Matthew Vaysfeld
I pledge my honor that I have abided by the Stevens Honor System.

Explanation:

Note: My taylor series starts at 1, so if there is 1 term then the answer is just 1.0

X is the number that you plug into the approximation
I is the number of terms

I divided my code into multiple functions, the main taylor series function, a sum function that would add up each ith term and would get the answer (in d0) , a function that got the ith term, a function that got the power of x to the current i, a factorial function that got the factorial of the current i, and a print function.
The general logic that the code followed was that it initializes the stack, then branches to the sum function, which then calls the ith term function to get the ith term of the current iteration of i, and then adds the ith term to d0 and decrements i. In the ith term function I also call the factorial and power function to get the answer for the ith term, and then multiply them together. At the end of the sum function (when i is 0) I branch to the print, which prints the approximation of the taylor series at x with i terms.

Each function in detail:
Main/taylor: initializes stack and branches to sum

```
 main:
 .global taylor
sub sp, sp, #16
str x30, [sp]

taylor:
ldr x1, =i //address of &i
ldr x1, [x1] // derefrence
ldr x2, =x //address of &i
ldr d2, [x2] // derefrence
mov x3, #1
SCVTF d3, x3
b sum
```

Print: prints the answer (in d0) reloads the stack and returns to caller

```
print:
ldr x0, =string
str d0, [sp,#8]
bl printf
ldr d0, [sp,#8]
ldr x30, [sp]
add sp, sp, #16
br x30 //return to caller
```

Sum: keeps adding the ith term for the current i, keeps track of the sum in d0, and then decrements every loop until i is 0 where it goes to the print function.

```
sum:

sumfunc:
cmp x1, #0 // compares i to 0
beq sumend  // if i is 0 then go back to caller
sub x1, x1, #1 //decrement global i
fadd d0, d0, d3 // adds ith term to total
str d0, [sp,#8]
b ith //get ith term
b sumfunc


sumend:
b print
```

Ith term: find the term for the current i by calling the factorial function on i, the power function on x and i, and then multiplying 1/factorial(i) and x^i to get the ith term and store it in d3.

```
ith:
b factorial // do factorial of i (store fact(i) into x5)
i1:
b power //get power of x to i (store power(x,i) into d11)
i2:
mov x15, #1
SCVTF d15, x15
SCVTF d5, x5
fdiv d10, d15, d5 // 1/fact(i)
fmul d3, d10, d4 // 1/fact(i) * x^i
b sumfunc //go back to sum function
```

Power function: finds the power of x to the i by multiply x by itself i times and storing it in d4

```
power:
mov x12, x1 // temp variable that equals i
mov x11, #1
SCVTF d4, x11

powerfunc:
cmp x12, #0
beq powend
fmul d4, d4, d2
sub x12, x12, #1
b powerfunc


powend:
b i2
```

Factorial: I decided on an iterative approach to finding the factorial of i.

```
factorial:
mov x5, #1
cmp x1, #0
beq factend
mov x13, x1

factfunc:
udiv x14, x1, x13
cmp x14, x1
beq factend
mul x5, x5, x13
sub x13, x13, #1
b factfunc


factend:
b i1
```

Combining all of these functions allows me to iteratively get the answer for the approximation of the taylor series plugging in x for i terms.

Debugger Print for (x=44, i=12):