## CSCI 262 Final Exam Review

### What to study – Overview

#### Basic C++ Concepts
- Functions
- Function overloading
- Value & reference parameters
- Memory model (function call stack)
- Classes & objects
- Syntax (scope resolution op, dot op)
- Member variables & functions
- Constructors
- Visibility
- Basic I/O
- Pointers & arrays

#### ADTs/Data structures:
- Vectors
- Stacks
- Queues
- Sets
- Maps
- Linked Lists
- Binary Trees/Binary Search Trees
- Hash Tables
- Graphs

#### Advanced C++ Concepts:
- Dynamic memory allocation
- Operator overloading
- The "big 3"
- Templates
- Inheritance

#### Other Important Topics:
- Analysis of algorithms ("big O")
- Recursion
- Libraries & interfaces
Basic C++ Concepts

- **Functions**
  - How to write them, how to use them
  - What does pass by value mean?
  - What does pass by reference mean?
- **Function Overloading**
  - Can two functions have the same name if they have different parameters?
- **Memory model (function call stack)**
- **Classes and objects**
  - Know how to write a class, how to implement class methods using the scope resolution operator (::)
  - Know what making class members and methods public/private means
- **Basic I/O**
  - Know how to use cout/cin

- **Syntax**
  - Know what the scope resolution operator is for (::)
  - Know what the dot operator is for
  - Know what the arrow operator is for (->)
- **Member variables and functions**
  - What’s it mean for a variable/function to be a member of a class?
  - What’s different about the syntax when implementing a member function vs. a normal function
- **Constructors**
  - Know how to write a constructor (i.e. what the syntax looks like)
  - What is a constructor for? When does it run?
- **Visibility**
  - Know what making class members and methods public/private means

Other Basic C++

- **Pointers and Arrays**
  - What is a pointer?
  - How are pointers and arrays similar?
  - What does the ampersand do? (&)
  - What happens when you add values to a pointer? (Understand pointer arithmetic)
  - Do arrays have a .size() function?
  - What does "dereference" mean? How do you dereference a pointer?
  - Can you use the dot operator on a pointer to an object to access the object’s members?
  - What the heck does this arrow thing do? ->
More advanced C++

→ Dynamic memory allocation
  ◆ What does declaring something on the stack vs the heap mean?
  ◆ How do we declare something on the stack?
  ◆ How do we declare it on the heap?
  ◆ How do we free heap memory from the system (i.e. “delete” a variable)? Do we need to do this for stack variables?
  ◆ Can a function return an array that was declared on the stack?

→ Operator overloading
  ◆ How do we overload an operator as a free function? What about as a class method?
  ◆ Which operator overload(s) must be a free function? What about a class method?
  ◆ What’s a friend function? Why do we need to make some of our operator overloads friend functions?

→ The “big 3”
  ◆ What are the big 3?
  ◆ How do we override each of the big 3 for a custom class?
  ◆ When is the copy constructor called? When is the assignment operator called? Which one is called here: complex a = b;
  ◆ Why are the copy constructor and the assignment operator important even though C++ offers us default behaviors for both?
  ◆ What does deep copy mean and how is it different from shallow copy? What is the default behavior if we don’t override the defaults?
  ◆ When is the destructor called? Why is the destructor important?

More advanced C++ (Continued)

→ Templates
  ◆ What does generic programming mean?
  ◆ How do templates make our lives easier as C++ programmers?
  ◆ How do we use a template in a function? What about a custom class? What does the syntax look like for each?

→ Inheritance
  ◆ What does it mean for a class to inherit from another class?
  ◆ What is the syntax for inheritance in C++?
  ◆ What does a child class inherit from a parent class?
  ◆ What does polymorphism mean?
  ◆ Can a child class override the functions of its parent? How is this done?
  ◆ What does the virtual keyword do? Does the virtual keyword matter when we’re not using pointers?
  ◆ What is a pure virtual function? What is an abstract class? How are these related?
Abstract Data Types/Data Structures

- Just like with previous exams, a page specifying the methods of each data structure in C++ will be provided for you, so don’t spend time memorizing names of functions!

- Vectors
  - Make sure you know the syntax differences between declaring an array of ints vs a vector or ints, for example
  - Know the algorithmic complexities in Big O of adds/inserts/deletes and how they compare to a linked list

- Stacks
  - Is a stack FIFO or LIFO?
  - Does pop() return the value on the top, or just remove it?
  - Can you iterate over a stack using a range based for loop?

- Queues
  - Is a queue FIFO or LIFO?
  - Does pop() return the value at the front, or just remove it?
  - Can you iterate over a queue using a range based for loop?

Abstract Data Types/Data Structures (Continued)

- Sets
  - What happens if you add the same element to a set twice?
  - In a range based for loop, what order are elements of a set accessed in, assuming we’re using a standard set?
  - How do you check if an element is contained within a set?
  - Can we use square brackets to access elements of a set?

- Maps
  - What happens when you use square brackets and the assignment operator to assign a value to a key that’s already in the map? (e.g. my_map['a'] = ?)
  - What happens when you use the methods insert or emplace with a key that’s already in the map? How is this different from using square brackets?
Maps (Continued)

- What happens when you try to access an element that isn’t in the map using square brackets? (e.g. `my_map[ 'a' ]++` when 'a' wasn’t a key in the map)
- How can you iterate over a map using iterators and range based for loops?
  - Where does the pair class help?
  - Where can the keyword 'auto' help?
- In a range based for loop, what order are elements of a map accessed in, assuming we’re using a standard map?

Linked lists

- Be comfortable with the basic linked list node class with "data" and "next" members.
- What makes a linked list different from a vector? How is each stored in memory?
- Why would we use a linked list over a vector and vice versa?
- How do we add/insert/erase nodes from a linked list? What is the big O complexity of each of these?
- How do we know that we’ve reached the end of a linked list?
- If our application needs to insert/delete from our list often, should we use a linked list of a vector? What if we need to access elements by index often?

Binary Trees

- What makes a tree a binary tree?
- What is the depth of a tree node? What is meant by the height of a tree?
- What are the minimum and maximum heights of a binary tree with n nodes?
- Given a tree, can you write the order that the tree’s nodes are traversed in a pre-order, in-order, and post-order traversal?

Binary Search Trees

- How does searching within a binary search tree work?
- How do you insert a new node into a BST?
- How do you remove a node from a BST? (3 cases)
- What are the Big O complexities of searching, inserting, and removing a node from a BST?
- What does it mean for a tree to be height balanced? Why is it important for a BST to be height balanced?
- Which data structures use BSTs in C++?
Abstract Data Types/Data Structures (Continued Again)

→ Hash Tables
  ◆ How do hash tables store data? How are arrays and linked lists involved?
  ◆ What is a collision?
  ◆ What is a hash function? What makes a good hash function? How is the modulus operator involved?
  ◆ What are the algorithmic complexities in Big O of finds, inserts, and erases with hash tables? (Assuming a good hash function)
  ◆ What can go wrong if we have a bad hash function?
  ◆ Which data structures use hash tables in C++?

→ Graphs
  ◆ What’s a vertex/node? What’s an edge?
  ◆ What’s the difference between a directed graph and an undirected graph?
  ◆ What makes a graph simple?
  ◆ What makes a graph complete?
  ◆ What are two ways to store a graph in a computer?
  ◆ Know the basic difference(s) between depth first search and breadth first search

Other Important Stuff

→ Analysis of algorithms ("big O")
  ◆ How does big O measure efficiency? Why does it not care about lower order terms and constants?
  ◆ Be able to read an algorithm and judge its complexity (including recursive algorithms)
  ◆ Ordering of dominance relations:
    \[(n!) > 3^n > 2^n > n^2 > n \log n > n > \log n > 1\]
  ◆ Know how to simplify, e.g. \(O(n^3 + n! + 5) = O(n!\)"

→ Recursion
  ◆ What does recursion mean?
  ◆ What is the base case? Why is it important?
  ◆ Why is it important to reduce the problem with every recursive step?
  ◆ Be able to judge the algorithmic complexity of a recursive algorithm (big O)

→ What does "compile" mean? What happens when you compile?

→ Libraries & interfaces
  ◆ Know that an "interface" is the user facing part of a library, and that a library is a collection of files/classes made by someone and packaged together for re-use.