EECS 280 Learning Objectives

1. **Memory Models/Function Calls and the Stack:**
    a. Understanding automatic/local/dynamic storage (identify code sequences)
    b. Being able to correctly draw a memory diagram
    c. Define the difference between a variable and an object
    d. Be able to define the difference between compile time and runtime

2. **Testing and Procedural Abstraction**
    a. Explain the purpose of procedural abstraction and why it is valuable
    b. Design/develop ~thorough~ test cases
    c. Identify bugs in code and identify how to fix the bug
    d. Translate problems into code (and vice versa)/ Explain what a given code snippet does

3. **Pointers**
    a. Understand that the value of a pointer is a memory address
    b. Use pointers to access elements within an array
    c. Understand difference between traversal by pointer and traversal by index
    d. Recognize and correct common pointer errors
    e. Explain actions that can cause undefined behavior or runtime error (not initializing pointer)

4. **Arrays**
    a. Be able to explain similarities and differences of traversal by index and traversal by pointers
        i. understand how arrays look in memory
        ii. Use traversal by index
        iii. Use traversal by pointer when working with arrays
    b. Be able to explain array decay
    c. Detect when array decay happens in a block of code
    d. Understand that arrays are not class-type objects and they are basically pointers

5. **Compound Objects and ADTs**
    a. Students should be able to define and create an ADT (e.g. define constructors properly, etc.)
        i. Translate a problem into ADT
        ii. ~Understand~ motivation when using ADTs
        iii. More examples where students build definition of an ADT from scratch
    b. Identify when interface has been broken
    c. Be able to use the scope resolution operator properly
    d. Understand how to access members of a class type object using the dot operator
    e. Understand how to access members of a pointer to a class type object using the arrow operator

6. **Strings**
    a. Apply array operations to c-strings (indexing, dereferencing, pointer arithmetic)
    b. Identify what aspects of a c-strings make it different from a *regular* array
        i. C-strings are treated as a whole when printed as opposed to arrays printing addresses
        ii. Null character is not part of length of c-string but included as an element of the underlying array
    c. Understand the null terminating character
        i. Must be at the end of all c-strings for them to be valid
        ii. Causes c-strings to stop printing to cout
    d. Develop code making use of cstring library functions to perform operations on C-Style strings (including strlen, strcmp, etc.)
    e. Explain the difference between C-Style and C++-Style strings

7. **Streams, and I/O**
    a. Develop programs that make use of command line arguments
    b. Explain how command line arguments relate to the argc and argv parameters of main
    c. Be able to use input and output streams to access data in files
        i. Be able to check if the input/output stream was opened successfully

8. **Const**
    a. Explain why const is used and how it modifies interactions with a variable
    b. Identify conversions between const and non-const types that would cause compile errors
    c. Distinguish between a const pointer and a pointer-to-const

9. **Inheritance**
    a. Understand “is-a” relationship between a data type that inherits from another data type
    b. Identify “is-a” relationships when given two or more data types
    c. Be able to differentiate between “is-a” (inheritance) and “has-a” (composition) relationships
    d. Understand access modifiers (public, private, protected)
    e. Understand the order in which base class constructors and destructors are called in derived types

10. **Polymorphism**
    a. Understand name lookup process
    b. Identify implicit downcasts and understand why they are prohibited
    c. Be able to identify overloaded functions
    d. Be able to identify overridden functions

e. Understand what characterizes an overloaded function and construct one given a description of what it should do
f. Understand what characterizes an overridden function and construct one given a description of what it should do
g. Understand why we can construct a base class pointer and assign it to the address of a derived class object
h. Understand why we use abstract classes and interfaces and how to use them
    i. Be able to identify an abstract class
    ii. Be able to identify an interface
    iii. Identify pure virtual functions and understand that they make the class abstract
    iv. Be able to write a pure virtual function
    v. Be able to explain that abstract classes cannot be instantiated, but that abstract class pointers can exist
i. Be able to use the ‘virtual’ keyword properly and explain what the ‘virtual’ keyword achieves
j. Be able to use the ‘override’ keyword properly and explain what the ‘override’ keyword achieves
k. Understand the name lookup process for dynamic binding
l. Know what conditions must be necessary for dynamic binding to work and how to implement them
m. Understand why we use factory pattern and why it works