# Day 1

You need to learn the basic mechanics of functions and how to refactor a problem into reusable subroutines.

* Main is first
* Code flows from top to bottom
* Calling other functions
* Return stack
* Basic syntax
* Refactoring functions

You need to learn the basics of the stack and how variables (global and local) are allocated kept and passed between functions.

* Locals are allocated and destroyed with the stack
* Copies are passed to functions
* Functions do not modify the caller
* Global variables are outside all functions

You need to understand expressions and function-return substitution. You need to know the basic built-in (primitive) types and operators.

* The sizes of the primitives depend on the platform
* Use parenthesis to build up a mathematical expression
* A function can be substituted in place of a value
* There are many built-in operators

You need to understand basic code flow constructs.

* If/else/do/while
* Break/continue
* For/switch

The “Hi/Lo” uses basic functions, expressions, and code-flow.

# Day 2

You need to understand “include” files and basic pre-processor directives.

* Text substitution
* Function prototypes
* Header files
* Libraries and the build process

You need to understand namespaces to avoid name collisions. The standard library and other code you use will require it.

* Using (never in a header)
* The :: scope operator

You need to understand pointers.

* Lots of memory diagrams
* Holds address in storage … compiler told how about size
* Pass pointers to modify values
* Pointer math and “index” math

You need to understand arrays.

* The name “points” to first value
* Initialization
* Size not kept at runtime
* Passing arrays to functions
* Arrays and Pointers are brothers
* Multi-dimensional arrays

You need to understand the basics of algorithms and performance timing.

* Basic looping through arrays
* Sorting algorithm
* Loop for timing

You need to understand C++ references.

* Syntactic sugar on pointers
* Rules: no null, must be init-ed, no math, can’t change
* Get used to compiler hiding a lot from you
* “const” keyword

TicTacToe uses an array and loops. Organize into files and namespaces.

# Day 3

You need to understand how to combine primitives into your own structures and access the pieces.

* The “dot” operator and multiple levels deep
* Passing on the stack
* Passing pointers to and the “->” for “dot”

You need to understand the mechanics of a “method”.

* Automatic “this” pointer
* Permission (“class” instead of “structure”)

You need to understand constructors and destructors.

* When they are called

You need to understand inheritance.

* Extending by adding new stuff
* Can’t take anything away
* Overriding methods

TicTacToe as an Object

# Day 4

You need to understand operator overloading and function overloading.

* Operators in a class (== and =)
* Global functions (<< and >>)

You need to understand the String class

* Basic substitutions and searches
* Overview of all the methods

You need to understand the basics of iostreams and File I/O.

* Opening a file
* Errors and exceptions (later)
* Reading and writing text (primitive types)
* Reading and writing binary
* Stream functions

MadLibs

# Day 5

Templates and STL Containers

Exceptions

Lab?

# Day 6

You need to understand dynamic memory management.

* New and delete
* Array new and array delete

Casting up and Down the Heirarchy

You need to understand virtual functions.

* Runtime information in each object
* User doesn’t know exactly what it is

Implementation, Interfaces, and Design Patterns, abstract, decorator

In-class lab