before a program source code can be typed in (or added), compiled, and run
      - The project will contain the source code (.cpp) files as well as any other files (like .h) files we need for the program
        - For now our projects will just contain simple .cpp files
    - To create a new project, use the File menu to select:
      - **File->New->Project**
    - You should see the New Project Window shown here
  - **NOTE: You only need to do this step *ONCE FOR EACH PROJECT***
  - ***VERY IMPORTANT NOTE:* YOU NEED TO CHANGE THE SETTINGS TO CREATE**
    1. ***A C++ PROJECT***
    2. ***AN EMPTY PROJECT***

- ***IMPORTANT:***
  - ***SEE THE NEXT SLIDE….***
    - ***DO NOT USE THE CLR DO NOT CREATE ANYTHING BUT A C++ PROJECT (an EMPTY ONE)***
    - ***You need to CHANGE the settings on this SCREEN BEFORE YOU PROCEED!!!!!!! – DO NOT USE THE DEFAULT SETTINGS - CHANGE THESE TO THE SETTINGS SHOWN ON THE NEXT SLIDE!***

- ***You also may want to change the path of where the project is being saved***

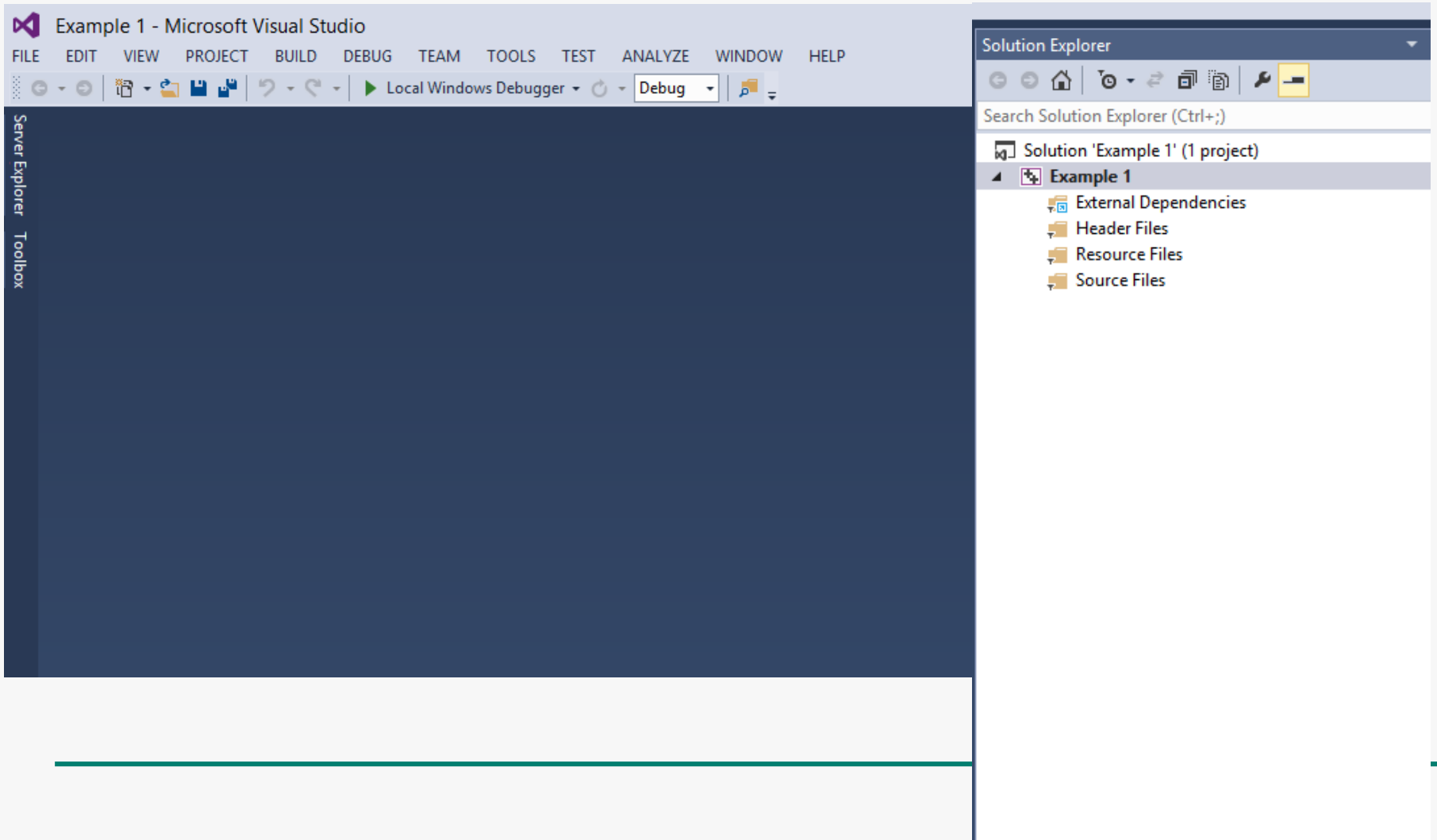
Running the Program using Visual C++

- **Using Microsoft Visual C++**
  - **Step 2: Create a New Project (continued)**
  - In the New Project Window, select **Empty Project** in the right pane
  - Next type in the name of your project, as you type in the name of your project it will be echoed in the Solution Name Field
    - Do NOT use the DEFAULT project name (“Project1”)
  - **Finally, use BROWSE to change the location of your project**
    - Note Visual C++ has a default location, if you don't change it your project will be saved in that default location and you might lose it
  - When you're finished select OK to continue



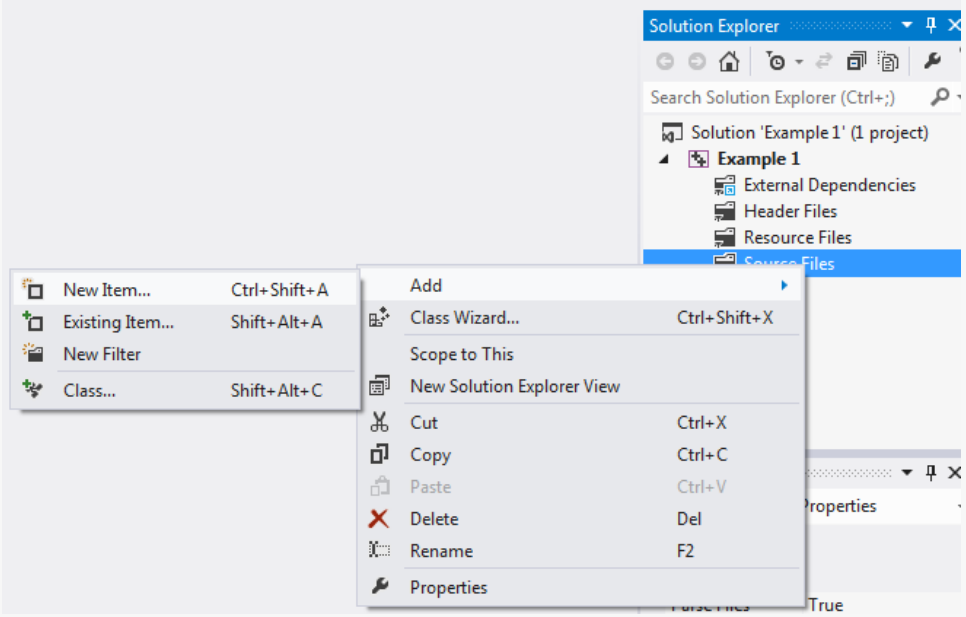
Running the Program using Visual C++

- **Using Microsoft Visual C++**
  - **Step 2: Create a New Project (continued)**
  - Your screen should look like the one below
  - Make sure the Solution Explorer is showing – if it's not choose View->Solution Explorer



Running the Program using Visual C++

- **Using Microsoft Visual C++**
- **Step 3: Create a New Source File (.cpp) and Type in Your Code**
  - When you finish these steps you will see your Visual C++ Solution. Note that there are folders for Header, Resource, and Source files in the window on the left (as well as External Dependencies)
    - Don't worry about Header, Resource, or External Dependencies folders for now
    - To create a new source file, right click on the Source files folder in the solution window on the left of the screen
    - You'll see a popup menu, from that menu select: Add: New Item
  - My colors here may be different because some of the slides are older – don't worry about colors
  - **To add a source file, right click Source Files then Add->New Item**

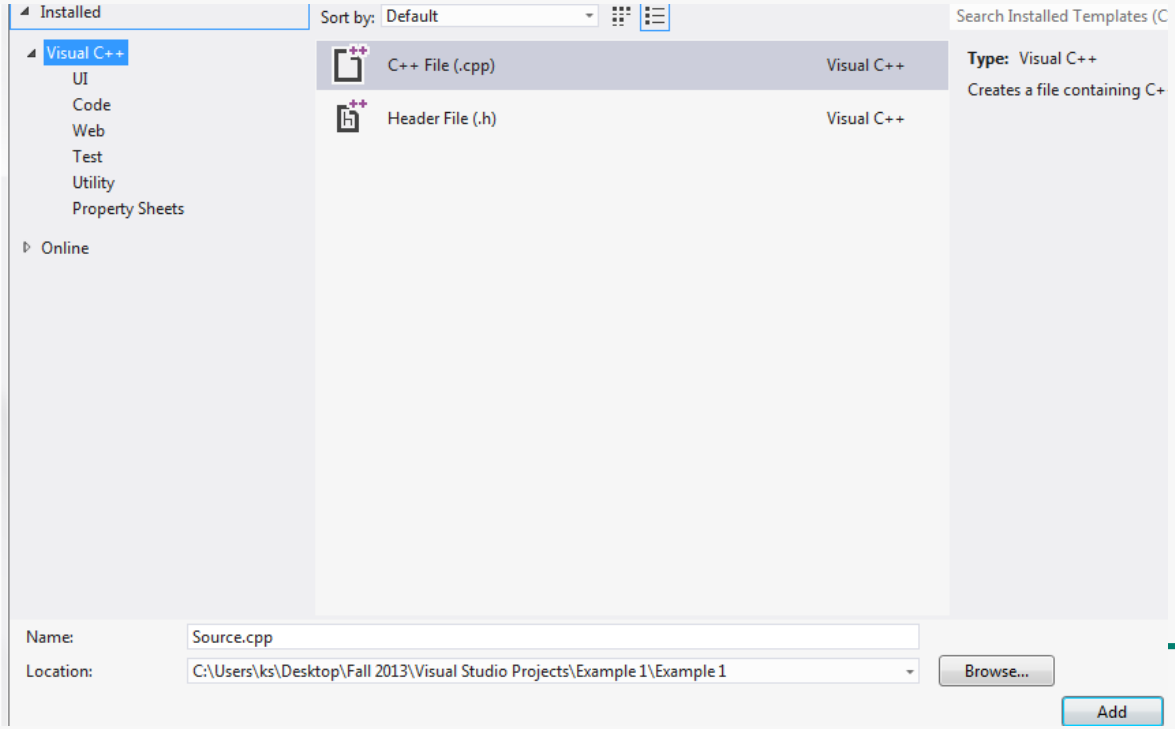


- You always need to add at least one source file to your project!!
  - This source file is your C++ code that you will compile, link and run
  - So this step always happens

If you don't see the solution explorer window go to View->Solution Explorer from the menu tabs at the top of the screen

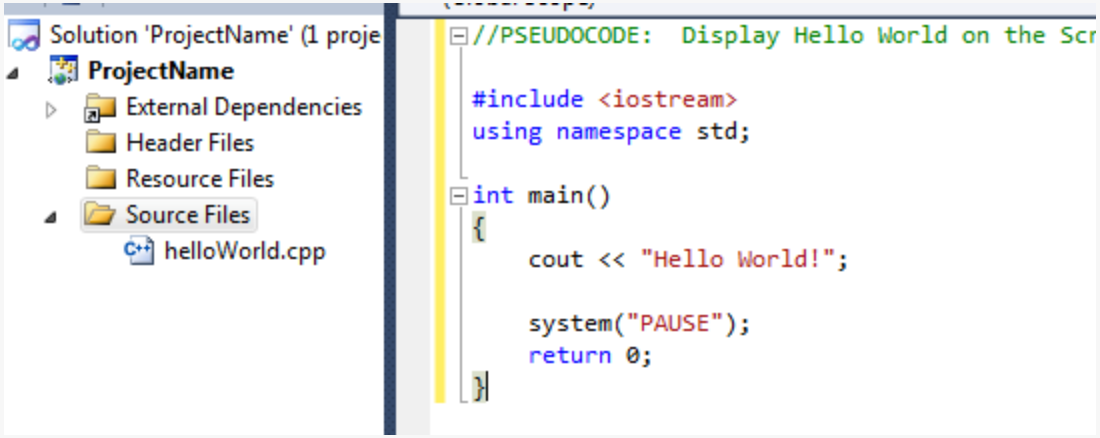
Running the Program using Visual C++

- **Using Microsoft Visual C++**
- **Step 3: Create a New Source File and Type in Your Code (continued)**
  - In the Add New Item Window
    - Select Code for the Category
    - Select C++ File(.cpp) for the Template
    - Type in the name of the file (In this case "HelloWorld.cpp")
      - Do NOT keep the DEFAULT file name Source.cpp
    - Select the Add Button
- ***Make SURE AND SELECT THE RIGHT FILE TYPE***
  - You are creating a .cpp file, .cpp is the extension for a C++ file



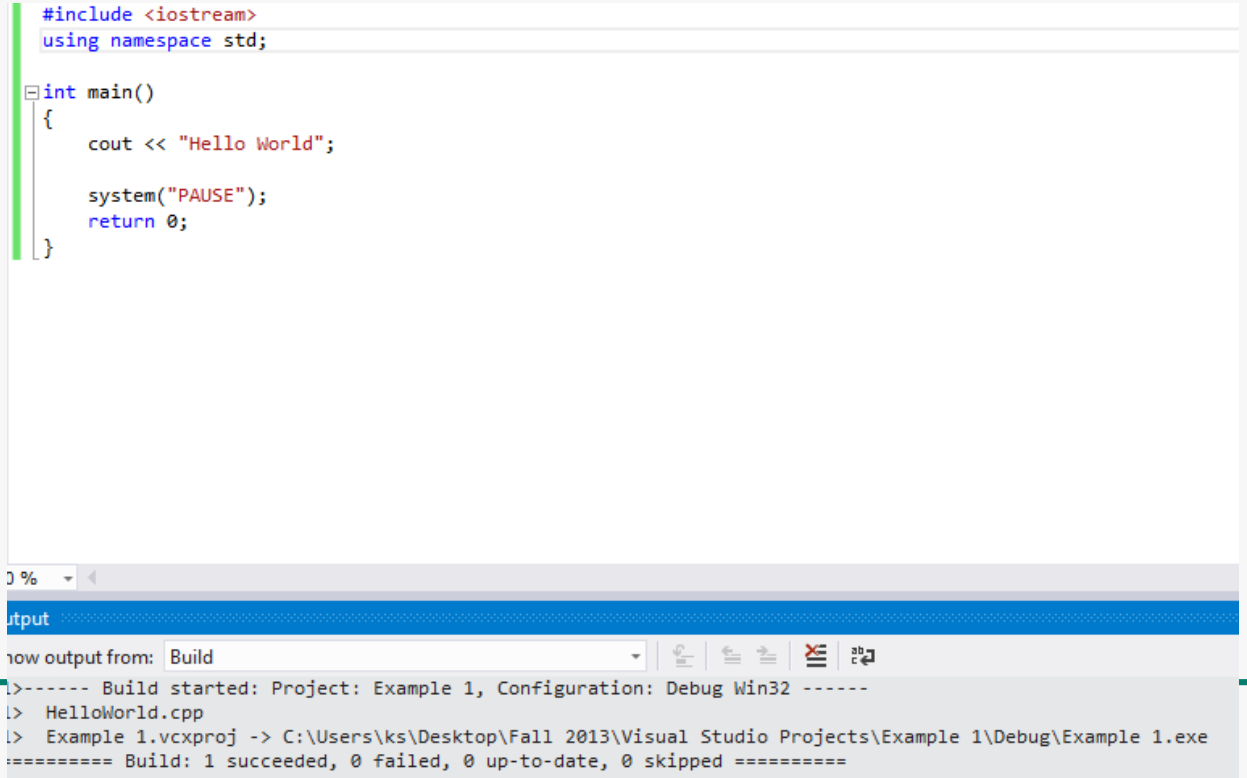
Running the Program using Visual C++

- **Using Microsoft Visual C++**
- **Step 3: Create a New Source File and Type in Your Code (continued)**
  - Next, enter the source code for HelloWorld.cpp exactly as I have in the previous slides
    - ***Type it in*** **rather than copy and paste so you get used to the structure of a C++ Program (watch for syntax errors!)**
    - **Make sure and save often**



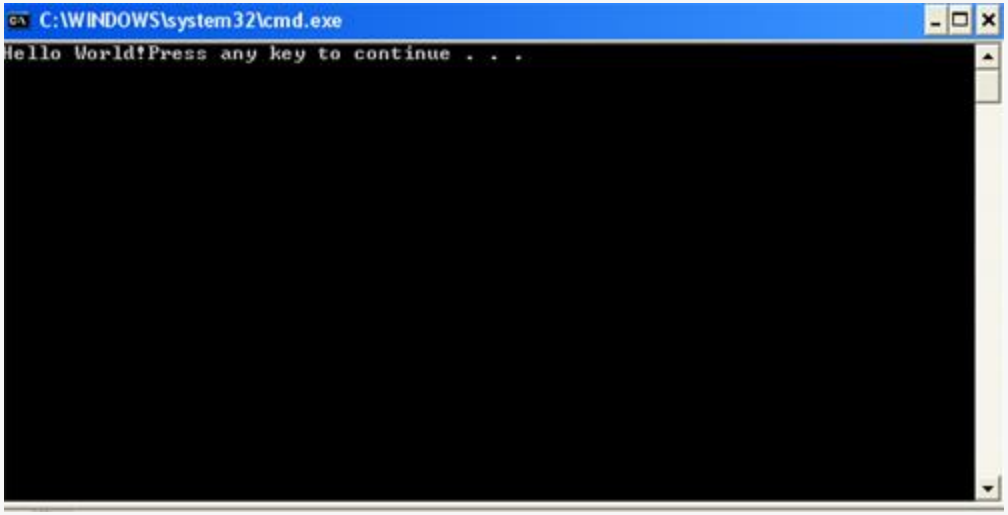
Running the Program using Visual C++

- **Using Microsoft Visual C++**
- **Step 4: Compile your source code**
  - To compile go to the Build drop down menu and select Build Solution
    - **Build->Build Solution**
    - Successful Compilation (no errors is shown below)
  - Any compilation errors are shown in the lower window
    - (try making an intentional error and recompiling your code)
  - NOTE: If you do have compilation errors double click on the line in the error window and Visual C++ will put the cursor on the line in the source code where it believes the problem exists
    - HINT: Often the error is on the line before the compiler gives
  - If you have the professional version View->Error list can help find errors (Better interface)



Running the Program using Visual C++

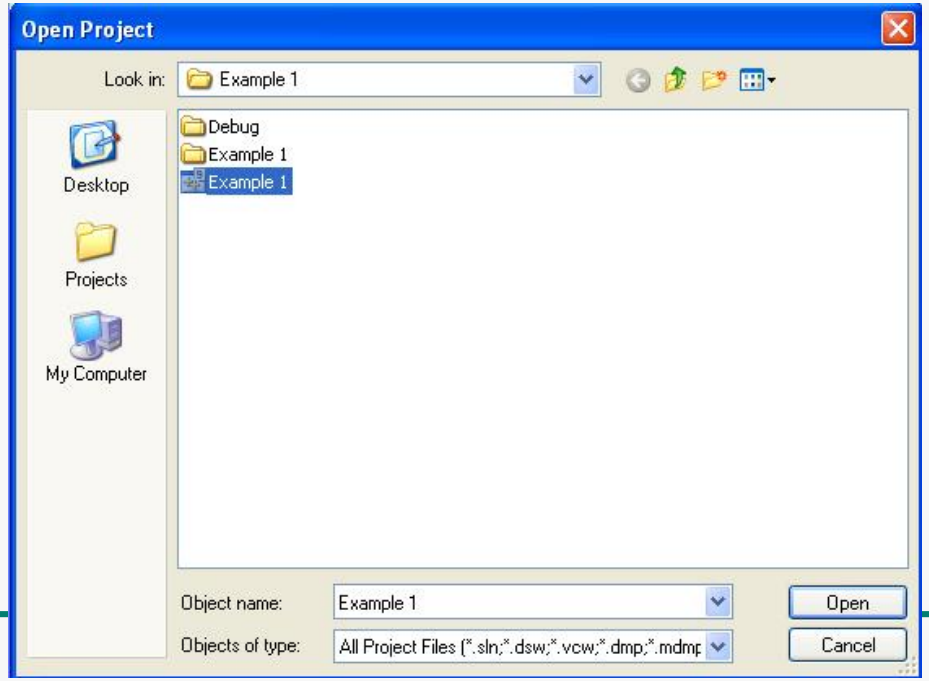
- **Using Microsoft Visual C++**
- **Step 5: Run Your Program**
  - To run your program go to the Debug drop down menu and select Start Without Debugging
    - **Debug->Start Without Debugging**
    - Successful Compilation (no errors is shown below)
  - Note the program displays this message at the end "Press any key to continue" – Press enter and close the window
    - We'll learn how to put this on a separate line than "Hello World" in a few slides



Running the Program using Visual C++

- **Using Microsoft Visual C++**
- **Step 6: Close your project**
  - Make sure and select **File->Save All** before you close your solution
  - Select **File->Close Solution**

- **How to Open a Saved Project**
  - Select **File->Open Project Solution**
    - Browse to the place where you save your project
    - Open the Solution File (.sln)
  - **DO NOT NAVIGATE TO THE C++ file and open that! Your project will appear to be open if you do this but it WILL NOT RUN**



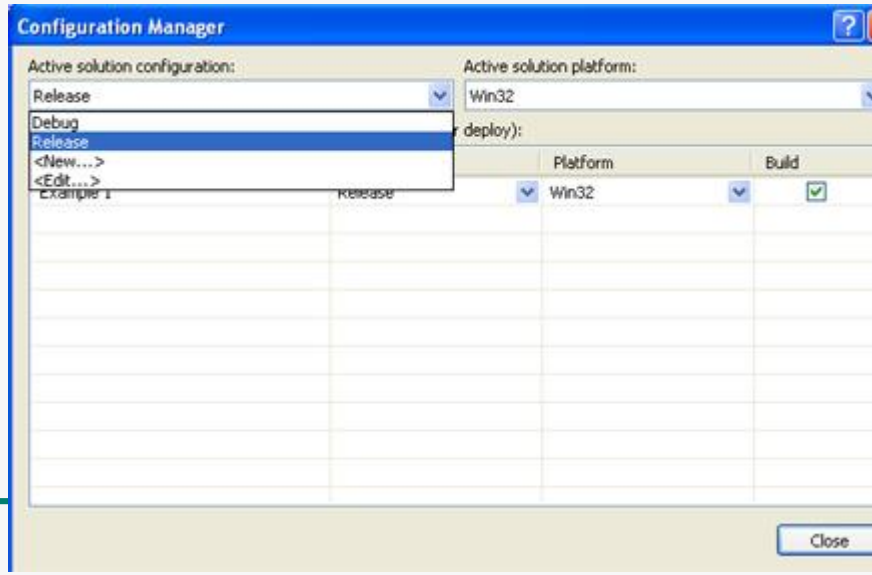
Running the Program using Visual C++

- **Using Microsoft Visual C++**
- **Check out all the Files Visual C++ has set up for you**
  - You have a Debug folder and another Folder that contains your Hello.cpp file as well as additional files
    - Don't worry about these additional files for now
  - Note we compiled using the Debug configuration which caused Visual C++ to create a lot of extra files (necessary hooks for the debugger which we don't yet know how to use)
    - This does make the program executable file fairly large

- Once your program is working well you may select the configuration of Release which compiles and builds your program without the debugging hooks
  - The projects we're doing are small so it's fine to just use the Debug configuration
  - Below are the instructions for release mode if you are interested
  - HOWEVER – MAKE SURE YOUR PROGRAM IS WORKING WELL FIRST
  - To do this:
    - Select: Build->Configuration Manager
    - Using the Drop down menu, change the selection from Debug to Release
    - Rebuild your solution:
      - Build->Rebuild
      - Now go look at the files generated

- In the instructor folder
  - I have many (not all, I should change them)

of my Visual C++ projects in Release mode



Running the Program using Visual C++

- **Important Tips**
  - **Warning 1:** It is a BAD IDEA to close Visual C++ by pressing the 'X' button in the corner of the Window. Save your workspace and then close it.
    - This ensures:
      - Visual C++ is shut down properly
      - ALL of your files and workspace are saved
      - Everything can be located when Visual C++ is restarted
  - **Warning 2:** When you start working on your project you must open the project/solution (.sln) file NOT the .cpp file.
    - DO NOT DOUBLE CLICK YOUR .cpp file to begin working on a program

Files to Submit For Homework

- **What to submit in Hypergrade when you turn in Programming Assignments and midterm/final Programming Projects**
  - As you can see the project files for Microsoft Visual C++ are large – fortunately the Hypergrade auto grader only needs the .cpp files when you submit homework assignments.
  - **This is what you should submit in Hypergrade from your project folder:** the `.cpp` and .h files only – I will show you how to find these in class
    - ***IMPORTANT NOTE:*** ***before submitting in Hypergrade you will also need to comment system("PAUSE") OUT of your code..Hypergrade is using yet a DIFFERENT C++ compiler***
    - We will do two Hypergrade Practice HW's IN CLASS next week, I like to show you hands on so you can follow along and do it in your Hypergrade account

- **Limitations on Moodle due to large file size**
  - **Unfortunately this means that on Moodle I will only be able to post the .cpp files for the examples rather than the entire project (I will try to post projects but most likely they will be too large)**
    - ***HOWEVER – most students find they really only need the .cpp and .h files, and if you really want the projects you can create them with the .cpp and .h files on Moodle at home and save them at home***

- We have done this for the past few semesters and it has worked out well
  - People enjoyed learning a real life compiler/IDE that is used in many companies

- **Next week as a class we will submit some easy practice assignments in Hypergrade (they will take less than 5 minutes each). It will be the easiest 10 pts of the entire class**
- ***Make sure you purchase Hypergrade before next week.***

Xcode Option

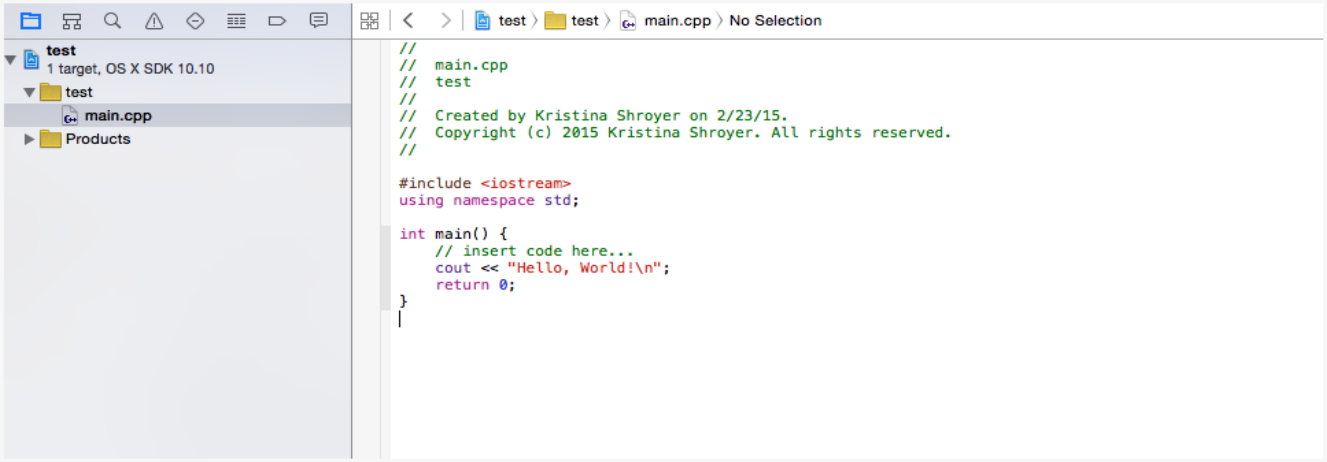
- For MAC Users – get Xcode for Class ASAP
  - Go to the App Store and download Xcode Developer Tools (it's a little blue icon with a hammer on it and is free)

Creating a Project - XCode

1. Open Xcode
2. File->New->Project
3. Select OSX->Application->Command Line Tool (VERY IMPORTANT)
4. Make sure language is C++
5. IMPORTANT – Do NOT use OL C++…add using namespace std and get rid of the stuff inside the parenthesis after main (see next slide for a screen shot)

X-Code Continued

- ***IMPORTANT!!*** This is what hello world and ALL projects should look like you MUST use namespace std; and you must clear out the inputs to main



Xcode Continued

- To Run your program do Command->R

More on cout

- Remember, **cout** is classified as a stream object which means it works with streams of data
  - To print a message on the screen you send a stream of characters to **cout**
  - The **<<** is called the stream-insertion operator. The information immediately to the right of the operator is sent to **cout** and displayed on the screen
  - Note that Strings in C++ are any combination of letters and special characters enclosed in double quotes Example: "string"
    - Strings in C++ are very similar to Strings in Java

- **Unless you specify otherwise, a C++ program displays output as a continuous stream of characters**
  - In the example below, all of the characters are printed on one line which may not be the result you are wanting
  - Solution: Newline Escape Sequence

- **Example (TopSellers1.cpp – Example 2)**

```
#include <iostream>
using namespace std;

int main()
{
   cout<<"The following items were topsellers";
   cout<<"for June:";
   cout<<"Computer Games";
   cout<<"Coffee";
   cout <<"Gum";

   system("PAUSE");
   return 0;
}
```

  - This program would print everything on one continuous line

- Hint: You can follow along with the programs I'm showing you in class, they are all in the instructor folder
  - Let's all open this one

More on cout

- Solution
- Example (TopSellers2.cpp – Example 3)

```
#include <iostream>
using namespace std;

int main()
{
 cout<<"The following items were topsellers ";
 cout<<"for June:\n";
 cout<<"Computer Games\n";
 cout<<"Coffee\n";
 cout <<"Gum\n\n";

 system("PAUSE");
 return 0;
}
```

- `\n` is an example of an ***escape sequence***
- Escape sequences are written as a backslash character followed by one or more control characters and **are used to control the way output is displayed**
  - The backslash signals to the compiler that the character following it should be interpreted with a special meaning

Newline Escape Sequence

- The newline escape sequence `\n` instructs the display device to move to a new line
  - A newline is caused when the a backslash `\` character followed by an `n` character are used together
  - ***Backslash*** **provides an "escape" from the normal interpretation of the character that follows**

- Newline escape sequences can be placed anywhere within a message to `cout` to force the display device (cursor) to move to the next new line

Other Common Escape Sequences

Escape Sequence	Name	Description
\n	Newline	Causes the cursor to go to the next line for subsequent printing
\t	Horizontal Tab	Causes the cursor to skip over to the next tab stop
\a	Alarm	Causes the computer to beep
\b	Backspace	Causes the cursor to back up, or move left one position
\r	Return	Causes the cursor to go to the beginning of the current line instead of to the next line
\\	Backslash	Causes a backslash to be printed
\’	Single Quote	Causes a single quotation mark to be printed
\”	Double Quote	Causes double quotation marks to be printed

Other Common Escape Sequences

- Example (EscapeSequences.cpp – Example 4)

```
//An example using Escape Sequences

#include <iostream>
using namespace std;

int main()
{
    cout << "\"This is how to print a direct quote.\"\n";
    cout << "The title of the book was \'Intro to C++\'.\n";
    cout << "The next line will use tabs to separate data with tabs.\n";
    cout << "apples\t oranges\t grapes\t \n\n";

   system("PAUSE");
   return 0;
}
```

More on cout - endl

- A second Solution
- Example (TopSellers3.cpp – Example 5)

```
#include <iostream>
using namespace std;

int main()
{
 cout<<"The following items were topsellers ";
 cout<<"for June:" << endl ;
 cout<<"Computer Games" << endl;
 cout<<"Coffee" << endl;
 cout <<"Gum" << endl;

 system("PAUSE");
 return 0;
}
```

- **endl** is a stream manipulator in the **iostream** file
- Every time **cout** encounters and **endl** stream manipulator it advances to the next line for printing
- The **endl** manipulator can be inserted anywhere in the stream of characters sent to **cout** outside the double quotes and after a **<<** symbol
  - Note since **endl** is a stream manipulator it must be placed after a **<<** symbol….it does not just go in the string like **\n** does

Syntax Errors

- ***Syntax*** is the set of rules for formulating grammatically correct C++ language statements (grammar rules of C++)
  - Compiler accepts statements with correct syntax without generating error message

- **A program statement can syntactically correct and logically incorrect**
  - Compiler will accept statement
  - Program will produce incorrect results
    - These type of errors are much harder to find than syntax errors and are called ***bugs***

- ***You will notice that C++ does not give as strict of compiler errors as Java does***
  - It is assumed by the C++ compiler that the programmer has done things for a reason
    - Therefore it is very important to watch errors that don't necessarily cause compiler errors but may instead cause your program not to run as expected

Programming Style

- Every C++ program must contain one and only one `main()` function
  - The program statements that make up the function included within the curly braces following the function header
    - `{ }`

- C++ allows flexibility in format for the word `main`, the parentheses `( )`, and braces `{ }`
  - More than one statement can be put on line
  - One statement can be written across lines

- However, use formatting for clarity and ease of program reading
  - Function name starts in column 1
    - The function header (name and parentheses) should be on its own single line
  - Opening curly brace of function body on next line
    - Aligned with first letter of function name
  - Closing curly brace is last line of function
    - Aligned with opening curly brace of the function
  - This standard form highlights the function as a unit

  - When possible try to put all statements associated with a particular command such as **`cout`** on a single line
  - The idea is to have a readable program organized by functions

Example of ***Bad Style*** that runs:

```
#include <iostream>
using namespace std;

int main(
){
cout<<"Hello World";
return 0;}
```

Programming Style

- Standard way to start a simple C++ Program
- This example uses good programming style

```
#include <iostream>
using namespace std;

int main()
{
     statements;
      system("PAUSE");
     return 0;
}
```

- Note: The editor in Microsoft Visual C++ (and many other IDEs) automatically indents your source code so that it is more readable. These indentation helps you line up the opening and closing curly braces for blocks or code making it easier to see things like the beginning and end of a function
  - Use this feature to make your programming style readable – readable code gets better grades

Programming Style - Whitespace

- Whitespace consists of

<space>
<tab>
<newline> (also called <enter> and <return>)

- Most whitespace is optional when writing C++ programs (it is ignored by the compiler)
  - However whitespace can be used to improve layout and readability of the code

- These examples are all valid syntax and produce the same result when executed

```
1.  cout << "hello";
2.  cout       <<"hello";
3.  cout
              <<          "hello"     ;
```

  - The first `cout` statement has the best programming style of the three examples
    - **It's easy to read and understand what the line of code is doing**

C++ Identifiers

- A C++ Program can contain more functions than just the main function
  - The additional functions are user defined functions and must be named
    - A user may name these functions with any name they like as long as it follows the rules to be a legal name in C++

- In addition, like in any other programming language, in a C++ program, variables must also be named

- ***C++ Identifiers*** are ***programmer defined names that represent some element of a program*** such as a variable or a user defined function (note main is always named main as it is special and identifies the starting point of a program)
  - So an identifier can be used to name a user defined function (such as the square function in the earlier example) or a user defined variable (next chapter)

C++ Identifiers

- You can choose any identifiers you want for your function and variable names as long as they are legal. To be legal they must adhere to the following rules:

1. The first character of the identifier must be a letter or an underscore _

2. After the first character any additional characters can only be letters, digits (0-9) or underscores

3. You can NOT use a keyword
   - ***C++ keywords*** are words that are set aside for a special purpose in the language and should only be used in the specified manner

- These rules are slightly different than the naming rules in Java

C++ Identifiers

- You should always choose names for variables and functions that give an indication of what the variable or function is used for
  - It is a good idea to try to use a mnemonic for a function, variable or class name so it will be easier to remember it
    - `degToRad` is a good function name for a function that converts degrees to radians because it gives someone reading the program an idea of what the function does as well as an easy way to remember what the function does
- Remember C++ is case sensitive
  - `DegToRad` would be a different identifier than `degToRad`
- There are a couple different standard ways that people like to name identifiers in C++
  - Some like to start with a lowercase letter and separate words with underscores
    - Ex. `deg_to_rad`
  - The standard in C++ is to start an identifier with a lower case letter and then capitalize the first letter of any additional words in the identifier
    - Ex. `degToRad`
  - Whatever way you choose to format your identifiers it is important that you format them all **consistently** throughout your program

C++ Keywords

TABLE 1.1 *C++ Keywords*

auto	default	goto	public	this
break	do	if	register	template
case	double	inline	return	typedef
catch	else	int	short	union
char	enum	long	signed	unsigned
class	extern	new	sizeof	virtual
const	float	overload	static	void
continue	for	private	struct	volatile
delete	friend	protected	switch	while

C++ Identifiers

- **Examples of valid identifiers**:

grossPay	taxCalc
addNums	degToRad
multByTwo	salesTax
netPay	

- **Examples of invalid identifiers**:

4ab3	(begins with a number)
e*6	(contains a special character)
while	(is a keyword)

Standard and Pre-standard C++

- Always keep in mind C\C++ has not always been standardized
- As a result there is a newer style (standardized) and older style (pre-standardized) of writing C++ code
  - We are using the standardized newer style in this class
- The differences between the older and newer styles are subtle but it is important to recognize them because some workplace's programming tools may only support the older conventions
  - Older style header files
    - In the older style C++ all header files end with the ".h" extension.
      - The iostream header in the older style would be written as:
        #include <iostream.h>
  - Older style use of namespace std;
    - Older style C++ programs typically do not use the using namespace std
      - Some older compilers do not support namespace at all and will produce and error message if you use it
- Some standard C++ compilers do not support programs written in the older style and the older pre-standard compilers do not support programs written in the newer style

- Our first program written in the older style

```
//A first C++ Program
#include <iostream.h>

void main(void)
{
  cout << "Hello World!";
}
```

- It's not important to know the older style exactly just to be able to recognize why a C++ program written in the older style may look different

- You will find one of the most frustrating things about working in C++ is the differences in compilers
  - If you learn C# or C++ programming with in .NET you will see a lot of these issues have been resolved

- ***DON'T FORGET IN THIS CLASS YOU ARE REQUIRED USE THE NEWER STYLE ANSI STANDARD C++, you will NOT get points for NON standard code from the 70s, 80s and 90s***

A Note on Comments

- Your comments should explain your thought process and logic. A comment like this:

```
#include <iostream>     //include the iostream library
```

does not tell the reader why the library is included

  - This comment is pointless, the reader of the code is a programmer that already knows what this is!

- Instead a comment like this tells the reader of the program ***why*** the library was included

```
#include <iostream>     //needed for cout statement
```

Tips for Programmers

- ***Compile Often***
  - Do not write your entire code and then try to compile, compile and find errors as you go
- ***Don't go to the computer start typing in code and hope you will be able to figure the program out as you go along***
  - This may work in the beginning
  - Can lead to less efficient algorithms and a lot of PAIN in this class
    - your first idea isn't always the best one
- ***Don't assume because your program compiles it is correct***
  - Test your program to see if it is doing what it's supposed to
    - A program can compile and not run correctly
- ***Don't avoid comments***
  - As programs become more complex, you could go back and look at a partially finished project and wonder what you were doing
- ***If you get frustrated leave the program alone for a while and come back to it***
  - A fresh perspective does wonders
- ***Don’t wait until the last minute!***
- ***Use good style from the beginning (don't put in the style later)***
  - This will just waste time in the line run
- ***Don't add random braces to try to get a program to compile!***

Common Programming Errors

- Omitting parentheses after **`main`**
- Omitting or incorrectly typing the opening brace **`{`**
  - Opening brace signifies start of function body
- Omitting or incorrectly typing the closing brace **`}`**
  - Closing brace signifies end of function
- Misspelling the name of an object or function
  - Example: Typing **`cot`** instead of **`cout`**
- Forgetting to close a string sent to **`cout`** with a double-quote symbol
- Omitting the semicolon at the end of each statement
- Forgetting **`\n`** or **`endl`** to indicate a new line

Summary

- C++ is an high level programming language
  - Some refer to it as an intermediate programming language because it is not as high level as Java or some other high level languages are
- C++ is an object oriented language with procedural aspects
  - We will begin by learning procedural programming and then build on that knowledge to learn to program in an object oriented manner
    - In this class we will focus much more on object oriented programming than we did in the Java class
    - We will quickly go through the concepts you have already learned in Java in C++ and then learn the more advanced concepts in C++
- Programs cannot be written until algorithms are selected and understood.
  - We will use pseudocode to understand algorithms before writing programs in this class
- Once a program is written in C++ it must be compiled and linked before it can be executed
- The simplest C++ program has the form – use this as a recipe for your programs – a standard starting point (until we get to objects in the second half of the class):

```
#include <iostream>
using namespaces std;
int main()
{
     program statements;
     return 0;
}
```

Summary (continued)

- C++ program statements are terminated by a semicolon
- The Standard library contains many functions and classes
  - Standard Library provided with C++ compiler
  - Includes **<iostream>** for input and output
- **cout** object displays text or numeric results
  - A stream of characters is sent to `cout` by:
    - Enclosing characters in double quotes
    - Using the insertion ("put to") operator, <<
- There are various ways of formatting program output generated with C++ (escape characters and endl)
- C++ identifiers are used to name various elements of a C++ Program such as user defined functions and variables
  - The rules for naming identifiers must be followed, and following the standards is a good idea to make your program more readable
- There are C++ formatting standards that should be followed to make your program more readable
- C++ is now a standardized language
  - There is an older style of C++ that you may run into in the workplace
  - Different C++ compilers will work differently and support different conventions