# Lesson #1: Introduction to C++

Welcome to C/C++ III

Course is 9 lessons

# Instructor Info

- Ray Mitchell III
  - ray@raymondmitchell.com
  - 10+ years software development
    - C/C++
    - Java
    - .NET

# Syllabus

- Course dates
- Description
- Prerequisites
- Course Materials
- Homework
- Grading
- Course Structure

# Lecture Notes Packet

- Provide code samples
  - Follow along in the lecture notes
- Notes version 1.0
- Slide title shows lecture notes section
  - Example, next slide title starts with “1.1”
  - Turn to lecture notes section 1.1

# 1.1 Introduction to C++

- ANSI-compliant C++
  - All homework must be ANSI-compliant
- Conventions used in lecture notes
  - Sample code
  - Program output

# 1.2 What This Course Covers

- Full coverage of C++
- Object-Oriented Programming
- Unified Modeling Language (UML)
  - Used throughout the course
- Overview of C++ Topics Covered

# 1.3 What You Should Already Know

- C Programming
  - Basic C Concepts
  - Pointers
  - Dynamic Memory Management
  - etc.
- No knowledge of C++ necessary

# 1.4 History of C++

- Created by Bjarne Stroustrup
  - 1979 at Bell Labs
- Enhancement to C language
  - Mostly a superset of C
- Standardized in 1998 (C++98)
- Philosophy
  - General purpose, as efficient as C
  - Support object-oriented programming

# 1.5 Standard C++

- 3 Parts to Standard C++
  - Core Language
  - Preprocessor Support
  - Standard Library

# 1.6 How C++ Extends C

- Object Oriented Programming
- Generic Programming
- Function and Operator Overloading
- Namespaces
- Runtime Type Identification

# 1.7 C++ Standard Library

- Includes C Standard Library
- String data type
  - Safer than C-style null-terimated strings
- Streams
  - Safer than C-style I/O
- Containers, iterators, algorithms
  - Safer & more powerful than arrays
- Improved dynamic memory management

# 1.8 Future of C++

- C++0x
  - Goal to release in 200X
  - Likely released in 201X
- New features
  - Extensions to core language
  - Extensions to standard library

# 1.9 Useful Resources

- C++ Standard Documentation
  - http://www.cplusplus.com
  - http://www.cppreference.com/wiki
- Bjarne Stroustrup's homepage
  - http://www.research.att.com/~bs/homepage.html
- Extensions to Standard Library
  - http://www.boost.org

# 1.10 Typical C++ Development Environment

- Text Editor
- Preprocessor
- Compiler
- Linker
- Loader
- Running Program

# 1.11 Hello, world!

- main function
- std::cout
  - std is namespace
  - :: is scope resolution operator
  - cout is stream object allowing output to STDOUT
- Stream insertion operator (<<)
  - Outputs value on right to stream on left
- File extension ".cpp"

# 1.12 Basic I/O

- std::cin
  - Stream object allowing input from STDIN
- std::cerr
  - Stream object allowing output to STDERR
- I/O can be chained
- I/O can operate on multiple types of objects

# 1.13 Why are Namespaces Necessary?

- Names collide
- Can't control names in 3rd party libraries
- Namespaces
  - Allow names to be used without risk of conflict

# 1.14 User-Defined Namespaces

- Namespaces create scopes
  - Names must be qualified with scope name
  - Separate name
- Namespace naming conventions
  - Pascal, camel, all lower
- Multiple include guard includes namespace
- Namespace “std”
  - Holds entire standard library

# 1.15 Using the C Standard Library in C++

- C Standard Library Available in C++
- Two versions of C Standard Library
  - New style
    - #include <cstdio>
    - Names in namespace “std”
  - Old style
    - #include <stdio.h>
    - Names in global namespace

# 1.16 Ways to Access Names in a Namespace

- Scope Resolution Operator ::
  - Left operand is namespace
  - Right operand is name
- Using Declaration
  - Imports single name into current namespace
- Using Directive
  - Imports all names into current namespace
  - Avoid using directives

# 1.17 Nested Namespaces

- Allow full control for modularizing code
- Convention
  - Outer namespaces general
  - Inner namespaces specific
  - Example:
    - Microsoft::Office::Word::SpellChecker

# 1.18 Namespace Aliases

- Namespaces can get long
- Alias allows shorthand
- Beware
  - Can make code harder to understand
  - Only use when makes code clearer

# 1.19 Programming Paradigms

- Paradigm enables certain thought process
  - Using right paradigm makes problems easier to solve
- Paradigms supported by C++
  - Procedural (same as C)
  - Object Oriented
  - Generic

# 1.20 Procedural Programming

- Supported by C (and C++)
- Focus is on actions
  - Procedures (functions), verbs
- Program viewed as series of function calls
- Data is secondary (parameters)
- Breaks down for large projects

# 1.21 Object Oriented Programming

- Supported by C++
- Focus is on the objects (data)
  - The data being manipulated by program
- Program viewed as set of object interacting with each other
- Actions secondary (belong to data)
- Scales well for large projects

# 1.21 Unified Modeling Language (UML)

- Useful for modeling objects and interactions in object oriented programming
- Independent of programming language (e.g. C++)
- Convert from UML to C++ (or other language)
- Class Diagrams
  - Model for classes of objects
- Absence of info in UML diagram does not mean info not present

# 1.23 Mapping a UML Class Diagram to C++

- C++ class definition
  - Keyword “class”
- Access specifiers
  - Keyword “public”
- Data members
- Member functions
- Objects created from a class

# 1.24 Adding Data Constraints

- Restricting values allowed for data member
- Public members can be changed
  - Need way to restrict access to data members

# 1.25 The “private” Access Specifier

- Object Oriented Principle: Abstraction
  - Also called “Information Hiding”
- Keyword “private”
  - Prevents access to members
- Convention: public before private
- Best practice:
  - All data members private

# 1.26 Accessors and Mutators

- Make all data members private
- Accessors and Mutators
  - Allow controlled access to data members
  - Allow adding data constraints, logging, etc.

# 1.27 Default Constructor

- Storage class determines initial values of data members
- Constructor
  - Called when new object created
  - Used to provide initial data member values
- Default Constructor
  - Constructor taking no parameters
  - Provided by compiler if you provide no constructors

# 1.28 Constructors with Parameters

- Constructors can take parameters
- Parameters allow custom initialization
- Remember
  - If you provide any constructor, compiler will not provide free default constructor for you

# 1.29 The “this” Pointer

- “this” points to object on which member function was invoked
- Useful when parameter names hide member names
- Best practice:
  - Prefer using “this” to access hidden members rather than avoiding hiding members by using clever parameter names

# 1.30 Destructor

- Opposite of constructors
- Used to perform cleanup when object destroyed
- Provided by compiler if you don't provide a destructor

# 1.31 C++ Classes vs. Structs

- structs have same capabilities as classes
  - Data members
  - Member functions
- Only difference:
  - struct members public by default
  - class members private by default
- Choice between classes & structs purely style

# End of Lesson 1