CSCI 211 **NAME** ________________________________

October 3, 2012 **Minutes used:** ______

First Midterm Exam

- This test contains 9 questions worth a total of 92 points.
- Questions 1-3 have short answers and are 6 points each.
- Questions 4-5 are a bit longer and are worth 8 points each.
- Question 6 is worth 10 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) 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 = 42"would cause a segmentation fault ("Segmentation Fault" is printed and the program is terminated).

2. (6 points) Assume that class Video has a function `print()` that takes no arguments. Show two ways `print()` can be called for the object pointed to by the `video` variable declared below. Hint: the two ways use different operators that do the same thing.

```
// Assume the following compiles w/o any errors
Video *video = new Video(title, url, comment, length, rating);

(*video).print();
video->print();
```

3. (6 points) You turn in assignment p3 on turnin.ecst.csuchico.edu and find that you fail test t13 because the standard output is incorrect. How can you run the test on a computer other than turnin and figure out what is incorrect about the output from your program? The first two steps are given.

   1) Download the test files for p3 and the run_tests script (~tyson/211/tests/p3/tests.tar) to the directory that contains your program.

   2) untar the files: $ tar -xvf tests.tar

   3) Execute the tests: $ run_tests videos

   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

   4) 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
```

4. (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 << 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
}
```

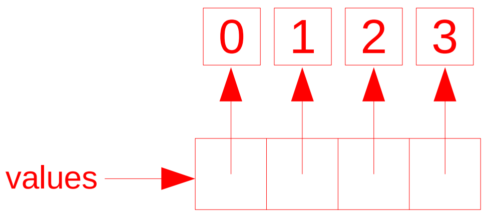
5. (8 points) Draw a boxes and arrows diagram (like the diagrams I draw in class) of the data structure created by the following code. What does this program print?

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


int main()
{
  int *values[4];

  for (int i = 0; i < 4; i++)
    values[i] = new int(i);

  for (int i = 0; i < 4; i++)
    cout << *values[i] << endl;
}
```



```
It prints:
0
1
2
3
```

6. (10 points) What does the following program print? This question is a little bit tricky. Explain your answer or you will not get any points.

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

bool is_true()
{
    cout << "is_true()" << endl;          each time called “is_true()” is printed
    return true;
}
bool is_false()
{
    cout << "is_false()" << endl;         each time called “is_false()” is printed
    return false;
}


int main()
{
  if (is_true() && is_false())       is_true() returns true, is_false() is called
      cout << "1 true" << endl;
  else cout << "1 false" << endl;    prints false:  (true && false) is false

  if (is_false() && is_true())       is_false() returns false, don't call is_true()
      cout << "2 true" << endl;
  else cout << "2 false" << endl;    print false:   (false && true) is false

  if (is_true() || is_false())       is_true() returns true, don't call is_false()
      cout << "3 true" << endl;
  else cout << "3 false" << endl;    prints true:    (true || ANYTHING) is true

  if (is_false() || is_true())       is_false() returns false, must call is_true()
      cout << "4 true" << endl;
  else cout << "4 false" << endl;    prints true:    (false || true) is true
}
```

This program demonstrates the short circuit evaluations of expressions. Consider (A && B), if A is false, don't need to evaluate B because (A && B) is false. If A is true must evaluate B. Consider (A || B). If A is true no need to evaluate B (A || anything) is true. if A is false, must evaluate B.

```
is_true()
is_false()
1 false
is_false()
2 false
is_true()
3 true
is_false()
is_true()
4 true
```

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. Modify the three reference parameters so they contain the correct counts (e.g. less_than should contain the number of elements *less than* value). Assume the list is not sorted.

```
void List::compare(int value, int &less_than, int &equal, int &greater_than)
{
    // must initialize reference parameter counters
    // so they start at zero
    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 the List member function `rotate()` that removes the first element of the list and inserts that element at the end 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 {2,3,4,1}. Your solution cannot have any memory leaks.

```
void List::rotate()
{
  // 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
  }

  Node *start = m_head;
  assert(start && start->m_next);

  // remove start node
  m_head = m_head->m_next;


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

  // insert the start node at end of list
  last->m_next = start;

  // update the "new" last node's m_next to point to NULL
  start->m_next = NULL;
}
```

9. (16 points) Consider a list that contains the values {4,2,7,1,17,42}. The sub-list (2,17) (that is, the sub-list that starts with 2 and ends with 17) contains four values: {2,7,1,17}. The sub-list (7,1) contains 2 values: {7,1}. The sub-lists (2,86) and (99,7) are not in the list. Write the function `sub_list_length()` that takes a starting value and an ending value. If there is a sub-list that starts and ends with the given values, return the length of the sub-list (the shortest sub-list will be 2). If there is no such sub-list, return 0. Assume the list is NOT sorted and contains NO DUPLICATES.

```
int List::sub_list_length(int start_value, int end_value)
{
  // find the node that contains the start_value
  Node *start_node = m_head;
  while (start_node && start_node->m_value != start_value)
  {
    start_node = start_node->m_next;
  }

  // if start_value not in list
  if (start_node == NULL)
  {
    return 0;
  }

  assert(start_node != NULL && start_node->m_value == start_value);

  // use size to count the number of elements between start_value and end_value
  int size = 0;

  // starting at the node after start_node
  // search through the list until we find end_value, or reach end
  Node *end_node = start_node->m_next;
  while (end_node && end_node->m_value != end_value)
  {
    end_node = end_node->m_next;
    size++;
  }

  // if reached the end of the list w/o finding end_value
  if (end_node == NULL)
    return 0;

  // we found end value, return size
  else
  {
    assert(end_node != NULL && end_node->m_value == end_value);
    return size + 2; // "+ 2" add start_node and end_node to the count
  }
}
```

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;}
     ~List();
     void insert(int value) {m_head = new Node(value, m_head);}

     // Functions you have to write
     void compare(int value, int &less_than, int &equal, int &greater_than);
     void rotate();
     int sub_list_length(int start_value, int end_value) ;


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