Introduction to C++

CS 1: Problem Solving & Program Design Using C++

Getting to Know Me and Me Getting to Know You

- First, a little about you
  - Your name
  - Have you ever worked with/used/played with any programming language? If so, talk about it
  - Why are you taking this course?
  - What are your expectations/goals from me, as well as this course?
- Then, a little about me
  - Background about myself
  - Cover the syllabus, including my expectations of you and this course
- Finally, any additional questions before we get started

Objectives

- Look at what a software developer needs for their skill set
- Define some rules for programmers
- Take a peek at C++ concepts and the background
- Understand the compiler and the compilation process
- Answer the question “Why program in C/C++?”
- Understand the basic rules for C++ statements
- Discover what it takes to do some basic output
- Learn how to setup and use the Microsoft Visual C++ Express IDE
- Check out what an identifier is and the rules for using one

Where Is C++ Used?

- C++ is widely used in today's programming environment
  - Game Programming
  - Controllers for Robots
  - User Interfaces
  - Graphics for Games
- Java is used more for mobile games and web development

Skill Set Needed for Software Developer

- Programming
- Design Skills
  - You must know/research the field you are designing the program for
  - Object-Oriented Design
- Documentation
  - Your software may be used for years to come
  - Someone should be able to read your code and easily understand it
- Communication
  - Team members, customers, end users
- Quick Learners

Rules for Programmers

- Keep your cool
- Work when you are rested
- Keep it simple
- Give help/Get help
  - On Homework – NOT on tests
  - What is help?
    - Ask me in lab, come to office hours or email
  - What is NOT help
    - Copying code
    - Copying code and only changing variable names or the order of certain statements
- Study and Know the Rules for the Language
  - Syntax

Rules for Programmers (2)

- Learn the Development Environment and Tools
  - Microsoft Visual C++
  - Microsoft Visual Studio .NET
  - X-Code
  - G++
- Understand the Problem You are Trying to Solve
- Build and test your software in steps
  - Compile often
- Save early/save often
- Practice and Study
  - Know both how to program and the programming concepts we discuss in lecture

C/C++ Brief History and Overview

- 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 versions 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 (2)

- 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++ Brief History and Overview (3)

- 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

C/C++ Brief History and Overview (4)

- 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

C/C++ Brief History and Overview (5)

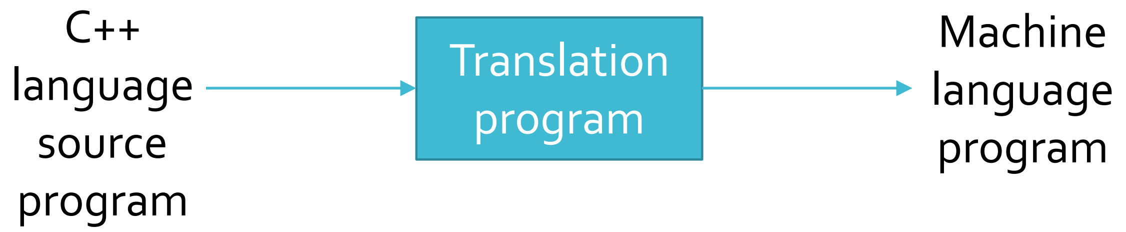
- Since the C++ language was an expansion of the C language:
  - When you learn C++, you are learning C as well
  - The entire C language is wrapped up in C++
    - if and for loops are essentially identical in C and C++
    - C has structures (structs), C++ has structures and classes (used with objects)

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

C++ Concepts/ Background (2)

- 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
- 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



Low-Level vs. High Level vs. Intermediate 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

Low-Level vs. High Level vs. Intermediate Level Languages (2)

- Low Level Languages (cont.)
  - 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 vs. High Level vs. Intermediate Level Languages (3)

- 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

Low-Level vs. High Level vs. Intermediate Level Languages (4)

- 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

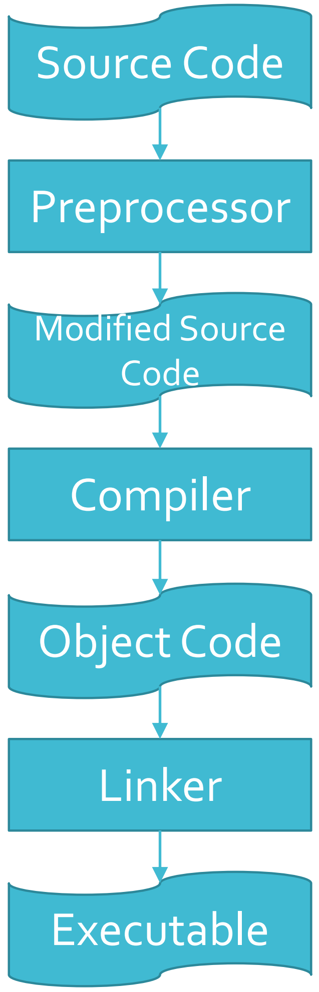
Low-Level vs. High Level vs. Intermediate Level Languages (5)

- 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)

Libraries

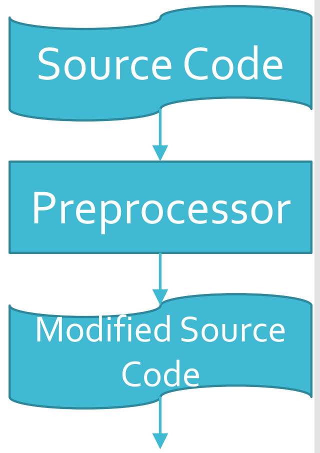
- C++ provides libraries containing classes and functions that a programmer can use in their programs
- One C++ library we will use in our very first program is a library that provides tools to receive input from the keyboard and write output to the monitor

C++ Code Processing



C++ Code Processing (2)

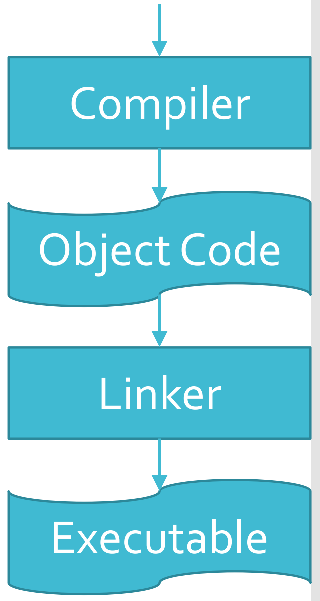
- Source Code
  - When a C++ program is written the first step is to type it onto a computer and save it into a file (.cpp)
  - After source code is saved into a file the translation process begins
- Preprocessor
  - The preprocessor searches in the source file for special lines of code that begin with 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++ Code Processing (3)

- Compiler
  - The compiler translates the source code into machine language
  - The Compiler creates an object code file that contains machine code for operations in source code file
- Object Code
  - The object code file only knows the location of functions defined in the source code file
  - You do NOT yet have an executable
- 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
- Executable
  - Once the linker is run and completed, the code is in executable form and can be run



General C++ Programming Process

- Use a text editor or IDE (integrated development environment) to enter the C++ source code
- Save the source code as a .cpp file
- 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

General C++ Programming Process (2)

- 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
- Run the program
  - The operating system loads your executable file into memory and passes control over to it
- Program runs

Portable Language

- 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

Portable Language (2)

- 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
  - 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

Portable Language (3)

- The programs we write in this class should be simple enough they will run easily on most C++ compilers
- However, new cross platform libraries are being 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 for building cross platform C++ programs
    - http://www.wxwidgets.org/

Procedural and 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

Procedural and Object-Oriented Programming (2)

- 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 Programming

- 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
  - 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

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 (2)

- Object oriented programming breaks down programming operations by object rather than by procedure
- 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

Object-Oriented Programming (3)

- 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 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

Object-Oriented Programming (3)

- 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 later in 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

Approach for This Course

- 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++
  - 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

Programming Fundamentals

- What is programming?
  - A program is a set of instructions a computer follows in order to perform a task
  - 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

Programming Fundamentals (2)

- 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

Programming Fundamentals (3)

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

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 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

Pseudocode (2)

- 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 data file one line at a time
  3. As each line of text is read in, search that line letter by letter for the letter 's'
  4. If the letter 's' is found, check the character before the 's' to make sure it is blank .
     a) If the character before the 's' is not blank
        1) if you have not reached the end of the line return to step 4.
        2) if you have reached the end of the line of data return to step 3 and read the next line of data
     b) 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
     c) If you find a match increase the sheepCount by 1 otherwise continue reading the line
  5. Repeat steps 3 and 4 for each line of code in the data file
  6. Display the number of time the word 'sheep' was found to the screen

Pseudocode (3)

- 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
     a) Notice there is no mention of HOW this will be implemented with code – Why? Pseudocode
     b) 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

Pseudocode (4)

- QUESTION: What do we call the previous series of statements?
  - Answer: PSEUDOCODE for an ALGORITHM
- Pseudocode is not rewriting your assignment 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

Algorithm and 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
    - Pseudocode 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

Pseudocode vs. 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

Pseudocode Example

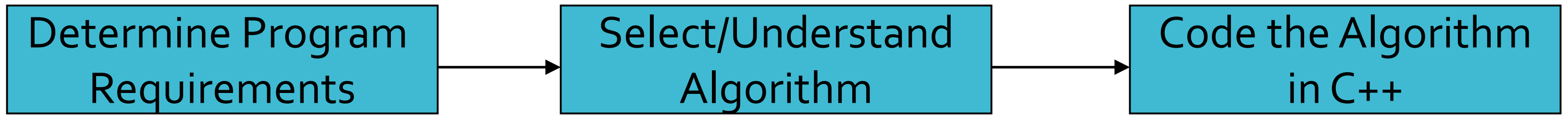
- PROBLEM: Calculate someone's Gross Pay
- High Level Pseudocode
  - Get Payroll Data
  - Calculate Gross Pay
  - Display Gross Pay
- Detailed Level Pseudocode
  - Display How Many Hours Did You Work?
  - Store user input in the hours variable
  - Display How Much Did You Get Paid Per Hour?
  - Store user input in the rate variable
  - Store the value of hours times rate in the pay variable
  - Display the value of the pay variable

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:
  - Solves the problem correctly
  - Solves the problem efficiently

Coding An Algorithm (2)

- 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, 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


Determine Program Requirements
Select/Understand Algorithm
Code the Algorithm in C++

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
  - 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
  - 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 and as we learn more will write programs that use many functions

C++ Program Structure (2)

- 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

C++ Program Structure (3)

- 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 (3)

- 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 first C++ Program
#include <iostream>
using namespace std;

int main()
{
  cout << "Hello World!";
  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

Comments (2)

- 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
  - Block comments
- 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
- 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 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 (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

The #include Directive – Header Files (2)

- 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 the code definition from the iostream library being used in this program for output to the monitor
- Libraries are very important, the code for input/output is complex

The #include Directive – Header Files (3)

- The #include directive is a Preprocessor Directive
- Peprocessor directives are not C++ statements, they are signals to the preprocessor that run prior to the compiler
  - They 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 Directive – Header Files (4)

- 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++ 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
  - 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)

Namespaces (2)

- 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)

Namespaces (3)

- 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!
- 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

Namespaces (4)

- So technically the Hello World program could have been written as follows:

```
//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
  - This is not OK for this class and is not ANSI Standard
- 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 (5)

- 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

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)

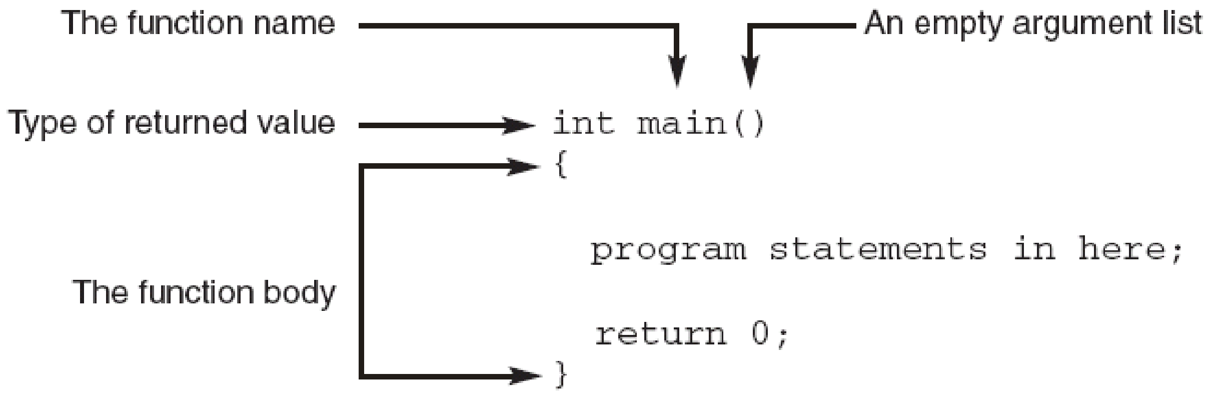
The main Function (2)

- The next line of the program is an open curly brace {
  - 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 first brace 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
  - 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

The main Function (3)

- 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*



```
int main()
{

  program statements in here;

  return 0;
}
```

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 create an instance of an object…this object 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 **C**onsole **Out**put 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

The cout Object (2)

- 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

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 }

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 o are separate statements

Whitespace (2)

- 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
  - `cout << "hello";`
  - `cout      <<"hello";`

  -
    ```
    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

Whitespace (3)

- 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()

C++ Basic Formatting Summary (2)

- 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

Integrated Development Environment (IDE)

- 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

Integrated Development Environment (IDE) (2)

- 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

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

More on cout (2)

- Example #1

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

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

   return 0;
}
```

  - This program would print everything on one continuous line

More on cout (3)

- Example #2

```
#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";

 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

```
//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";

    return 0;
}
```

endl

- Example

```
#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;

 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

Programming Style (2)

- 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

Programming Style (3)

Bad Style

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

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

Good Style

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

int main()
{
        statements;
        return 0;
}
```

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 (2)

- 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:
  - The first character of the identifier must be a letter or an underscore _
  - After the first character any additional characters can only be letters, digits (0-9) or underscores
  - You cannot 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 (3)

- 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

C++ Identifiers (4)

- 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++ Identifiers (4)

- 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)

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

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

Standard and Pre-Standard C++ (2)

- 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

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

Tips for Programmers (2)

- 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

Common Programming Errors (2)

- 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
- C++ is an object oriented language with procedural aspects
- Programs cannot be written until algorithms are selected and understood.
- 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

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

Summary (2)

- C++ program statements are terminated by a semicolon
- The Standard library contains many functions and classes
- cout object displays text or numeric results
- 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
- There are C++ formatting standards that should be followed to make your program more readable
- C++ is now a standardized language