CSCI 2170 Spring 2006
Review for test 2 (Tuesday, March 14th 2006)

Class

- Copy constructor
  - Shallow copy vs. deep copy
  - C++ code for copy constructor in linked list
- Overloaded function
- Overloaded operator (==, !=, <=, =, >=)
- Destructor
  - When is it necessary to define a destructor for a class?
  - When is destructor called in a program?
  - Destructor for pointer-based list class

Recursion (chapters 2 and 5)

- Four basic components for each recursive function
- Trace program execution (including backtracking)
- Write recursive function for simple problems, as well as recursive functions

Algorithm Efficiency and sorting (chapter 9)

- Be able to analyze an algorithm and derive its **growth rate function**, f(n), and **run time complexity O(g(n))**

C++ pointers (chapter 4)

- Understand the difference between static and dynamic allocation of memory for variables
- Understand the meaning and how to use pointer related operators, such as ***, &, →, ., new, delete**
- Be able to set up and use pointers to
  - simple data types (int, float, char, …)
  - struct data types
  - 1D and 2D array
- Know how to pass pointer variable to function
- Understand the pros and cons of using array-based vs. pointer-based implementation of List

Linked list (chapter 4)

- Define linked list, Node
- Understand the code in pointer based implementation of listClass (**unsorted version**)
- Know how to **retrieve/insert/delete** nodes to/from the **beginning/middle/end of a list**

Practice questions:

1. questions in homework 3 (recursion) and Quiz 1 (overloaded operator)
2. Show output from the following C++ code segment:

```
typedef   int * intPtr;
void  func(int *, int);
int  main()
{
      intPtr   p, q;
      int    x, y;

      p = & x;
      *p = 4;
       q = & y;
       y=6;
       cout << *p << "  " << *q << "  " << x  << " " << y << endl;

        q = p;
```

```
    *p = 3;
    cout << *p << "  " << *q << "  " << x  << " " << y << endl;

    func (&x, y);
    cout << *p << "  " << *q << "  "  << x  << " " << y << endl;

    p = new int;
    *p = *q+y;
    x = x + *q;
    cout << *p << "  " << *q << "  "  << x  << " " << y << endl;

    delete p;

    p=NULL;
    q=NULL;

    return 0;
}
void func(int * p1,  int p2)
{       p2++;
    *p1 = *p1+p2;
}
```

3. Given a structure defined as:

```
struct   CarStruct
{        string   maker;
         string   model;
         float    price;
};
```

(a) declare a pointer variable "pointer" that can be used to a CarStruct type data
(b) dynamically allocate space for a CarStruct type data, and let "pointer" points to that memory location
(c) assign 24000.00 to be price of the newly created structure of type CarStruct
(d) release the memory located in (b) and make "pointer" not pointing to anything
(e) modify the CarStruct structure to include one additional member "next" which is a pointer to CarStruct type data
(f) set up a pointer variable "head" which can point to a linked list of records of type CarStruct defined in (e). Initialize head such that it does not pointing to anything
(g) Write a complete function "Insert" which receives one record of car information (maker, model, price for each car that is for sale), and store it in position specified by user into the linked list pointed at by "head".

```
bool List :: Inserte(CarStruct toad, int position)  // returns true if insertion is
                                                    // successful,  head is private data of class "List"
```

(h) Write a complete function "Delete" which will delete information of a car from the linked list, if it has been recorded in the list. Display appropriate message if the car is not found in the list.

```
bool  List::Delete(CarStruct toDelete)    // returns true if deletion is successful
                                          // head is private data of class "List"
```

(i) Write a function that display information of all cars in the list that has price between 10000 and 15000. void List::Display(int lowerPrice, int upperPrice)
(j) Write a complete function called DeleteWholeList which deletes (frees memory of) every node in the linked list. void List::DeleteList()