1. What are the values of `x`, `y`, and `z` when this code is finished executing?

```
int x, y, z;

x=12; y=9; z=23;
x = y++ + ++z;       y = z / x;
if ( y > x )
   z = 119;
```

First, `x` gets 12, `y` gets 9, and `z` gets 23.

Then we have to evaluate `x = y++ + ++z;`. The value of `y++` is 9, and the compiler remembers to bump up the value of `y` by 1 sometime before the end of the statement. The value of `++z` is 24, which is what `z` will be after the compiler bumps up the value of `z` by 1, which it must also do before the end of the statement. So `x` gets $9 + 24$, which is 33. That's the end of the statement, so now `y` and `z` have both been bumped, and `y` is 10 and `z` is 24.

The next statement is `y = z / x;`. `z` is 24 and `x` is 33, and they're both `<int>`s, so `/` means integer division, which means we discard the fractional part of the result. So `y` gets 0.

Now `y > x` is false, so the computer skips the whole `if` statement. The final result is: `x`: 33, `y`: 0, `z`: 24. These were worth a point each.

2. What are the values of `x`, `y`, and `z` when this code is finished executing? (Caution! This is a trick question.)

```
int x, y, z;

x=12; y=9; z=23;
if ( x = y )
   z *= 2;
```

This is a trick question because the condition in the `if` clause is `x = y`, and *not* `x == y`. The expression `x == y` compares `x` and `y`, yielding true if they're equal and false otherwise. The expression `x = y`, on the other hand, assigns the value of `y` to the variable `x`, and its value is whatever value got assigned to `x`—in this case, 9. So after the condition in the `if` clause is evaluated, `x` is 9, `y` is 9, and `z` is 23.

Now, a condition is 'true' when its value is not zero, and the value of this condition is 9. Therefore the statement `z *= 2;` is executed. This assigns the value 46 to `z`. So the final score is: `x`: 9, `y`: 9, `z`: 46. `y` was easy and was worth a point; the values of the other two depended on you knowing realizing what the `=` was doing and were worth two points each.

3. What does this print? And are are the values of `x` and `y` when it is finished?

```
int x, y

x=7; y=9;
if ( --x > 6 &&  y++ > 8 ) printf("Foo.\n");
```

```
else printf("Bar.\n");
```

The thing to remember here is that `&&` short-circuits. That means that the compiler evaluates the left-hand part of the `&&` expression first, and only evaluates the right-hand part if it needs to. The value of `--x` is 6, and the compiler remembers that it must decrement `x` before the end of the statement. Since `6 > 6` is false, the compiler knows that whatever's on the right of `&&` is irrelevant—the whole expression will be false no matter what. So it never evaluates the `y++ > 8` part, and in particular it doesn't evaluate `y++`, so `y` never gets bumped.

The condition was false, so the computer jumps to the `else` clause and prints `Bar.`. `x` got decremented by this time, so `x` is now 6. `y` never got incremented at all, so `y` is still 9.

Getting the `Bar.` and the value of `x` were worth a point apiece; to get the value of `y` you had to remember that `&&` short-circuited and so that was worth two points.

4. On the back of this sheet, write one sentence about each of three things that the preprocessor does.

The three things I was looking for in particular were:

- The preprocessor removes comments, which are any text between and including the sequences `/*` and `*/`.
- The preprocessor substitutes the appropriate text for the names of manifest constants that have been defined with the `#define` directive.
- The preprocessor incorporates the text of files included with the `#include` directive into the input seen by the compiler.

Most people got at least two of these. They were worth a point each.