First Midterm Exam
• This test contains 9 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 7-9 require you to write code. They are worth 10, 15, and 20 points respectively.
• You have 50 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.

void increment(int *i)
{
    (*i)++;
}
void decrement(int &j)
{
    j--;
}
int main()
{
    int k = 100;
    increment(&k);
    decrement(k);
    cout << k << endl;
}

This code prints 100. Since the address of k is passed to increment(), the (*i)++ increments k to 101. Since the argument to decrement() is a reference parameter, k is decremented to 100.

4. (5 points) What is the difference between a class and an object?

A class is a template for an object, it defines the member functions and the member variables. A class has no memory associated with it.
An object is an instantiation of a class like in “int i” i is an instantiation of an integer. There can be many objects of a single class.
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 shortcircuit 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() true
g() false
h() false
no
6. (10 points) Given the following class definition:

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

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

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

declare a new variable called my_bar that is

a pointer to a Bar object

initializes it to NULL

```cpp
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

```cpp
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

```cpp
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
7. (10 points) Write the function `bool List::is_sorted()` that returns true if the list is sorted (from smallest number to largest number) and false if it is not sorted. Assume that an empty list is sorted.

```cpp
bool List::is_sorted()
{
    // special case that the list is empty
    if (m_head == NULL)
        return true;
    Node *ptr = m_head;
    while (ptr != NULL && ptr->m_next != NULL)
    {
        // if any pair of neighbors in the list are out of order, then the
        // entire list is not sorted
        if (ptr->m_value > ptr->m_next->m_value)
            return false;
        ptr = ptr->m_next;
    }
    // all pairs of neighbors are in order, so the entire list is sorted
    return true;
}
```

8. (15 points) Write the function `void 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. Insert the number into the list even if it is already in the list. Assume this list is sorted when the function is called.

```cpp
void 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);
    }
    else
    {
        Node *ptr = m_head;
        while (ptr->m_next != NULL && ptr->m_next->m_value < value)
        {
            ptr = ptr->m_next;
        }
        assert(ptr != NULL);
        ptr->m_next = new Node(value, ptr->m_next);
    }
}
```
9. (20 points) Write the function `bool List::most_common_number(int &value, int &count)` that searches the list and finds the number that appears the most times. Return false if the list is empty. Return true if the list is not empty. Use the reference parameters to return the most common element and the number of times it appears. If there is a tie (two or more numbers appear the most) return the first. Example: if the list == (1,2,3,3,4,5,5,5,7,9) the most common number is 5 and it appears 3 times.

```cpp
bool List::most_common_number(int &value, int &count)
{
    // fails on empty list
    if (m_head == NULL)
    {
        return false;
    }
    // we have not seen anything yet, so the number of occurrences of value = 0
    count = 0;
    // keep track of the best cur_value and the number of times it has been seen
    int cur_count = 1;
    int cur_value = m_head->m_value;
    for (Node *ptr = m_head->m_next; ptr; ptr = ptr->m_next)
    {
        // if this value is the same as the last, count it
        if (ptr->m_value == cur_value)
        {
            cur_count++;
        }
        // else this value is different then the one we have been counting
        // if it is the largest sequence we have seen so far, keep it
        // start the counting over with the current element
        else
        {
            // if the last element was the most frequent so far, keep it
            if (cur_count > count)
            {
                count = cur_count;
                value = cur_value;
            }
            // start over counting
            cur_value = ptr->m_value;
            cur_count = 1;
        }
    }
    // the list is not empty so we should have counted something
    assert(count != 0);
    // the list was not empty so return true
    return true;
}
```
Use the following class definitions for questions 7, 8, & 9. You may not alter or add to these 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 List::is_sorted();
        void insert_sorted(int value);
        bool most_common_number(int &value, int &count);
    private:
        Node *m_head;
};