Motivation

• One of the most evident problems that arises in novice programming is a lack of scalability.
  – This is often fine for initial learning – simplicity leaves much less room for confusion.
  – The more interesting question – why is the typical novice programming style not scalable?
Motivation

- Two key things to note in novice-style coding:
  - Note how we’re organizing data.
  - Note how we’re accessing data in the various functions of our proposed programs.
Motivation

• Two key things to note in novice-style coding:
  – Note how we’re organizing data.
  – How is the data grouped together?
  – Do these groupings help clarify things?
  – Are we limited to a fixed size/count of data?
  – Note how we’re accessing data...
Motivation

• Two key things to note in novice-style coding:
  – Note how we’re organizing data.
  – Note how we’re accessing data...
  – Do we have to copy-paste code to multiple points of our program, with slight modifications each time?
  – Do we have to assume all code copies operate perfectly for any of our code to work correctly?
Object Orientation

• The coding style of object-orientation provides one popular solution to these concerns.
  – Data are organized to represent distinct objects of the scenario being modeled.
  – The card deck
  – Each player’s hand
  – Each individual card
  – This is done by defining custom data types.
Object Orientation

• The coding style of object-orientation provides one popular solution to these concerns.
  – When these conceptual “objects” of the program are modeled as custom data types, we may then manipulate them through functions designed to operate upon those custom types.
  – CardHand[] CardDeck::dealHands
    – (int numHands, int numCards)
The coding style of object-orientation provides one popular solution to these concerns.

– Additionally, we may provide some functionality that will be seen as inherent to these custom data types.
– These allow accessing and manipulating attributes of our program’s objects.
– `void CardDeck::shuffle();`
Object Orientation

• We don’t think about it like this, but such functions already exist for our basic data types...
  – 1 + 1
  – 3.14159 * 2.71828
  – From Java:
    – “Hello ” + “World”
    – System.out.println(“The answer is ” + 42);
  – As written, these do not translate directly into C++.
  – In C++, cout << “The answer is ” << 42 << endl;
Object Orientation

• Programming then becomes about recognizing the distinct “objects” that need to exist within the system and coding them appropriately.
  – This includes needed interactions among objects.
C++ Code Structure

Cooperating with the Compiler
C / C++ Compilation

• In Java (and many other modern languages), the compiler is designed to make multiple passes over code files during compilation.
  – In doing this, the compiler first finds all objects, variables, and functions of interest that are available before beginning the actual computation.
In C++, you are required to manually “declare” any object, variable, or function within a code file before using it.

– Note: “declaring” vs “defining.”
– “declare” – “function X exists.”
– “define” – “this is what function X does.”
– If you try to use something before it’s declared, a compiler error will result.
C / C++ Compilation

• To simplify the process of declaring relevant code objects, C++ has two core file types.
  – Header files: “*.h”
  – Contains relevant declarations
  – Source files: “*.cpp” (“*.c” in C.)
  – Contains code definitions
  – Source and header files then include other header files with needed definitions.
C++ Resources

• Like Java, C++ has a substantial amount of pre-coded resources for use in programs.
  – This being said, Java’s built-in collection is far more extensive than C++’s.
  – These are also utilized by use of `#include`, as opposed to Java’s `import`.
  – However, built-in resources are included through `<angle brackets>` rather than “quotes.”
C++ Resources

• Very common imports:
  – `<string>`
    – Includes the std::string class, a C++ counterpart to Java’s string. This is not a fundamental type in C++.
  – `<iostream>`
    – Includes the std::cout and std::cin output and input streams.
    – As used in class, these are the console output and console input structures, like System.out and System.in from Java.