**Recursion Problems**

For each problem below, write two different functions to solve the problem. The first function should solve the problem using a loop, the second should use recursion (without a loop). **When solving a problem recursively, write out the base case and recursive case in words or pseudocode first. Make sure that for every reasonable input, the recursive case will eventually reduce down to the base case.**

1. Write a function to convert a string to uppercase. The function should not change the original string, but rather return a new string that is the original string converted to all caps. For this, #include <cctype> and use toupper to convert an individual character to uppercase.

   `string all_caps_iter(const string & s)` [ use this definition line for the iterative version ]

   `string all_caps_rec(const string & s)` [ use this definition line for the recursive version ]

2. Write a function to test if a string is a palindrome (reads the same forwards and backwards).

   ```
   bool is_pal_iter(const string & s)

   bool is_pal_rec(const string & s)
   ```

3. Write a function to compute a^b, (a raised to the b power) where a and b are integers.

   ```
   int power_iter(int a, int b)

   int power_rec(int a, int b)
   ```

4. Write a function to find the maximum element in a vector of integers. You've done this with a loop many times, so don't worry about writing the iterative version of this, just write the recursive version.

   Hint: The "natural" recursive formulation involves calling the function recursively on slices of the original list that continue getting smaller:

   **Base case**: the maximum element in a vector of size 1 is the lone element in the vector; that is, vec[0]
   **Recursive case**: the maximum element in a vector of size > 1 is the larger of vec[0] and the maximum element in vec[1:size] *(using Python slice syntax here)*

   Because in C++ there is no convenient "slice" operator like there is in Python, we must be a little clever to work around this. Instead of slicing the list over and over, create a "helper" function that takes two arguments: the vector of integers and a "starting position" for the vector. The helper function will actually do all the recursion, and you should use the starting position parameter to keep track of what fraction of the list you've already looked at; that is, as you recursive, the "starting position" parameter will keep growing larger and larger.

   ```
   int maximum_rec(const vector<int> & vec)
   ```