CSCI 112 **NAME _____**
March 6, 2008 **Minutes used: _____**

First Midterm Exam

- This test contains 10 questions worth a total of 100 points.
- Questions 1-4 have short answers and are 5 points each.
- Question 5 & 6 are a bit longer and are worth 10 points.
- Questions 7-10 require you to write code. They are worth 10, 15, 15, and 20 points respectively.
- You have 75 minutes to complete this exam.
- You may **not** use your text, notes, or any other reference material.
- 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. (5 points) The -> operator is a shortcut for two other operators. For the 2nd statement below, show an equivalent statement using different operators (that is, the other two operators).

```
Bill *my_bill = new Bill();
my_bill->print();
```

```
(*my_bill).print();
```

2. (5 points) What does the following code do?

```
assert(expression);
```

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.

3. (5 points) What does the following code print? You must explain your answer for any credit.

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

void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}

int main()
{
    int i = 42;
    int j = 86;

    swap(i,j);
    cout << "i = " << i << " j = " << j << endl;
}
```

It prints:

i = 42 j = 86

Since the arguments to swap() are pass-by-value, i & j are not changed by the call to swap().

4. (5 points) Does the following program compile? Explain?

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

int main()
{
    int i = 42;
    double *x = &i;

    cout << *x << endl;

    return 0;
}
```

No. The type of x is a pointer to a double. Since i is an int, &i is an address of an int. You cannot assign an int address to a double pointer. Only double address can be assigned to a double pointer.

5. (10 points) What does the following code print? You must explain your answer for any credit.

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

bool f()
{
    cout << "f()" << endl;
    return true;
}

bool g()
{
    cout << "g()" << endl;
    return true;
}

bool h()
{
    cout << "h()" << endl;
    return false;
}

int main()
{
    if (f() || g() || h())
        cout << "true" << endl;
    else cout << "false" << endl;

    if (f() && g() && h())
        cout << "yes" << endl;
    else cout << "no" << endl;
    return 0;
}
```

The short-circuit evaluation of logical expressions is key to understanding this program.

The expression in the first if statement is true. This can be determined as soon as f() returns. Thus g() and h() are not called.

The expression in the second if is false but this can only be determined after f(), g(), and h() are called.

f()
true
f()
g()
h()
no

6. (10 points) Given the following class definition:

```
class Bar
{
    public:
     Bar(int value);
     ~Bar();
     void print();
    private:
     int m_value;
};
```

Explain each of the following lines of code. Provide complete explanations.

```
Bar *my_bar = NULL;
```

declare a new variable called my_bar that is a pointer to a Bar object
initializes it to NULL

```
my_bar = new Bar(42);
```

instantiate a new Bar object

(1) reserves some memory for the new object
(2) calls Bar::Bar() w/42 as the argument

```
my_bar->print();
```

Call print function for the object pointed to by my_bar

same as (*my_bar).print
the *dereferences the pointer my_bar
the .print() calls the print function for that object

```
delete my_bar;
```

(1) Call the destructor for the object pointed to by my_bar: ~Bar()
(2) free the memory associated with this object so it can be reused

For the following 4 questions, use the code on the last page.

7. (10 points) Write the function `int List::unique_elements()` that returns the number of unique elements in the list. Assume the list **is** sorted and may contain duplicates. For example, if the list contains {1,2,3,4,4,4,4,5,5,5} it contains 5 unique numbers.

```
int List::unique_elements()
{
    int unique = 0;
    Node *ptr = m_head;
    while(ptr != NULL)
    {
        if (!ptr->m_next || ptr->m_value != ptr->m_next->m_value)
        {
            unique++;
        }
        ptr = ptr->m_next;
    }
    return unique;
}
```

8. (15 points) Write the function `bool List::smallest_largest(int &smallest, int &largest)` that finds the smallest and the largest numbers in the list. If the list is empty return false. If the list is not empty put the smallest number in the smallest parameter, the largest number in the largest parameter, and return true. Assume the list is **not** sorted.

```
bool List::smallest_largest(int &smallest, int &largest)
{
    if (m_head == NULL)
    {
        return false; // list is empty, there is no smallest or largest
    }

    // as a starting point assume the first element is both largest and smallest
    // handles special case of only one element in list
    smallest = m_head->m_value;
    largest = m_head->m_value;

    Node *ptr = m_head->m_next;
    while (ptr != NULL)
    {
        if (ptr->m_value < smallest) // found a new smallest element
        {
            smallest = ptr->m_value;
        }
        if (ptr->m_value > largest) // found a new largest element
        {
            largest= ptr->m_value;
        }
        ptr = ptr->m_next;
    }
    return true; // list not empty so there must be a smallest and largest
}
```

9. (15 points) Write the function `bool List::insert_sorted(int value)` that inserts the given value into the list in such a way that the list is ordered from smallest to largest. If the number is already in the list, do not insert it and return false. If the number is not in the list, insert it and return true. Assume this list **is** sorted.

```
bool List::insert_sorted(int value)
{
    // if list is empty or new element belongs at front of list
    if (m_head == NULL || value < m_head->m_value)
    {
       m_head = new Node(value, m_head);
       return true;
    }
    else
    {
        Node *ptr = m_head;
        // look for the node we want to insert after
        // if number already in list, this loop stops before the value
        while (ptr->m_next != NULL && ptr->m_next->m_value < value)
        {
           ptr = ptr->m_next;
        }
        assert(ptr != NULL);
        // check to see if number already in list
        // if number already in list, the above while loop stops before the value
        if (ptr->m_next != NULL && ptr->m_next->m_value == value)
            return false;
        ptr->m_next = new Node(value, ptr->m_next);
        return true;
    }
}
```

10.(20 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. Assume the list is **not** sorted.

```
bool List::delete_value(int target)
{
    // special case, empty 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 may 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;
    }

    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, & 10. You **may not** alter or add to this class definitions.

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

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

class List
{
    public:
     List(){m_head = 0;}
     ~List();
     bool insert_sorted(int value);
     bool delete_value(int target);
     bool smallest_largest(int &smallest, int &largest);
     int unique_elements();
    private:
     Node *m_head;
};
```