CSCI 211 **NAME** _____
March 4, 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) Describe the difference between private member functions and public member functions.

   private member functions can only be called from other member functions of the given class

   public member functions can be called from any part of the program

2. (6 points) The following code compiles without error. What happens when it is executed? What would happen if the assert was removed?

```
int *ptr = NULL;
assert(ptr != NULL);
*ptr = 42;
cout << *ptr << endl;
```

   Since the expression passed to assert() evaluates to false, assert prints an "Assertion failed" error message (which includes the filename and the line number) and terminates the program.

   If the assert was removed *ptr would cause a segmentation fault ("Segmentation Fault" is printed and the program is terminated).

3. (6 points) Given the following class definition:

```
class Dog
{
    public:
        Dog(int age, int weight)
        {m_age = age; m_weight = weight;}
        void print()
        {cout << "dog's age = " << m_age << " weight = " << m_weight <<endl;}
    private:
        int m_age;
        int m_weight;
};
```

Write code to instantiate a Dog object without using new and then call it's print function. Now do the same thing using new.

```
// instantiate w/o using new
Dog sam(1,21);
sam.print();

// instantiate using new
Dog *spot = new Dog(11,103);
spot->print();
```

4. (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

```
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
```

5. (8 points) What does each cout statement in the following program print?

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

int main()
{
    int i = 42;
    int *ptr = &i;

    cout << &i << endl;         // the address of i

    cout << ptr << endl;        // the address of i

    cout << &ptr << endl;       // the address of ptr

    cout << *ptr << endl;       // 42, the value of i

    cout << *(&i) << endl;      // 42, the address of i (&i) is dereferenced
                                // using the * operator, so we get the value of i
}
```

6. (8 points) What does the following program print? This question is very tricky, be careful!
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 = 17;

    f(i);                       i is passed by value and thus not changed by f()
    cout << i << endl;          prints: 17

    g(i);                       i is passed by reference and thus change by g()
    cout << i << endl;          prints: 27

    h(&i);                      i is passed by address to h() and thus changed by h()
    cout << i << endl;          prints: 127

}
```

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

7. (16 points) Write the function `void List::compare()` that compares the given value to each element in the list and counts the numbers in the list that are less_than, equal to, and greater_than the value. For example, if the list contained {1,2,3,4} and the number 3 was passed as the value parameter to `compare()`, `compare()` would find that 2 numbers in the list are less than 3, 1 number is equal to 3, and 1 number is greater than 3.

```
void List::compare(int value, int &less_than, int &equal, int &greater_than)
{
    less_than = 0;
    equal = 0;
    greater_than = 0;
    Node *ptr = m_head;
    while (ptr != NULL)
    {
        if (ptr->m_value < value)
            less_than++;
        else if (ptr->m_value > value)
            greater_than++;
        else equal++;
        ptr = ptr->m_next;
    }
}
```

8. (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 = list_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 (!list_length())
        return NULL;
    int i = 0, *array = new int[size = list_length()];
    for (Node *ptr = m_head; ptr; i++, ptr = ptr->m_next)
        array[i] = ptr->m_value;
    return array;
}
```

9. (16 points) Write a list member function that removes all even numbers from the list. Also write the helper function `bool is_even(int num)` that returns true if the argument is even. If you don't know how to write the helper function, just assume it works and write the remove function. Assume the list is unsorted. Do nothing if the list is empty. Hint: make sure your code works if there are many even numbers in a row; don't forget to consider any possible special cases.

```
// return true if given number is even, false otherwise
bool is_even(int num)
{
  // when the result of (num % 2) is zero (false) the number is evenly divisible
  // by 2, thus the number is even.  In other words, when (num % 2) is false
  // the number is even, so !(num % 2) is true when number is even
  return !(num % 2);
}


void List::remove_even_numbers()
{
    // special case, one or more even number at head of list
    while (m_head != NULL && is_even(m_head->m_value))
    {
        Node *tmp = m_head;
        m_head = m_head->m_next;
        delete tmp;
    }

    // look for even numbers after the first
    Node *ptr = m_head;

    while (ptr->m_next != NULL)
    {
        // as long as the node after ptr is even, delete it
        while (ptr->m_next != NULL && is_even(ptr->m_next->m_value))
        {
            // delete the node after ptr
            Node *tmp = ptr->m_next;
            ptr->m_next = ptr->m_next->m_next;
            delete tmp;
        }
        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;}
     int length(); // returns number of values in list (you may call this function)
     void insert(int value); // inserts value at front of list

     // Functions you have to write
     void compare(int value, int &less_than, int &equal, int &greater_than);
     int *convert_to_array(int &size) ;
     void remove_even_numbers();

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