CMSC 202 Midterm 2 Topic Outline

Textbook Chapters and Sections:

Chapter 6
Chapter 7 (in Section 7.2, *not* including inline functions)
Chapter 8 (in Section 8.3, just through overloading << and >>)
Chapter 10 (some overlap with material on Exam 1)
Chapter 14
Chapter 18

Topic Outline

Classes and Objects

Basic Classes and Objects

Basic classes and objects will not be tested directly since they were covered on the first midterm exam; however, you will need to know how to create and use classes and objects to answer questions about advanced topics, e.g. constructors, inheritance.

Constructors

What is a constructor and how is one defined?
Why might you need to define your own constructors?
How do you invoke the constructors? In particular, how do you invoke the constructor with no arguments?
Why, if you are defining constructors in a class, should you always include a default constructor?
How is the *initialization section* of a constructor used? What, besides class variables, can be initialized in the initialization section?

Miscellaneous

How is const used to…
- indicate that a parameter should not be changed?
- indicate that a class method does not change class variables?

Why is it important to be consistent in your use of const?

What is the purpose of the vector class?
What advantages does a vector have over an array?
How do you declare a vector object?
How do you add elements to a vector?
How do you access elements of a vector?
How do you determined the size of a vector?

Operator Overloading

Why might you want to overload an operator?
Basic overloading: how would you overload +, - (binary), ==, or - (unary) *outside* the class? How would you overload them *inside* the class?
Why do we typically return a const value from an overloaded operator?

What is the purpose of declaring a function or operator to be a "friend" of a class?
How is friend used with operator overloading?

When we overload << or >>, we do *not* return a const value. What do we return?
How would you overload << or >> as a friend? As a non-friend?

Dynamic Memory and Pointers

`new` and `delete`

What is the purpose of the `new` operator, and how is it used?
What is the purpose of the `delete` operator, and how is it used?
Why, after using delete with a pointer, do we assign `NULL` to the pointer?
What is the "rule of thumb" for `new` and `delete`?

How would you create a dynamic array? How would you destroy it?

What is the purpose of the `->` operator? How would you use it? Why is this important when using dynamic memory (hint: what does `new` return?)

How do you create a linked list? How do you traverse a linked list?
Basic operations: insert at beginning, insert at end, determine length.

Destructors

Why do we need to defined destructors for classes that utilize dynamic memory?
How is a destructor defined?

Copy and Assignment

What is a *shallow copy*? What is a *deep copy*? Why does the difference matter?

Overloading assignment (=)

When do we need to overload assignment, i.e. when is the default assignment operator not sufficient? What is the essential function of the overloaded operator?

Why do we return `*this` from the overloaded assignment operator? Why is it returned by reference?

How is `this` used to prevent self-copying?

Copy Constructors

What is the purpose of a copy constructor? When are copy constructors used? Why are they needed for classes that utilize dynamic memory?

How do you define a copy constructor?

The parameter of a copy constructor is typically a const reference. Why does this make sense?

Inheritance

Be familiar with the "is a" relationship and how it relates to inheritance.

Basic Inheritance

What are the benefits of inheritance?
What is a *base class*?
What is a *derived class*?
How do you create a derived class from a base class?
What does the derived class inherit from the base class?
What does it mean for a derived class to *redefine* (or *override*) a function from the base class?

What is the purpose of the `protected` qualifier?

What is the difference between *redefining* (or *overriding*) and *overloading*?

Can a base class object be assigned to a derived class variable? What about the other way around?

Can a base class object call derived class functions? What about the other way around?

Exceptions

What are the advantages of exceptions over other methods of error handling? When is the use of exceptions appropriate?

When and how do you use `throw`, whether with a basic exception type or an exception class? How do you define an exception class?

When and how do you use `catch`? What is the purpose of `catch(…)`?