CSCI 211 **NAME** _____

March 2, 2012 **Minutes used:** _____

First Midterm Exam

- This test contains 9 questions worth a total of 86 points.
- Questions 1-6 have short answers and are 6 points each.
- Question 7 is a bit longer and is worth 8 points.
- Questions 7-9 require you to write code. They are worth 16 points each.
- You have 50 minutes to complete this exam.
- You may **not** use your text, notes, or any other reference material.
- The test with answers will be posted on the class webpage after the test.
- No electronic devices (music, phone, calculator, etc).
- **Do not turn this page until instructed to do so.**

Test Taking Advice

- The amount of space after a question does not always indicate how long the answer should be. Sometimes I add space so questions fit well on pages.

- Some questions have multiple parts such as "Explain your answer." Make sure you answer all the parts of each question.

- If you can't answer a question, move on and come back to it later. I often hear something like this, "I spent 40 minutes working on this 10 point problem and left 30 points worth of problems blank."

- If you have time left over, use it to review your answers. Students who turn tests in early often make trivial mistakes that they would catch if they went back over their answers. Some of my questions are difficult, go back and make sure you understood the question.

- If you don't understand a question ask me during the test. It is too late to ask for clarification after the exam.

1. (6 points) Define a class and an object.

   A class is a template for an object, it defines the member functions and the member variables. A class has no memory associated with it.

   An object is an instantiation of a class like in "int i" i is an instantiation of an integer. There is memory associated with an object. There can be many objects of the same class.

2. (6 points) What does the following code do?

```
assert(expression);
```

   First the expression is evaluated. If it is true, this code does nothing. If it is false, an "assertion failed" message is printed and the program is terminated.

3. (6 points) The -> operator is a shortcut for two other operators. For the 2nd statement below, show an equivalent statement using different operators (that is, do the same thing as this line w/o using ->).

```
Bill *my_bill = new Bill();
my_bill->print();
```

```
(*my_bill).print();
```

4. (6 points) The following code segment is from a working program. Until end of input is reached, it reads the fields of a Video object (title, url, comment, length, rating), instantiates a video object, and calls the new object's print function. While this program works, it has a problem. What is the problem? Briefly explain the problem.

```
int main()
{
    // declare variables

    while (getline(cin, title) )
    {
      // read the video's fields (url, rating, length, comment)

      Video *video = new Video(title, url, rating, length, comment);
      video->print();
    }
}
```

   It has a memory leak.

   This program allocates dynamic memory (using new) but never deletes it. Each time through the loop the video pointer is overwritten. Thus the address returned from new is "lost" and delete can not be called on that memory.

5. (6 points) You download the posted tests and the run_tests script for p3. You run the tests using the run_tests script and it reports that you fail test t13 because your standard output is incorrect. How can you discover what is incorrect about the output from your program?

The correct output for test t13 is in the file tests/t13.out
The output your program generated is in the file results/t13.myout

In order to determine the difference you must compare these two files. You could use diff:

```
$ diff tests/t13.out results/t13.myout
```

or you could use vimdiff (which will highlight the differences in red):

```
$ vimdiff tests/t13.out results/t13.myout
```

6. (8 points) The following program compiles and runs. What does each cout statement in the following program print? This question is very tricky, be careful! **Briefly explain your answers or you will not get any points.**

```
#include <iostream>
using namespace std;


void f(int &value)
{
    if (value == 0)
    {
        value += 1;
    }
    else
    {
        int value = 42;
    }
}

void g(int value)
{
    value += 10;
}

void h(int *value)
{
    *value += 100;
}

int main()
{
    int j = 10;

    f(j);                   10:  j is passed by reference, but in f() the argument
    cout << j << endl;           (value) is not changed.  A local variable (also
                                 called value) is changed.


    g(j);                   10: j is passed by value and thus not changed by g()
    cout << j << endl;


    h(&j);                  110: j is passed by address and changed by h()
    cout << j << endl;

}
```

For the following questions, use the code on the last page. Tear off last page for easy reference.

7. (16 points) Write the function `int List::within_range(int lower_bound, int upper_bound)` that returns the number of elements in the list that are greater or equal to `lower_bound` **and** less than or equal to `upper_bound`. Return 0 if the list is empty.

```
int List::within_range(int lower_bound, int upper_bound)
{
    // regular case handles empty list
    int count = 0;
    Node *ptr = m_head;
    while (ptr != NULL)
    {
        if (lower_bound <= ptr->m_value && ptr->m_value <= upper_bound)
          count++;
        ptr = ptr->m_next;
    }
    return count;
}

// same as above
int List::within_range(int lower_bound, int upper_bound)
{
    int count = 0;
    for (Node *ptr = m_head; ptr; ptr = ptr->m_next)
        if (lower_bound <= ptr->m_value && ptr->m_value <= upper_bound)
          count++;
    return count;
}
```

8. (16 points) Write a List member function that finds and removes the largest number in the list (the list is NOT sorted). Return false if the list was empty (cannot remove from an empty list) and true if it was not empty (a value is always removed if the list is not empty). You have to figure out how the argument should be used. Hint: don't do this all at once, break it into pieces and implement each piece.

```
bool List::remove_largest(int &largest)
{
  // special case, empty list
  if (m_head == NULL)
    return false; // empty list, nothing removed

  // find the largest (works for list with only 1 element)
  largest = m_head->m_value;
  Node *ptr = m_head->m_next;
  while (ptr != NULL)
  {
    if (ptr->m_value > largest)
    {
      largest = ptr->m_value;
    }
    ptr = ptr->m_next;
  }

  // at this point the variable largest contains the largest value
  // remove the largest value

  // special case, first element is largest
  if (m_head->m_value == largest)
  {
    Node *tmp = m_head;
    m_head = m_head->m_next;
    delete tmp;
  }
  // regular case, first element is not largest
  else
  {
    Node *ptr = m_head;
    while (ptr->m_next != NULL && ptr->m_next->m_value != largest)
      ptr = ptr->m_next;

    assert(ptr->m_next != NULL); // largest must be in list (found above)
    assert(ptr->m_next->m_value == largest);

    Node *tmp = ptr->m_next;
    ptr->m_next = ptr->m_next->m_next;
    delete tmp;
  }

  return true;
}
```

9. (16 points) Write a function that (1) dynamically allocates an array of integers the exact size of the list, (2) copies all the list's values into the array, (3) assigns the size of the array (which is also the length of the list) to the reference parameter size, and (4) returns the array. Assume there exists a function you can call that returns the number of elements in the list: `int List::length()`. If the list is empty, return NULL. Hint: you can dynamically allocate an array of 10 integers using the following statements:

```
int length = 10;
int *array = new int[length];
```

```
int *List::convert_to_array(int &size)
{
    size = length();

    if (size == 0)
    {
        return NULL;
    }

    int *array = new int[size];

    Node *ptr = m_head;
    int i = 0;
    while (ptr != NULL)
    {
        array[i] = ptr->m_value;
        ptr = ptr->m_next;
        i++;
    }
    return array;
}

// here is a more compact version
// (a little too compact, this version is harder to read and understand)
int *List::convert_to_array(int &size)
{
    if (!length())
        return NULL;
    int i = 0, *array = new int[size = length()];
    for (Node *ptr = m_head; ptr; i++, ptr = ptr->m_next)
        array[i] = ptr->m_value;
    return array;
}
```

Use the following class definitions for questions 7,8, and 9. You **may not** alter or add to this class definitions.

You may tear this page off so it is easier to reference.

```
class List
{
    public:
     List() {m_head = NULL;}
     void insert(int value); // insert values at the head of the list
     int length(); // returns number of elements in the list

     // Functions you have to write
     bool remove_largest(int &largest);
     int within_range(int lower_bound, int upper_bound);
     int *convert_to_array(int &size);


    private:
     class Node
     {
         public:
          Node (int value, Node *next) {m_value = value; m_next = next;}
          int m_value;
          Node *m_next;
     };
     Node *m_head;
};
```