Shawn S HIllyer

CS 162-400

November 27, 2015

Lab 10 – Analysis of Recursive, Iterative, and Tail-recursive functions

Predictions

Fibonacci Function

I have a few predictions about the run time performance of the recursive Fibonacci vs. the iterative solution. I know from prior experience with recursion in C++ and C# that the recursive functions can cause a lot of overhead. Each function call causes a new pointer to the return location to be placed on the stack. Also, each variable that is passed needs to be accounted for – declared, allocated, and dealt with when the function returns to the call down one layer on the stack. All of these actions take up processing time.

So, my obvious prediction is that, with large enough values of n, the recursive Fibonacci will take up a lot more run time than the iterative solution.

Factorial Function

I also have a little bit of prior knowledge about tail recursion. I know that some compilers are capable of turning a tail recursive function into a loop (iterative) function behind the scenes. They often (always?) skip this optimization during a debug build, so I've heard it's important to ensure that you rewrite the function iteratively if it's important for this optimization to take place for performance or other run-time reasons. For example, if the function must be optimized in order to avoid stack overflows during use, then you need to fix it before-hand.

Anyways, my expectation is that the optimized tail-recursion function will run faster than the non-tail version because, behind the scenes, the tail-recursion becomes an iterative process in the executable.

Results

Fibonacci Function

The results matched my expectations in this case. Here is the output file from my Fibonacci tests:

```
14:06:38. Click Time: 46(Start:    fib_rec())
14:06:47. Click Time: 9281(End:  fib_rec())
14:06:47. Click Time: 9281(Call:  fib_iter())
14:06:47. Click Time: 9281(End:  fib_iter())
```

As you can see, the recursive solution executed in $9281 - 46 = 9{,}235$ "clicks", or 9.235 seconds. I found similar differences in the 30 to mid-40's range of n inputs, getting larger as n increases.

The iterative function call takes no discernable time, as shown by the time stamps of 9281 and 9281.

Factorial

I'm not sure if the compiler is somehow able to optimize both the tail recursive and non tail recursive Factorial functions. According to every resource I found online, the optimization should require that I call a command line argument during the compile command of –O2 or –O3 (presumably for "optimize level 2" etc.).

However, there is no discernible runtime for either function calling the function once each. This holds true whether or not I compile with the optimization flag.

```
14:06:23. Click Time: 46(Start:    rfactorial())
14:06:23. Click Time: 46(End:      rfactorial())
14:06:23. Click Time: 46(Call:     factorial())
14:06:23. Click Time: 46(End:      factorial())
```

I decided to start and stop the time stamp for this function before and after a loop of 10000 function calls to see if I could detect a difference:

```
14:58:08. Click Time: 15(Start:    rfactorial())
14:58:08. Click Time: 15(End:      rfactorial())
14:58:08. Click Time: 15(Call:     factorial())
14:58:08. Click Time: 15(End:      factorial())
```

As you can see, however, the timestamps are all the same, which means all 20000 iterations completed in less than a millisecond!

I even modified the example of the non-tail recursive function to make the recursive call very explicitly not at the end of the method:

```
long TestSuite::rfactorial(int n)
// static
{
  if (n == 1)
    return 1;
  else {
    int result = rfactorial(n -1);
    return n * result;
  }
}
```

So, I'm a little surprised by the results! For some reason, the factorial calculations involving multiplication are executing quickly (almost instantaneously) even when I call n with a value of 1,000 (which obviously is giving erroneous factorials, but that's not the point). I know the recursion is working because I checked the values from 1-16 for n against a table of the Factorials online and the results are correct up to n=16.