Jack Byrne
I pledge my honor that I have abided by the Stevens Honor System.

- First, the program initializes by loading in every double. It also calculates the opposite of epsilon so that the program can ensure the calculations are within the proper range.
- At the beginning of each main loop,the midpoint is calculated and the function that calculates the value at f(midpoint) is called. The result is returned into d15.
- The “funfunction” function calculates f(x). It does with two loops, the outer summing the powered x values multiplied by the coefficients. The inner loop finds the powered x values.
- The program checks if the root was found at the midpoint and accordingly exits
- Otherwise, it checks whether the root would be in either to the left or right of the midpoint and changes a and b accordingly.
- In the exit script, it prints both the x value of the root estimation and the value at that root

```
.text
.global _start
.extern printf


_start:
        .global main
main:
        ldr x1, =N //loads N
        ldr x4, [x1]
        ADR x0, neg_one
        LDUR d4, [x0] //-1 for opposite epsilon
        ADR x0, epsilon
        LDUR d5, [x0] //ep
        ADR x0, epsilon
        LDUR d5, [x0] //ep
        ADR x0, a
        LDUR d6, [x0] //a
        ADR x0, b
        LDUR d7, [x0]           //b
        ADR x0, two
        ldur d2, [x0] //constant 2
        ADR x0, zero
```

```
        ldur d8, [x0] //0
        fmov d9, d8 // c
        fmov d10, d8 // input for func
        fmov d13, d8 //stores b-a for loop check
        fmov d14, d8 //temp var for addition
        fmov d17, d8 //temp var 2 for addition
        fmov d18, d8 //temp for storing result
        fmov d19, d8 //init d19
        fmul d19, d5, d4 //opposite of epsilon




        //d6: a | d7: b | d8: zero | d5: epsilon | d13: temp var
        //d14: temp var | d9: c | d2: 2 | d10: input x | d15: sum var
        //d16: square product temp var | d17: temp sum  2 |d18 store result
main_loop:


        fadd d14, d6, d7
        fdiv d9, d14, d2 //c = (a+b)/2

        //test if f(c) is 0
        fmov d10, d9 //input x must be in d10
        bl funfunction

        fcmp d15, d8
        fmov d18, d15 //store current c in d18
        b.eq exit //if f(c) is a root then exit

        fmov d10, d6
        bl funfunction //find f(a) put in d15

        fmul d16, d15, d18 //if f(c)*f(a) < 0
        fcmp d16, d8
        b.ge else
        fmov d7, d9
        b loop_check
else:
        fmov d6, d9
loop_check:
        fsub d13, d7, d6
        fcmp d13, d5 // b-a >= epsilon?
        b.ge main_loop
        fcmp d18, d19 // <= negative epsilon?
        b.lt main_loop
        b exit
exit:
        fmov d0, d9
        ldr x0, =found //print
```

```
        bl printf

        fmov d0, d18
        ldr x0, =actual //print
        bl printf

        //test
        //mov x1, x4
        //ldr x0, =actual //print
        //bl printf

        mov x0, #0
        mov w8, #93
        svc #0

.func funfunction
funfunction:
        //x is in d10
        // use d16 to temp store x
        //degree defined by n in x4

        mov x5, #0 //set counter i to N
        fmov d15, d8 //variable for keeping func sum
        ldr x1, =coeff //loads address of coeff into reg 1
        ldr d1, [x1] //load first coeff
        fadd d15, d15, d1 // add the constant

oloop:
        add x1, x1, #8 //add offset
        ldr d1, [x1] //load coeff
        fmov d17, d8 //zero temp sum var

        mov x6, x5 //set j to i. first j at 1
        fmov d16, d10 //reset d16
iloop:
        cmp x6, #0 // if j = 0 finish loop
        b.eq loopexit
        fmul d16, d16, d10 //square input x
        sub x6, x6, #1 //j--
        b iloop
loopexit:
        add x5, x5, #1 //i++

        fmul d17, d16, d1
        fadd d15, d15, d17


        cmp x5, x4 //if i > n
        b.le oloop
        br x30 //result in d15
```

```
.endfunc

.data
found:
        .ascii "Root esitmated at x = %f\n\0"
actual:
        .ascii "Actual value at f(x) = %f\n\0"

coeff:
        .double 0.2, 3.1, 0.3, 1.9, 0.2
N:
        .dword 4
epsilon:
        .double .01
a:
        .double -1
b:
        .double 2
zero:
        .double 0.0
neg_one:
        .double -1
two:
        .double 2.0
.bss
        .align 8
```