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

Final Exam Material

11. Dynamic Memory

a. Understand the difference between the stack and the heap
b. What the stack and heap look like in memory (memory model)
c. Understand how the scope of dynamic memory variables can be controlled
d. Identify use of dynamic memory in code
e. Understand that the 'new' operator allocates space in the heap **and** evaluates to its address
f. Know when the delete operator should be used and how to use it
g. Create and use dynamic arrays
    i. Be able to use the specific delete[] operator when deleting dynamic arrays
    ii. Be able to identify when it is needed to iterate through a dynamic array to delete elements in the array
h. Follow the scope of an object and identify where it is created/destroyed (ex. writing ctor and dtor print statements in the correct order)
i. Identify and correct dynamic memory errors
    i. Double free, memory leak, etc.

j. Develop code that uses dynamic memory and prevents memory leaks and other errors
k. Draw memory diagrams with dynamic memory (drawing the heap)
l. Be able to explain RAII, and why using RAII is beneficial
m. Develop code that makes use of RAII

12. The Big Three

a. Be able to identify classes which need custom implementations of the big three
b. Be able to correctly implement the big three if needed
c. Be able to explain the difference between shallow and deep copies
d. Be able to explain why destructors should be marked as virtual in classes that make use of inheritance
e. Identify errors in big three implementations or shallow copies and how they could cause dynamic memory errors later on
f. Understand what the default, compiler generated versions of the big three do

13. Containers and Templates

a. Be able to identify time complexities of insert, find and remove for both Sorted and Unsorted Sets
b. Be able to explain an advantage of using templates when defining the interface for a container
c. Be able to use templates properly
d. Be able to identify various types of containers and each underlying functionality (i.e. some containers have member functions like push_back())
e. Be able to use various types of containers when given a problem
f. Be able to explain why operator overloading is necessary
g. Be able to write an implementation of an operator overload

14. Recursion

a. Be able to identify recursion, tail recursion, tree recursion, and structural recursion problems
b. Be able to write a recursive function given a problem
    i. Break down problem into smaller sub-problems
c. Understand how to write a structural recursive function
    i. Break down data structures into smaller data structures
d. Be able to write an appropriate base case given a problem
e. Be able to differentiate between the memory complexity of normal linear recursive functions and tail recursive functions
f. Be able to draw a memory diagram of the stack given a recursive function call
g. Convert iteration into recursion and recursion into iteration
h. Understand when to use recursion vs. iteration

15. BSTs

a. Be able to identify the difference in time complexities between removing, inserting, and finding an item in a BST vs an Unsorted Set and a Sorted Set
    i. Best case, worst case, and average case
b. Be able to draw a BST representation in memory with pointers
c. Be able to use recursion to implement functions such as insert, find, and height for a BST
d. Be able to identify whether the big three is needed for a BST (and why) and properly implement the big three using recursion in a BST interface
e. Be able to write preorder and inorder traversals of BSTs
f. Be able to identify and explain the stack frame creation/destruction when using recursive functions in a BST

16. Iterators and Functors

a. Be able to traverse through a container with an iterator
b. Be able to identify scenarios in which an iterator can be invalidated
c. Be able to explain the concept of an end iterator
d. Be able to define what a functor is
e. Be able to identify at least one advantage in using functors
f. Be able to use a functor in code
g. Be able to define what a predicate is
h. Be able to define what a comparator is
i. Be able to write a predicate
j. Be able to write a comparator
k. Be able to identify that range based for loops are effectively the same as traversal by iterator
l. Iterators are meant to provide a standardized interface for traversing through different types of containers
    i. Know when to dereference an iterator
m. Know how to iterate through a container using iterators
n. Understand why iterators should not implement the big three, even though they reference to dynamic memory
o. Be able to define what a function pointer is
p. Be able to use a function pointer in code

17. Linked Lists

a. Be able to identify at least two differences between using a linked list and a vector
b. Be able to draw a linked list representation in memory
c. Be able to declare a linked list iterator
d. Be able to use a linked list iterator
e. Be able to define friend classes
f. Be able to use friend classes

g. Be able to add and delete nodes into the linked list without creating any memory errors
h. Perform operations on linked lists without errors (chopping a list, stretching)
i. Explain how linked lists are recursive data structures
j. Explain why iterators are necessary for traversing a linked list
k. Explain why linked lists are better (have faster time complexity) than arrays/vectors for certain operations (removing from the top, middle)

~~18. Error Handling and Exceptions~~

~~a. Be able to explain how throwing an exception affects the flow of a program~~
~~b. Be able to develop code that accounts for exceptions through the use of try/catch blocks~~
~~c. Understand and apply inheritance to catching exceptions~~

19. Impostor Syndrome

a. Be able to identify common characteristics of impostor syndrome
b. Be able to identify steps that can be taken to overcome impostor syndrome
c. Understand that most individuals feel imposter syndrome at some points in their careers
d. Understand that certain groups of individuals are more prone to feeling imposter syndrome through their careers