CSCI 112 **NAME** _____

October 2, 2008 **Minutes used:** _____

First Midterm Exam

- This test contains 10 questions worth a total of 100 points.
- Questions 1-3 have short answers and are 6 points each.
- Question 4-6 are a bit longer and are worth 8 points.
- Questions 7-10 require you to write code. They are worth 10, 16, 16, and 16 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. (6 points) Give an example of an implicit type conversion. Explain your answer.

```
int i = 42;
double x = i;  // i is converted to a double automatically or "implicitly"
```

2. (6 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. This message include the filename and line number of the assert statement so it is easy to find the problem.

3. (6 points) Given the following class definition:

```
class Dog
{
    public:
        Dog(int age, int weight)
        {m_age = age; m_weight = weight;}
        void print()
        {cout << "dog's age = " << m_age << " weight = " << m_weight <<endl;}
    private:
        int m_age;
        int m_weight;
};
```

Write code to instantiate a Dog object without using new and then call it's print function. Now do the same thing using new.

```
Dog sam(1,21);
sam.print();

Dog *spot = new Dog(11,103);
spot->print();
```

4. (8 points) In the following diagram, each box represents a file with the filename at the top of the box. Add to the code in the boxes the necessary include directives (#include) so the program will compile. Ignore the "..." they are simply a place holder for real code.

list.h

```
#include "node.h"


class List
{
   public:
        List();
        void insert(int value);
   private:
        Node *m_head;
};
```

node.h

```
class Node
{
   public:
        Node(int v, Node *next);
        int m_value;
        Node *m_next;

};
```

list.cpp

```
#include "list.h"


List::List()
{
        ...
}

void List::insert(int value)
{
        ...
}
```

node.cpp

```
#include "node.h"


Node::Node(int v, Node *next)
{
        ...
}
```

main.cpp

```
#include "list.h"


int main()
{
        List the_list;

        the_list.insert(42);
        ...
}
```

5. (8 points) What does the following program print? This question is very tricky, be careful! **You must explain your answers for any credit.**

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

int f(int value)
{
    return value + 1;
}
void g(int &value)
{
    value = value + 1;
}
void h(int *value)
{
    value = value + 1;
}

int main()
{
    int i = 42;

    f(i);
    cout << i << endl;          42, f()'s return value is ignored.  Its argument
                                is passed by value, but that is irrelevant.
    g(i);
    cout << i << endl;          43, g()'s parameter is a reference parameter
                                so when value is change in g(), i is changed.
    h(&i);
    cout << i << endl;          43, h() modifies the local pointer value, but
                                not the value it points to, thus i does not
                                change.
}
```

6. (8 points) What does each line in the following program print?

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

int main()
{
    int i = 42;
    int *ptr = &i;

    cout << &i << endl;         // the address of 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 value of i
}
```

For the following 4 questions, use the code on the last page. Tear off last page for easy reference.

7. (10 points) Write the constructor and the destructor for the List class.

```
List::List()
{
     m_head = NULL;
}

List::~List()
{
    Node *ptr = m_head;
    while (ptr != NULL)
    {
        Node *tmp = ptr;
        ptr = ptr->m_next;
        delete tmp;
    }
}
```

8. (16 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.

```
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;
}

// here is a shorter version of the above code
// this code handles the empty list case (special case code above isn't necessary)
bool List::is_sorted()
{
    for (Node *ptr = m_head; ptr && ptr->m_next; ptr = ptr->m_next)
    {
        if (ptr->m_value > ptr->m_next->m_value)
            return false;
    }
    return true;
}
```

9. (16 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.

```
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);
    }
}
```

10.(16 points) Write a function that removes the last element from the list. Return true if the list is not empty, false if it is empty. Place the value of the node removed in the reference parameter value.

```
bool List::delete_last(int &value)
{
    // special case, empty list
    if (m_head == NULL)
        return false;

    // special case, only one element in list
    // this is special because the value of m_head changes
    if (m_head->m_next == NULL)
    {
        value = m_head->m_value;
        delete m_head;
        m_head = NULL;
        return true;
    }

    // "normal case" more than 1 element in list
    // find the element BEFORE the one we want to delete (before the last element)

    Node *ptr = m_head;
    while (ptr->m_next->m_next != NULL)
    {
        ptr = ptr->m_next;
    }
    assert(ptr->m_next != NULL);
    assert(ptr->m_next->m_next == NULL);

    value = ptr->m_next->m_value;
    // don't need to keep a tmp in this case
    delete ptr->m_next;
    ptr->m_next = NULL;
    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();
     ~List();
     bool is_sorted();
     void insert_sorted(int value);
     bool delete_value(int target);

    private:
     Node *m_head;
};
```