HW #3 – DUE Monday 1/9

1. Suppose you create a local variable like this:
   
   ```
   string *x = new string(“hello”);
   ```
   
   A. Did you create an object?
   B. What happens on the stack?
   C. What happens on the heap?

2. What must you eventually remember to do if you execute the statement suggested in the previous question?

3. Show how you would allocate an array of 6 empty string objects on the heap.

4. Show how you would de-allocate that memory when you are finished using the array from the previous question.

5. TRUE/FALSE A const reference can refer to non-const data.

6. TRUE/FALSE A non-const reference can refer to const data.

7. Assume y is a (non-const) int variable. Write “yes” or “no” in each box:

<table>
<thead>
<tr>
<th>Pointer x can be moved to point elsewhere</th>
<th>int * const x = &amp;y;</th>
<th>int const * x = &amp;y;</th>
<th>int const * const x = &amp;y;</th>
</tr>
</thead>
<tbody>
<tr>
<td>y can be modified using x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8. Consider the following function prototypes. Write “yes” or “no” in each box:

<table>
<thead>
<tr>
<th>Makes a copy of the argument</th>
<th>void f(Cat c)</th>
<th>void f(Cat *c)</th>
<th>void f(Cat &amp;c)</th>
<th>void f(Cat const &amp; c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has the ability to modify the argument (the caller’s original Cat)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9. Assume you have written a function with the prototype below:

   ```
   void foo(int &x)
   ```

Which of the following function calls are allowed?

   a. foo(7);
   b. int y = 12;
      foo(y);
   c. const int y = 12;
      foo(y);

[Continued on the following page…]
10. Consider running this program:

    Cat foo(Cat x)
    {
        Cat &q = x;
        Cat &r = q;
        Cat &s = r;
        Cat &t = s;
        Cat &u = t;
        Cat &v = u;
        return v;
    }

    int main()
    {
        Cat y, z, *a, *b, **c, **d;
        z = foo(y);
        a = &y;
        b = &z;
        c = &a;
        d = &b;
        return 0;
    }

a. How many Cat objects are **instantiated** during the running of this program?
b. How many times is the state of one cat object **copied** over to a **different** Cat object as this program runs?