# Lab 12

## Recursion

Skills Required

* Recursion
* Function design

**Assignment**

If a subproblem is the same type as the original then recursion may be a good solution. A recursive function is one that calls itself. You must create RECURSIVE solutions to the functions given. As you are working you may want to comment out some of the main program and inject some code to allow you to test your solution rather than waiting until the end. When thinking about recursive functions it helps to think about the 2 cases. The first case to consider is the base case. When will your function be done? If it calls itself each time then it will be an infinite call, so there needs to be the exit condition to consider. The second case is what subproblem can call the function again.

All functions should have declarations and definitions. You should also document PRE and POST conditions for the functions. Creating functions that are correct requires RECURSION. Simply solving the problem will not get points.

## Functions

### long factorial\_r(long value);

This function returns an integer that is the factorial of the value passed to it. The base case is when we value is 0 and the factorial of 0! is 1. Recursively . Your function can assume that the integer passed in is >= 0. Note that in testing, you should only use relatively small values for the parameter, as the factorial function grows exponentially fast. If you call this function repeatedly with successive values, and suddenly get a large negative number as an answer, that’s what’s happening—the value is ‘overflowing’ into the leftmost bit, which is taken as the sign of the number.

int fibonacci\_r(int n);

This function returns an integer the fibonacci value for the term. The Fibonacci sequence is 0, 1, 1, 2, 3, 5, 8, … Each value is the sum of the last two. Therefore, there are 2 base cases, where n = 0, the result is 0, and when n = 1 the result is 1. Recursively if n is not one of those then the result is the sum of the previous 2 values of Fibonacci\_r(n).

bool is\_prime\_r(int value, int n);

A number is prime if it is not divisible by any other number except one and itself. Calling this function, value will not change, but the value for n will. The base case for this function is if n is larger than the sqrt of value in which case it must be a prime. The other base case for this function is if value is divisible by n exactly; that is, if value % n is 0. If so, then value is not a prime number. Recursively: if value and n are not a base case, then return the result for is\_prime\_r with value and the next value of n. The initial call should be is\_prime\_r(value, 2);

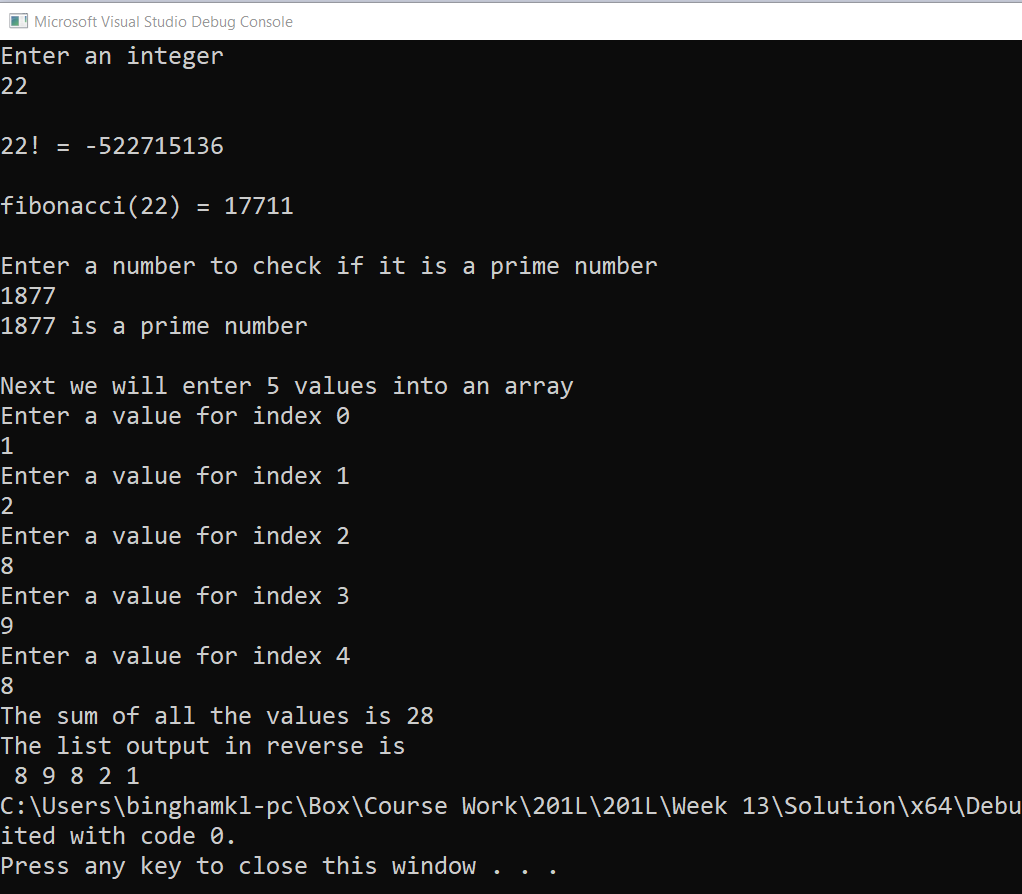
int sum\_r(int list[], int list\_size);

This function recursively calculates the sum of the list. If you have a list then the sum can be thought of as the last item in the array added together with the sum of the all the previous items. The base case is when list\_size = 0, in which case the sum of an empty list is 0. Otherwise, add the last element in the list with the result of sum\_r with the list and an index size of one less.

void output\_reverse\_list\_r(int list[], int list\_size);

This function recursively outputs the array in reverse.

### Example



**Submit your assignment**

Update your files on GitHub, or clean up the Debug directory, and upload all of the files.