CS162 – Module 4

Daniel Beyer

Dan.beyer@gmail.com

06/02/16

Analysis of Recursive, Iterative, and Tail-Recursive Functions

Predictions

Fibonacci Function:
I predict that the recursive function will consume far more run time than the iterative function when calculating Fibonacci numbers. This is because recursive functions of this nature are much more complex in relation to variables. Each variable passed needs to be accounted for as the function moves down the stack. This causes increased run time because each function call results in a pointer to the return location being placed on the stack.

Factorial Function:
I predict the optimized tail-recursion factorial function will run faster than the non-tail function because optimization allows the compiler to transform the tail-recursion function into an iterative loop. This will consume less run time and so therefore fun faster.

Results

Fibonacci Function:

Variable Number	Recursive Fibonacci Function Time (s)	Non-Recursive Fibonacci Function Time (s)
20	0.00	0.00
30	0.02	0.00
32	0.06	0.00
34	0.15	0.00
38	0.55	0.00
40	1.92	0.00
50	178.449997	0.00

Factorial Function:

Variable Number	Tail Recursion Time (s)	Non-Tail Recursion Time (s)
20	0.00	0.00
100	0.00	0.00
1000	0.00	0.00
12000	0.00	0.00
20000	0.00	0.00
50000	0.00	0.00

Discussion

The results from the Fibonacci function tests confirmed my predictions. The recursive Fibonacci functions took noticeable time to execute, and this run time drastically increases as the variable number increases. The Non-recursive Fibonacci function took no discernable time based on my timing method, indicating that is a very efficient function.

For the factorial functions, I found no discernable time for either the Tail Recursion function or the Non-Tail Recursion function. Also, the addition of the command –O2 to optimize the compiler for tail recursion did not seem to make a noticeable difference either. When performing my tests without –O2 added to my makefile, there was no noticeable run time noted. Additionally, I know the factorial calculations are working because I display the factorial number correctly up to variable = 16 (beyond which is past the signed limit of C++ and so displays incorrect numbers). I was surprised to find no discernable run time for these functions, and it appears these factorial calculations are executing very quickly and efficiently.