Final Review

This is a general study guide for the final. In addition to the list of topics below, study the class notes and the relevant sections of the textbook. Do not just memorize terms; most questions will be testing your understanding of the concepts. The exam will be closed book/notes and may contain several different kinds of questions (such as multiple choice, short-answer, debugging, coding, etc.). If you have to write any code, it will not be extensive, but you will be graded for syntax and correctness.

The exam will concentrate on the material covered after the midterm, but you are still expected to be able to answer questions on material from the first half of the quarter.

- **Overloading**

  Basics: What does it mean to overload a function? How do the prototypes of overloaded functions differ?

  Operators: Why and how do we overload operators? When do we make an overloaded operator a member of a class? Why shouldn't an overloaded `operator<<` be a member of a class?

- **Templates**

  Basics: What is the main idea behind templates? How do you define and how do you use a template function? How do you define and how do you use a template class? How do you define a template function/class with more than one type parameter?

- **Classes**

  Basics: What is the idea behind OOP? What is a class and what is an object? How is a class defined? What are `private`, `protected` and `public` members? What is *information hiding* and how is it enforced? Why and how do we separate the interface from the implementation? How is conditional compilation achieved?

  Special functions: What are the constructors, destructor, copy constructor? When and how are they called? When and why are they necessary? What are the main issues when overloading the assignment operator? What are `friend` functions? Why and how do we declare/define/use them? Can we avoid them, and if so, how?

  `this`: What is the `this` pointer? How is it used?

- **Inheritance**

  Basics: What is the main idea behind inheritance? What do we mean by the *is-a* relationship? What is public inheritance?

  Special functions: How are objects of derived classes constructed and destructed? Can a derived class use a base class method? Can a derived class override a base class method, and if so how? Why should we be careful with diamond inheritance?

  Virtual inheritance: What is virtual inheritance? When do we need to make a function virtual? How does it work? What is `dynamic_cast` and how is it used? Why should it be avoided?

- **Standard Template Library**

  Basics: What is the STL? What are iterators and how are they used? How are STL classes used?

  Vectors: Main characteristics of vectors. How are they implemented? How efficient are various operations (e.g. inserting at the front vs. inserting at the back)? When would you use a vector? You should be able to create an empty vector and use the following methods: `push_back()`, `pop_back()`, `front()`, `back()`, `empty()`, `size()`, `begin()`, `end()`

  Lists: Main characteristics of lists. How can a list be implemented? How efficient are various operations in singly-linked and doubly-linked lists? When would you use a list? You should be able to create an empty list and use the following methods: `push_back()`, `pop_back()`, `front()`, `back()`, `empty()`, `size()`, `begin()`, `end()`, `push_front()`, `pop_front()`, `insert(iterator, value)`.

  Stacks, Queues: Main characteristics. Why are there no iterators for them? When would you use a stack or a queue?

- **Sorting Algorithms**

  Basics: You should be able to apply insertion sort, selection sort, quicksort and mergesort. You should know the basic idea behind each algorithm. For the recursive ones you should know how the recursive part works and what the base case is.

- **I/O**

  Basics: What is a stream? How is I/O performed? What are the stream insertion and extraction operators and how do they work? What is type-safe I/O? What are the error bits and how do we check their values?

  File I/O: How is file I/O performed? What do the functions `peek()`, `seekg()`, `seekp()`, `eof()` do? How is unformatted I/O performed? How is `read()` different from >>?