

First Midterm Exam

- This test contains 8 questions worth a total of 80 points.
- Questions 1-4 have short answers and are 5 points each.
- Question 5 is a bit longer and is worth 10 points.
- Questions 6-8 require you to write code. They are worth 15, 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.
- **Do not turn this page until instructed to do so.**
- The test with answers will be posted on the class webpage immediately after the test.
- No electronic devices (music, phone, etc).

Test Taking Advice

- 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.

1. (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 is memory associated with an object. There can be many objects of the same class.

2. (5 points) The following is a legal and working program. What does it print? You must explain your answer for any credit. Be careful, this is a trick question.

```
void find_max(int nums[], int size, int &max)
{
    max = nums[0];
    for (int i = 1; i < size; i++)
    {
        if (nums[i] > max)
            max = nums[i];
    }
}

void find_min(int nums[], int size, int min)
{
    min = nums[0];
    for (int i = 1; i < size; i++)
    {
        if (nums[i] < min)
            min = nums[i];
    }
}

int main()
{
    int values[] = {-42, 17, 34, 42, 99}; // this initialized the array
    int the_max = 0;
    int the_min = 0;

    find_max(values, 5, the_max);
    find_min(values, 5, the_min);

    cout << "max = " << the_max << " min = " << the_min << endl;

    return 0; // normal exit
}
```

max = 99, min = 0

The find_max() function takes a reference to the integer max, so when it puts 99 in max it changes the_max in main(). The find_min() function takes a integer value for min, so when it puts -42 in min that value is NOT put in the_min.

3. (5 points) Write a worthwhile assert statement for function `Foo::set()`. Briefly explain why it is worthwhile.

```
#include <assert.h>

const int SIZE = 42;

class Foo
{
public:
    void set(int index, int value);
private:
    int m_values[SIZE];
};

void Foo::set(int index, int value)
{

    assert(index < SIZE);
    m_values[index] = value;

}
```

The `m_values` array is `SIZE` elements long. It would cause memory problems if an index larger than or equal to `SIZE` was used to set an element.

4. (5 points) In the following code, what job do the lines that start with `#` perform? Explain why this is so important.

```
#ifndef CALENDAR_H
#define CALENDAR_H

class Foo
{
    // class definition goes here
};

#endif
```

They prevent this header file from being included multiple times via the `#include filename` mechanism. Multiplying including header files almost always causes compilation errors.

It would be very hard for a programmer to figure out what files have already been included and thus don't need to be included. This mechanism allows programmers to include the files needed without worrying about any files being included multiple times.

5. (10 points) Given class Foo and the following code, indicate which lines of code in main() cause Foo() and/or ~Foo() to execute. For example, if you think that the line "Foo g;" causes Foo() to be executed, put Foo() on the blank line to the left of "Foo g;"

```
class Foo
{
    public:
        Foo() {cout << "Foo::Foo() called" << endl;}
        ~Foo() {cout << "Foo::~~Foo() called" << endl;}
        void print() {cout << "Foo::print_foo() called" << endl;}
};

void print_foo(Foo *my_foo)
{
    my_foo->print();
}

_____ int main()
_____ {
_Foo()_____ Foo g;           // declaring a new Foo object
_____      Foo *f;
_Foo()_____ f = new Foo();    // dynamically instantiating a Foo object
_Foo()_____ print_foo(new Foo()); // dynamically instantiating a Foo object
_~Foo()_____ delete f;       // deleting a dynamically instantiated Foo
_~Foo()_____ }              // local variable g goes out of scope
```

For the following 3 questions, use the code on the last page.

6. (15 points) Write the function List::find_element() that finds the element with the given index in the list. For example, if the index parameter is 1, the function should find the 1st element in the list. If the index is 2, it should find the 2nd element in the list. If the index is n, it should find the nth element. Return false if the list does not have that many elements.

```
bool List::find_element(int index, int &value)
{
    for (Node *ptr = m_head; ptr; ptr = ptr->m_next)
    {
        if (index == 1)
        {
            value = ptr->m_value;
            return true;
        }
        index--;
    }
    return false;
}
```

7. (15 points) Write the function `List::count_even_numbers()` that returns the number of even numbers in the list. Hint: %.

```
int List::count_even_numbers()
{
    int count = 0;

    for (Node *ptr = m_head; ptr != NULL; ptr = ptr->m_next)
    {
        if (ptr->m_value % 2 == 0)
            count++;
    }
    return count;
}
```

8. (20 points) Write the function `List::insert(int value)` that inserts the given value into the list in such a way that the list is ordered from largest to smallest. Do not insert the number if it is already in the list.

```
void List::insert(int value)
{
    Node *ptr = m_head;

    // check if value already in list
    while (ptr != NULL)
    {
        if (ptr->m_value == value)
            return;
        ptr = ptr->m_next;
    }

    // special case: the new value goes at the front of the list
    if (!m_head || m_head->m_value < value)
    {
        m_head = new Node(value, m_head);
        return;
    }

    // find the node we want to insert after
    ptr = m_head;
    while (ptr->m_next != NULL)
    {
        if (ptr->m_next->m_value < value)
        {
            ptr->m_next = new Node(value, ptr->m_next);
            return;
        }
        ptr = ptr->m_next;
    }

    // special case: the new value goes at end of list
    ptr->m_next = new Node(value, ptr->m_next);
}
```

More compact answer for insert() (question 8)

```
void List::insert(int value)
{
    // check if value already in list
    for (Node *ptr = m_head; ptr; ptr = ptr->m_next)
        if (ptr->m_value == value)
            return;

    // special case: the new element goes at front of list
    if (!m_head || m_head->m_value < value)
    {
        m_head = new Node(value, m_head);
        return;
    }

    // find the node we want to insert after
    Node *ptr;
    for (ptr=m_head; ptr->m_next && ptr->m_next->m_value > value; ptr = ptr->m_next)
        ; // empty for loop
    ptr->m_next = new Node(value, ptr->m_next);
}
```

Use the following class definitions for questions 6,7 & 8 questions. 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();
        void insert(int value);
        bool find_element(int index, int &value);
        int count_even_numbers();
    private:
        Node *m_head;
};
```