Exam 2 Review Guide

# Exam format

* 75 minute test
* The exam is comprehensive by nature (you still need to know all of the C++ concepts already covered), but the main focus will be on topics covered since Exam 1.
  + Any material covered in lecture, assigned readings, and homeworks could be tested.
* 4 problems, similar to homework problems. Possible problems include:
  + One on C++ programming.
  + One on interpreting when/how to use a data structure
  + Implement part of a data structure in C++, modify a data structure in C++, give pseudocode for one, or explain how it works in writing.
  + Show by hand how a data structure would work.
* Bring something to write with. No notes, computers, or other materials will be allowed during the test.
* You will not be allowed to leave the classroom during the test; you must hand-in your exam if you want to go to the bathroom.
  + If you have a medical condition or disability, I will excuse you from this constraint if you provide me with appropriate documentation.
* I’ll provide a C++ reference sheet with all the C++ keywords and an ASCII table.
* An [old Exam 2](https://docs.google.com/document/d/1giZSsgcHs2MfmTBgkc0xLlV0FFGgy3pYv4DlDPsmxFc/edit?usp=sharing) with [solutions](https://docs.google.com/document/d/19_pEdmCrdcSu6tG5osfKmEmobgvWkSsxDHdyqVEZ1dk/edit?usp=sharing) has been provided on Google Drive in the Old Exams folder.

# Exam 2 review

This list of review topics is meant to guide your studying to the most important content. However, the instructor reserves the right to include material on the exam not explicitly listed here if it was covered in lecture, in assigned readings, or in a homework assignment.

1. C++ Language Features
   1. All of the C++ features from Exam 1 Review.
   2. Operator overloading, **operator=**
   3. **this** keyword
   4. Generic types/templates.
      1. Write a generic function or class.
      2. Specialize a generic function or class for a specific purpose.
   5. Implement code similar to any of the ZyLab programming assignments (in part or whole).
2. Hash Tables
   1. Organization of a hash table
   2. Critical properties in the design of a hash table
   3. Explain the differences between linear probing versus chaining
      1. Explain the rationale for empty-since-start and empty-after-removal in linear probing.
   4. Implement an integer hash function in C++.
   5. Draw a picture of a hash table after performing insertions and/or removals.
3. Binary Search Trees
   1. Describe the principles for how a BST is organized
   2. Draw a binary search tree based on input data
   3. Perform an insertion operation (by hand) on a BST
   4. Perform a removal operation (by hand) on a BST
   5. Write code to perform a search on a binary search tree, given a class definition for a node.
   6. Write code to perform an insertion on a binary search tree, given a class definition for a node.
   7. Write code to calculate the height of a binary search tree, given a class definition for a node.
   8. Explain the three cases for node removal on a BST. Given an example of each case.
4. Max heap
   1. Describe the organization of a max-heap; give the rationale for a max-heap’s organization - how does it help it achieve O(log n)
   2. Calculate children of a node in a max-heap.
   3. Calculate the parent of a node in a max-heap.
   4. Manually perform percolate up or percolate down.
   5. Manually perform insertion or removal on a max-heap.
   6. Write pseudo-code for percolate-up or percolate-down.
   7. Show a max-heap after performing insert or remove operations on it by hand.
5. Heapsort
   1. Perform the steps of the heapsort algorithm on an array, showing each step’s actions.
   2. Describe the time complexity of heapsort.
   3. Given a max-heap object and interface, implement a heap sort in C++.
6. Balanced Trees
   1. Calculate height and balance factor of a node in a BST.
   2. Describe the AVL balancing heuristic (all cases); write pseudo-code for the balancing heuristic.
   3. Show the steps to balance a BST after an insertion or removal.
   4. Describe the steps for inserting and removing nodes on an AVL tree in pseudocode.
7. Simple graphs and representation.
   1. Define the parts of a graph.
   2. Define adjacency.
   3. Describe the adjacency list and draw a picture of one for a graph.
   4. Describe the adjacency matrix and draw a picture of one for a graph.
   5. Implement code to represent a graph.
8. General data structure knowledge and know-how.
   1. Compare and contrast the complexity of insertion and removal of an item on the following data structures:
      1. List
      2. Hash table
      3. BST
      4. AVL tree
      5. Max-heap (removal of max only)
   2. Pick a data structure in order to achieve a stated performance goal or need.
   3. Pick/Use a data structure to solve a problem (like using a stack to balance parentheses) you’ve seen before.

# Exam 1 review

1. C++ Language Features
   1. class, public, protected, private
   2. Using classes, calling methods, accessing members
   3. Default function parameters
   4. references
   5. public/private inheritance
   6. using
2. Class Constructors and Destructors
   1. Default Constructors
   2. Specialized Constructors
   3. Destructor
   4. When do constructors and destructors execute for local, global, and heap objects
3. Allocating and deallocating classes using new and delete
   1. Use new or new[] for heap allocation
   2. Use delete or delete[] for freeing heap memory
4. Runtime complexity
   1. Evaluate the big-O runtime complexity of operations on a linked list
5. Data structure
   1. Define data structure and identify an example of one
6. Linked Lists
   1. Describe a linked list or doubly linked list; describe common operations on linked lists.
   2. Implement basic operations on a singly- or doubly-linked list in C++:
      1. append to end
      2. append to head
      3. remove from head
      4. remove from tail
      5. remove from middle
      6. insert into middle
      7. empty
      8. traversal using an iterator
   3. Explain the big-O time complexity of operations on linked lists
   4. Explain the purpose of the iterator class; change how it works
   5. Draw a picture of a linked list as an operation is performed on it.
7. Queue
   1. Describe the operation of a Queue and common functions in its interface
   2. Interpret the execution of code manipulating a Queue
   3. Implement a queue in terms of a linked list
   4. Evaluate the big-O complexity of operations on a Queue
8. Stack
   1. Describe the operation of a Stack and common functions in its interface
   2. Interpret the execution of code manipulating a Stack
   3. Implement a stack in terms of a linked list
   4. Evaluate the big-O complexity of operations on a Stack
9. Separate a class into source file and header file.
10. Inheritance: Public Inheritance
    1. Explain the difference between a base class and a derived class
    2. Explain the protected keyword and what it’s used for.
    3. Describe the benefits of public class inheritance; is-a relationship
       1. Describe how public, protected, and private members of the base class treated in the derived class
    4. Define polymorphism and its properties
       1. Virtual Functions
       2. Derived Types are treated as base types
    5. Define Abstract Base Class
       1. What’s a pure virtual function?
       2. What is a feature of an abstract base class?
11. Private inheritance
    1. Describe the benefits of private inheritance; has-a relationship
       1. Describe how public, protected, and private members of the base class treated in the derived class
       2. Has-A relationship between base and derived classes
    2. Explain when private inheritance makes sense versus simple composition
    3. Explain the using keyword and its usefulness for re-declaring inherited class members.