Introduction to C++ - Quiz 1, 13 Feb 2012

Name: _____ ID: _____

This is a *closed book* quiz. There are seven questions on three pages, for a total of 20 possible points.

In the code fragments, assume that the context (e.g. appropriate header files and `using namespace` statements) has been specified correctly.

In questions that ask what a fragment would print, in each case your answer should consist of one or more decimal digits (don't worry about spaces or newlines).

1. What would the following code fragment print (4 pts)? _____

```
int a[] = { 10, 4, 20 };
int* q = a + 1;
int& r = *(q+1);
*q = 3;
r = 6;
cout << a[0] << " " <<  a[1] << " " << a[2]  << endl;
```

It prints 10 3 6
int *q = a+1 means declare a pointer which will store the address of a+1 (where 4 is now)
int&r = *(q+1) means r should be an alias for the data point one past q, where 6 is
*q = 3 changes data at q to be 3.
r=6 changes the data at q+1 to be 6

2. How could you correctly call the function swap below to swap the values stored in the variables a and b (2 pts)?

```
void swap(int* x, int* y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main()
{
     int a = 1, b = 2;
    swap(&a,&b);
    return 0; //full credit if you omitted this as well
}
```

3. Rewrite the above code snippet (both the swap function and the main function) so that the function swap takes as input two *references* to an `int` instead of 2 pointers to ints. (3 points)

```
void swap(int& x, int&y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

```
int main()
{
    int a = 1, b = 2;
    swap(a,b);
    return 0;
}
```

4. Suppose I have a struct defined as follows:

```
struct DoubleNode
{
     double value;
     DoubleNode* next;
};
```

The following function is supposed to create a new DoubleNode and return a pointer to it. The new double node's value property should be set to the double passed as input and the next property should be set to NULL

```
DoubleNode* createElement(double value)
{
     DoubleNode newElement;
     newElement.value = value;
     newElement.next = NULL;
     return &newElement;
}
```

The above code does compile but it does not obey proper memory management techniques. In fact, when you compile the code, you get the following warning:

```
warning: address of local variable 'newElement' returned
```

Explain in one or two sentences why it is problematic to return the address of a local variable. (2 points)

When you create a variable, you are reserving and labeling a certain space in memory. Once a function leaves, any memory reserved for local memories is released. This means tha the data at that address may not have any meaning, so returning the address of that elemen has no purpose.

5. Rewrite the above function so that it correctly satisfies proper memory management techniques (and still creates a new DoubleNode with appropriate values) (3 points)

```
DoubleNode* createElement(double value)
{
    DoubleNode* newElement = new DoubleNode;
    newElement->value = value;
    newElement->next = NULL;
    return newElement;
}
```

6. Write a function `copy` to take as input three things: A start iterator of a collection, the location one past the end of the same collection (i.e. an end iterator), and a start iterator of a second collection. You may assume for simplicity that the two collections have identical sizes.

   Your function should copy the contents from the first collection into the second collection. You may assume that the 2nd start iterator is a write-able iterator and that both iterators have the `++` operator defined on them. You may also assume that enough space exists to be able to write every value into the destination collection. (4 points)

   Hint: Remember that to allow a function to work with *any* kind of iterator, you can add the following line before your function. You may then use `InputIterator` in place of the actual type.

   ```
   template<class InputIterator>
   ```

   Since the two kinds of iterators are different (one could be a vector iterator and the other a list iterator for example), you'll need to have two different types in this command.

   ```
   template<class InputIterator, class OutputIterator>
   void copy(InputIterator start1, InputIterator end1, OutputIterator start2)
   {
        for (InputIterator current = start1; current != end1; current++)
        {
             *current = *start2;
             start2++; //could also move into for loop header using comma
        }
   }
   ```

7. Explain in one or two sentences why we are able to combine multiple « operators into one command such at `cout « "x" « "y"` and the program still type checks. (2 points)

   ```
   The << operator is overloaded to act on a stream and a string
   and return a stream. This means that if you operate on the cout object and
   pass the string x to it, the returned value of the expression cout << "x"
   is the cout object as well,
    allowing it to be strung together so at the next step we evaluate cout << "y"

   Note the difference between something like  0 < x < 2 to check if x is between 0 and 2.
   In that case,  0 < x returns a boolean.
   It doesn't make sense to check whether a boolean is less than 2 or not.
   ```