433 Midterm Answers, March 9th

1. (15 points - virtual method invocation) What is printed by this program?

```
#include <stream.h>

class A {
public:
  virtual void f() { cout << "A::f()" << endl; }
          void g() { cout << "A::g()" << endl; }
          void h() { cout << "A::h()" << endl;
                     f();
                     g();
                        }
  };

class B : public A {
public:
          void f() { cout << "B::f()" << endl; }
          void g() { cout << "B::g()" << endl; }
  };

void main(int argc, char ** argv) {
  A a;
  a.h();
  B b;
  b.h();
  A * p = new B();
  p->h();
  delete p;
  }
```

Answer:

```
A::h()
A::f()
A::g()
A::h()
B::f()
A::g()
A::h()
B::f()
A::g()
```

2. (15 points - Parameter passing)

```
// C++
void f(Point p, Point q) {
  p.x = 42;
  p = q;
  }
```

Local copies of p and q are constructed as a result of invoking the function. The changes made in f only effect the local copies.

```
// C++
void g(Point & p, Point & q) {
  p.x = 42;
  p = q;
  }
```

The arguments are passed by reference. The x field of the first argument is changed to be 42, and then operator= is invoked on the first argument with the second argument as a parameter.

```
// C++
void h(Point * p, Point * q) {
  p->x = 42;
  p = q;
  }
```

Arguments are passed by pointer. The x field of the first argument is changed to be 42. The assignment p = q only effects the local variables used to hold the arguments, and is not visable outside the function (the assignment is, in fact, dead code that would be eliminated by an optimizing compiler).

```
// Java
void i(Point p, Point q) {
  p.x = 42;
  p = q;
  }
```

Same as the C++ pointer example.

3. (15 points) Say you have class A, and class B being a subtype of A. In C++, how are arrays of A and arrays of B handled? In particular, can you pass one to a function that expects the other? Describe how it works and any issues or problems.

   Answer: In C++ you can pass an array of B where an array of A is expected, because an array of X is just treated as a pointer to X, and a pointer to B is a subtype of pointer to A. However, if B is larger than A, horrible things will happen, because when it indexes into array, it will use the wrong size.

   What about Java? **Answer:** In Java, it will work. However, whenever a store is made into the array, there is a runtype check that the type of the element being stored is a subtype of the run-time element type of the array. For example, if you passed an array of B to a function that expected an array of A, and then stored an object of type A into the first element of the array, it would be OK at compile time, but would generate a run-time exception.

   Back to C++: what if you had an array of pointers to A and an array of pointers to B. Should this work? (I don't expect you to have the C++ standard memorized, so I don't expect you to tell me what the standard says)? Why or why not? What are the issues?

   Answer: In C++, you can't pass an array of pointers to B to someone who expects an array of pointers to A, because **B is not a subtype of **A. If you did, you would have to worry about someone storing a pointer to an A into the array, as above. However, since all pointers are the same size, at least that problem would go away.

4. (25 points – C++ design) Consider the skeleton classes given below. Define appropriate "hidden" functions for these classes (e.g., copy constructor, ...).

   Solution The main design decision that must be made is what happens when operator= is used to update a DisplayedColorText with an DisplayedText. In my solution, I have made it a run-time error.

```
#include <stream.h>

class AssignmentException {};

class DisplayedText {
public:
  DisplayedText(char * buf, int p) : pos(p) {
        txt = new char[1+strlen(buf)];
        strcpy(txt,buf);
        }
 DisplayedText(const DisplayedText & that) : pos(that.pos) {
        txt = new char[1+strlen(that.txt)];
        strcpy(txt,that.txt);
        }

 // make it virtual
 virtual DisplayedText & operator=(const DisplayedText & that) {
        // check for self assignment
        if (this == &that) return *this;
        pos = that.pos;
        delete [] txt;
        txt = new char[1+strlen(that.txt)];
        strcpy(txt,that.txt);
        return *this;
        }
  // make it virtual
  virtual ~DisplayedText() {
```

```
        delete [] txt;
        }

private:
  char * txt; // ref to malloc'd, null-terminated string
  int pos; // position text is to be displayed at
  };

class DisplayedColorText : public DisplayedText {
public:
  DisplayedColorText(char *buf, int p, int c) : DisplayedText(buf, p),
                                clr(c) {};

 // default copy constructor and destructor will chain appropriately

 // covariant return types allowed (and appropriate),
 // but argument types must match exactly in order to override
 virtual DisplayedColorText & operator=(const DisplayedText &that) {
        const DisplayedColorText * p
                = dynamic_cast<const DisplayedColorText *>(&that);
        if (p == 0) {
                cerr << "Can't assign DisplayedText to DisplayedColorText";
                throw AssignmentException();
                };
        DisplayedText::operator=(that);
        clr = p->clr;
        return *this;
        }
private:
  int clr; // color of text
  };
```

5. (15 points) What are the possible results of this code fragment? What is the effect of the synchronization?

```
Point a = new Point(1,2);
Point b = new Point(3,4);

Threat t1 = new Thread() {
    public void run() {
        synchronized(a) {
            a.x = b.x;
            a.y = b.y;
            }
        }};
Threat t2 = new Thread() {
    public void run() {
        synchronized(b) {
            b.y = 40;
            b.x = 30;
            }
        }};
t1.start(); t2.start();
```

```
t1.join(); t2.join();
System.out.println("a = " + a);
System.out.println("b = " + b);
```

Answer: The synchronization has no effect, because the threads obtain locks on different objects. b will always be equals to [30, 40]. The point a could be: [3, 4], [30, 4], [3, 40] or [30, 40].

6. (15 points) Consider the following Java function:

```
static int f(int i, int j) {
    while (true) {
        try {
            if (i <= 0) throw new IllegalArgumentException();
            if (i == 1) return 42;
            i--;
        }
        finally {
            if (j == 0) {
                j++;
                continue;
            }
            if (j < 0) throw new IllegalStateException();
            if (j > 10) return 17;
        }
    }
}
```

What does this function do for each of the following sets of arguments?

f(-1, -1)	f(-1, 0)	f(-1, 5)	f(-1, 20)
f(1, -1)	f(1, 0)	f(1, 5)	f(1, 20)

The function returns:

f(-1, -1) : IllegalStateException	f(-1, 0) : IllegalArgumentException
f(-1, 5) : IllegalArgumentException	f(-1, 20) : 17
f(1, -1) : IllegalStateException	f(1, 0) : 42
f(1, 5) : 42	f(1, 20) : 17