CSCI 211 NAME ________________________________

September 28, 2011 **Minutes used: ______**

First Midterm Exam

- This test contains 9 questions worth a total of 90 points.
- Questions 1-3 have short answers and are 6 points each.
- Question 4-6 are a bit longer and are 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) Which of the two following code fragments is better? Why?

```
// I think size is greater than 0 so this will work
for (int i = size; i != 0; i--)
      cout << i << endl;


assert(size > 0);
for (int i = size; i != 0; i--)
      cout << i << endl;
```

The second is better. In the second, if size is less than 0 the assert statement will print an error message and terminate the program.

In the first if size starts less than 0 the loop is an infinite loop and the program will never terminate (actually it will terminate when i underflows (runs out of bits and the smallest negative number becomes the largest positive number) and then counts down to 0, but this process is long enough that we can call it “an infinite loop”)).

2. (6 points) Describe what each of the following statements does.

int i = 42;	Declares an integer variable called i, initializes it to the value 42
int *ptr = & i;	Declares a pointer variable called ptr, it is a pointer to an integer. Initializes ptr to point to the address of i (the location in memory where i is stored).

3. (6 points) When `m_head == NULL`, the following would result in a segmentation fault:

```
if (m_head->m_value > value)
```

The following code fragment from a linked list insert function does not result in a segmentation fault when `m_head == NULL`. Why not?

```
if (m_head == NULL || m_head->m_value > value)
{
    m_head = new Node(value, m_head);
}
```

Expressions are evaluated using short circuit evaluation. When `m_head == NULL`, the second clause of the expression (`m_head->m_value > value`) is not evaluated. Since (true OR ____) is always true there is no reason to evaluate ____.

4. (8 points) For each statement in the program below that has a blank line after it: if this statement will cause a compilation error write error and very briefly explain the error. If it does not cause a compilation error, write the text that will be printed when this statement executes.

```
class Point
{
  public:
    Point(int x, int y) {m_x = x; m_y = y;}
    void print() {cout << "(" << m_x << "," << m_y << ")";}
  private:
    int m_x;
    int m_y;
};

int main()
{
    Point *p1 = new Point(1,2);
    Point p2(3,4);

    p1->print();       (1,2)

    p1.print();        error: p1 is a pointer, can only use . with objects

    *p1.print();       error: . has higher precedence than *, that means this
                       is just like above (the p1.print() happens before the *

    (*p1).print();     (1,2)

    p2->print();       error: p2 is not a pointer it is an object, cannot
                       dereference an object

    p2.print();        (3,4)

    (*p2).print();     error: p2 is not a pointer it is an object, cannot
                       dereference an object

    (&p2)->print();    (3,4)
}
```

5. (8 points) Given the following declarations, for each if statement, circle true if the if statement will evaluate to true, false if it will evaluate to false. Explain your answer for any credit.

```
int i = 42;
int j = 42;

int *ptr1 = &i;
int *ptr2 = &j;

if (i == j)           true    false  true: values of i and j are 42


if (&i == &j)         true    false  false: i and j are different variables
                                            and thus have different addresses


if (ptr1 == ptr2)     true    false  false: ptr1 points to i, ptr2 points to j
                                            i and j are different variables


if (*ptr1 == *ptr2)   true    false  true: *ptr1 is the value of i which is 42
                                            *ptr2 is the value of j which is 42
```

6. (8 points) 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)
{
    value += 1;
}
void g(int value)
{
    value += 10;
}
void h(int *value)
{
    value += 100;
}

int main()
{
    int i = 10;

    f(i);                      i is passed by reference and thus changed by f()
    cout << i << endl;         prints: 11

    g(i);                      i is passed by value and thus not change by g()
    cout << i << endl;         prints: 11

    h(&i);                     i is passed by address to h() BUT h() changes the
                               value of the pointer (the address or what the pointer
                               points to), not the value pointed to by the pointer,
                               thus the value does not change

    cout << i << endl;         prints: 11

}
```

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::count_positive_numbers()` that returns the number of positive numbers (greater than or equal to 0) in the list. Return 0 if the list is empty.

```
int List::count_positive_numbers()
{
    int count = 0;

    for (Node *ptr = m_head; ptr != NULL; ptr = ptr->m_next)
    {
        if (ptr->m_value >= 0)
            count++;
    }
    return count;
}
```

8. (16 points) Write a member function that finds an element in a linked list by its index (the first element in the list has index == 0, the second has an index == 1, and so forth). If the index is >= 0 and the element is in the list (i.e. the list is long enough to contain the requested element), put the value in the reference parameter *value* and return true. Otherwise return false. It should not matter if the list is sorted or not sorted. For example, if the list contained the numbers {17,42,86,99} then the number with index == 0 is 17 (that is, the value found for index == 1). The number with index == 1 is 42, and so on.

```
bool List::get_element_by_index(int index, int &value)
{
    // special case, illegal index
    if (index < 0)
    {
        return false;
    }

    // look through the list decrementing index at each step
    // when index == 0 we have counted down to the correct element
    Node *ptr = m_head;
    while (ptr != NULL)
    {
        if (index == 0)
        {
            value = ptr->m_value;
            return true;
        }
        index = index - 1;
        ptr = ptr->m_next;
    }
    // if the value was found, the return above would have already returned
    // thus if we reach here the value was not in the list
    return false;
}
```

9. (16 points) Write a function that removes all duplicate entries in the list. Assume that the list is ordered from smallest to largest, thus all duplicates will be next to each other. For example, if the list contained {1,2,2,2,3,3,4,5,5,5,5,5,6} before calling remove_duplicates(), it should contain {1,2,3,4,5,6} after. Do nothing if the list is empty or contains no duplicates.

```
void List::remove_duplicates()
{
    // consider all nodes in the list
    Node *ptr = m_head;
    while(ptr != NULL)
    {
        // while there is a next node
        // and next node has same value as the current node, remove next node
        while (ptr->m_next != 0 && ptr->m_value == ptr->m_next->m_value)
        {
            Node *tmp = ptr->m_next;
            ptr->m_next = ptr->m_next->m_next;
            delete tmp;
        }
        ptr = ptr->m_next;
    }
}
```

```
The following is another solution
void List::remove_duplicates()
{
    Node *ptr = m_head;
    while(ptr != NULL && ptr->m_next != NULL)
    {
        if (ptr->m_value == ptr->m_next->m_value)
        {
            Node *tmp = ptr->m_next;
            ptr->m_next = ptr->m_next->m_next;
            delete tmp;
        }
        // the "else" below is very important
        // without it, this function would not remove all duplicates
        else ptr = ptr->m_next;
    }
}
```

Use the following class definitions for questions 7, 8, & 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); // inserts values so list is sorted smallest to largest

     // Functions you have to write
     int count_positive_numbers();
     bool get_element_by_index(int index, int &value);
     void remove_duplicates();

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