CSCI 211 **NAME** _____

March 6, 2013 **Minutes used:** _____

First Midterm Exam

- This test contains 9 questions worth a total of 82 points.
- Questions 1-5 have short answers and are 6 points each.
- Question 6 is worth 8 points.
- Questions 7-9 require you to write code. They are worth 12, 16, and 16 points.
- 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) What is a pointer? Give example code that declares a pointer and shows the pointer being used on the left-hand-side of an assignment statement.

   A pointer is a variable that stores the address of location in memory (the index into the large array that is memory).

```
int i = 0;
int *ptr = &i;  // ptr is a pointer variable.  It is initialized to hold the
address of the variable i.
*ptr = 42;       // the * dereferences the pointer and puts 42 at the location
help by ptr.
                      // the result is that i will have the value 42.
```

2. (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();
```

3. (6 points) What does the C-preprocessor do? How can you run the C-preprocessor without compiling your code?

   The C-preprocessor handles all the "#" directives. For example, the C-preprocessor interprets #include <iostream> to mean that all the code in this file should be inserted into the temporary file that will be used by the compiler. The C-preprocessor also handles the #ifndef/#define/#endif that prevents .h files from being included multiple times.

   The following runs the C-preprocessor w/o compiling the program. In this example, the output of the C-preprocessor is put in the file preprocessor.out.

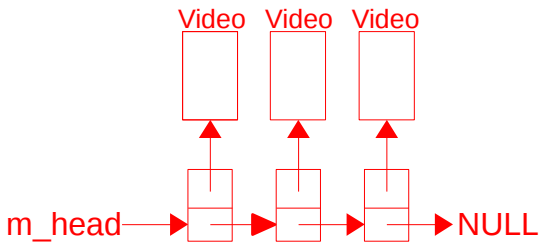
```
$ g++ -E  >  preprocessor.output
```

4. (6 points) The following function inserts a value at the end of a linked list. Something important is missing from this function. Briefly describe in words what is missing. Do not write the missing code, just explain what should be done that is not done.

```
void List::insert_end(int value)
{
    // You don't have to write this code.
    // I'm including it to help future classes prepare for future exams
    if (m_head == NULL)
    {
        m_head = new Node(value, m_head);
    }
    else
    {
        Node *ptr = m_head;
        while (ptr->m_next != NULL)
        {
            ptr = ptr->m_next;
        }
        ptr->m_next = new Node(value, NULL);
    }
}
```

This code does not check the special case of an empty list. If the list is empty, m_head should point to the new Node and the above block of code should not be executed.

5. (6 points) Draw a boxes and arrows diagram (like the diagrams I draw in class) of the data structure required for assignment 3. Specifically: a linked list of pointer to video objects. Draw the list with three videos.



6. (8 points) The following is a legal and working program. What does it print? **This is a tricky question. Look carefully at the code.** Briefly explain your answer or you will not get any points.

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

void compare(int values[], int size, int target, int equal)
{
    equal = 0;

    for (int i = 0; i < size; i++)
    {
        if (values[i] == target)
        {
          equal++;
        }
    }
}



int main()
{
    int values[] = {1,2,3,4,5,5,5,6,7,8};

    int equal = 0;

    compare(values, 10, 5, equal);

    cout << equal << endl;
}
```

The program prints 0

equal is passed by value
so even though the equal in the function is change, the equal in main() is not changed

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

7. (12 points) Write the function contains_duplicates() that returns true if the List contains any duplicate entries and false if it contains no duplicate entries. Assume the list is sorted from smallest to largest.

```
bool List::contains_duplicates()
{
    // a list with zero or one element cannot have duplicates
    if (m_head == NULL || m_head->m_next == NULL)
    {
        return false;
    }

    // we know there are 2 or more elements
    Node *ptr = m_head;
    while (ptr->m_next != NULL)
    {
        // all duplicates must be neighbors
        if (ptr->m_value == ptr->m_next->m_value)
            // we found a duplicate, we are done
            return true;
        ptr = ptr->m_next;
    }
    // looked at the entire list and did not find any duplicates: must not be any
    return false;
}
```

8. (16 points) Write the List member function `rotate_last_to_first()` that removes the last element of the list and inserts that element at the front of the list. For example, if the list contains the values {1,2,3,4} and `rotate()` is called once, the list will contain the values {4,1,2,3}. Your solution cannot have any memory leaks.

```
// move last element to first front of the list
void List::rotate_last_to_first()
{
  // special cases of empty list and a list with one element
  if (m_head == NULL || m_head->m_next == NULL)
  {
    return; // do nothing for these cases
  }

  // we know there are at least two elements in the list
  assert(m_head && m_head->m_next);

  // find the second to the last node
  Node *second_to_last = m_head;
  while (second_to_last->m_next->m_next != NULL)
  {
    second_to_last = second_to_last->m_next;
  }

  // save the last node
  Node *last_node = second_to_last->m_next;

  // delete the last node from the end of the list
  second_to_last->m_next = NULL;

  // insert the last node at the front of the list
  last_node->m_next = m_head;
  m_head = last_node;

}
```

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. This function should NOT change the list. Hint: you can dynamically allocate an array of integers large enough to hold all the elements in the linked-list using the following statement:

```
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 definition for questions 7,8, and 9. You **may not** alter or add to this class definition.

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

```
class List
{
    public:
     List() {m_head = NULL;}
     ~List();
     int length(); // return number of elements in the array
     void insert_at_end(int value);

     // Functions you have to write
     bool contains_duplicates();
     void rotate_last_to_first();
     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;
};
```