Final Review

The final exam will concentrate on the material covered after the midterm. However, you should still be familiar with the topics from the first half of the class.

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.

- **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 and where can it be useful? What is the *uses-a* relationship?

  *Special functions*: How are objects of derived classes constructed and destroyed? Can a derived class use a base class method? Can a derived class override a base class method, and if so how? What assignments are allowed between derived and base class objects? Why should we be careful with diamond inheritance? How does virtual inheritance help?

  *Virtual methods*: When should a method be virtual? How does it work? What is *dynamic_cast* and how is it used?

- **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? When shouldn’t an overloaded operator be a member of a class? What difficulties arise when trying to overload operators such as ++?

- **Templates**
  
  *Basics*: What is the main idea behind templates? How do you define and how do you use a function template? How do you define and how do you use a class template?

- **Standard Template Library**
  
  *Basics*: What is the STL? What are iterators and how are they used? How are STL classes used?

  *Basic containers* (*vectors*, *lists*, *queues*, *stacks*): Main characteristics. How are they implemented? How efficient are various operations (e.g. inserting at the front vs. inserting at the back)? When would you use each one of them? Why don’t stacks and queues have iterators? You should be able to create an empty container, add elements to it and traverse it.

  *Algorithms*: You should be able to use the algorithms described in the class notes.
Function objects: Where do we use them? You should be able to create a function object and use it in conjunction with an algorithm.

• Sorting Algorithms
  Basics: What are the main ideas behind insertion sort and selection sort? What are their running times in the best/worst/average case? You should be able to apply them on an array of elements.

• I/O
  File I/O: What are streams? 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 `>>`?

• Strings
  C-strings: Strings as pointers to characters. The string.h library. Using the null character to signify the end of a string.
  C++ strings: Hiding the memory book-keeping required by C-strings. Overloading +, =, ==.