CSCI 211 **NAME** _____

October 2, 2009 **Minutes used:** _____

First Midterm Exam

- This test contains 9 questions worth a total of 76 points.
- Questions 1-4 have short answers and are 6 points each.
- Question 5-6 are a bit longer and are worth 8 points.
- Questions 7-9 require you to write code. They are worth 10, 10, and 16 points respectively.
- 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 creates a pointer and shows how it can be used on the left-hand-side of an assignment statement.

   A pointer is a variable that stores the address of location in 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) What does the following code do?

   ```
   assert(value > 42);
   ```

   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. This message includes the filename and line number of the assert statement so it is easy to find the problem.

3. (6 points) There are two ways that a destructor is called. Explain both of them. Example code might help explain.

   The destructor for statically declared objects is called when the variable goes out of scope (such as at the end of the enclosing block of {}):

   ```
   void f()
   {
        List list;
        …
   }  // destructor for list is called here
   ```

   The destructor for dynamically allocated objects is called when the object is deleted using the delete operator:

   ```
   Node *ptr = new Node(42, NULL);
   …
   delete ptr;  // destructor for ptr is called here
   ```

4. (6 points) The following program has a single compilation error on the second to the last line. What is the error? How could this code be changed so that error does not occur?

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

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

int main()
{
    Point *p = new Point(86,99);
    p.print();  // the error is issued for this line
}
```

The variable p is a pointer. The "." can only be used for a non-pointer. Thus an error that p is not a class type is issued.

This code can be fixed by replacing the "." with a "->"

```
p->print();
```

Or it can be fixed by using the "(*)." notation

```
(*p).print();
```

5. (8 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 g()
{
    cout << "g()" << endl;              each time g() is called "g()" is printed
    return true;
}
bool h()
{
    cout << "h()" << endl;              each time h() is called, "h()" is printed
    return false;
}

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

    if (h() && g())                     since h() returns false, h() not called
        cout << "2 true" << endl;
    else cout << "2 false" << endl;     print false:   (false && true) is false

    if (g() || h())                     since g() returns true, no need to call h()
        cout << "3 true" << endl;
    else cout << "3 false" << endl;     prints true:   (true || ANYTHING) is true

    if (h() || g())                     since h() returns false, must call g()
        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.

```
g()
h()
1 false
h()
2 false
g()
3 true
h()
g()
4 true
```

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 += 2;
}
void g(int &value)
{
    value += 2;
}
void h(int *value)
{
    value += 2;
}

int main()
{
    int i = 42;

    f(i);                       i is passed by value and thus not changed by f()
    cout << i << endl;          prints: 42

    g(i);                       i is passed by reference and thus change by g()
    cout << i << endl;          prints: 44

    h(&i);                      i is passed by address to h()
    cout << i << endl;          however, the address is changed, not the integer
                                prints: 44

}
```

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

7. (10 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.

```
bool List::get_element_by_index(int index, int &value)
{
    if (index < 0)
    {
        return false;
    }
    Node *ptr = m_head;
    while (ptr != NULL)
    {
        if (index == 0)
        {
            value = ptr->m_value;
            return true;
        }
        index = index - 1;
        ptr = ptr->m_next;
    }
    return false;
}
```

8. (10 points) Write a list function that counts the number of elements within a range. Return the number of of elements in the list with values greater than or equal to *min* **and** less than or equal to *max*. If the list is empty, return 0. Assume that *max* is greater than or equal to *min*. Assume the list is NOT sorted.

```
int List::count_values_in_range(int min, int max)
{
    int count = 0;
    Node *ptr = m_head;
    while (ptr != NULL)
    {
        if (min <= ptr->m_value && ptr->m_value <= max)
        {
            count++;
        }
        ptr = ptr->m_next;
    }
    return count;
}
```

9. (16 points) Write a function that removes the specified element from the list. Return true if the target number is found. Return false if the target number is not in the list. If the target element is in the list more than once, remove only the first instance. Assume the list is not sorted.

```
bool List::delete_value(int target)
{
    // special case, empty list so target is NOT in the list
    if (m_head == NULL)
        return false;

    // special case, the value we are looking for is at head of list
    // this is special because the value of m_head would change
    if (m_head->m_value == target)
    {
        Node *tmp = m_head;
        m_head = m_head->m_next;
        delete tmp;
        return true;
    }

    // the list is not empty, the value we are looking for is not the first
    // find the node BEFORE the one we want to delete
    Node *ptr = m_head;
    while (ptr->m_next != NULL && ptr->m_next->m_value != target)
    {
        ptr = ptr->m_next;
    }

    // did we walk off the end of the list in the above loop w/o finding the target?
    if (ptr->m_next == NULL)
    {
        // did not find target in the list, cannot remove it
        return false;
    }

    assert(ptr->m_next != NULL);

    Node *tmp = ptr->m_next;
    ptr->m_next = ptr->m_next->m_next;
    delete tmp;
    return true;
}
```

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 value at front of list

     bool get_element_by_index(int index, int &value);
     int count_values_in_range(int min, int max);
     bool delete_value(int target);

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