ECE 331 Lab 1: Implementing a sorting algorithm for Primes and Composites in Arm Assembly

Code:

(All of it is on a GitHub repo, it is private so no one uses my code while the project is live)

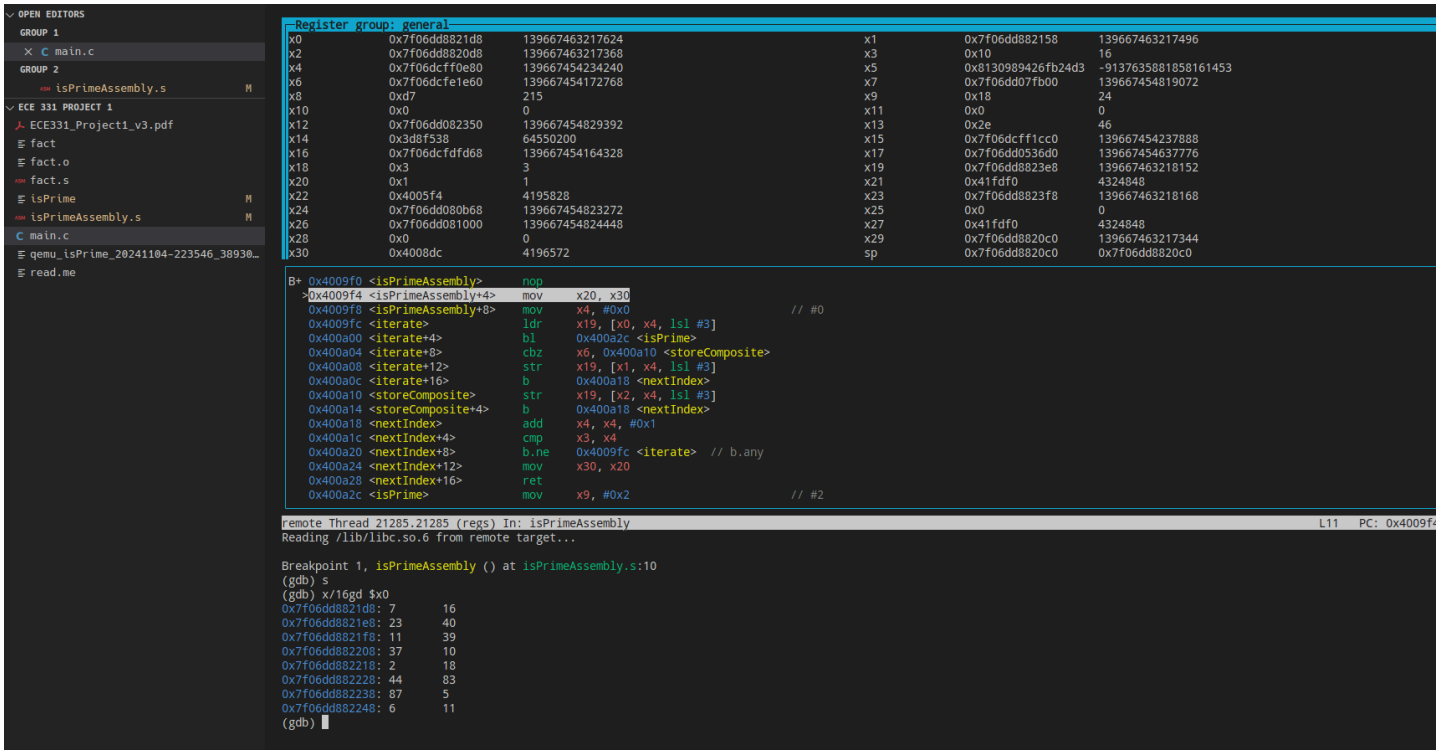
```
	.globl isPrimeAssembly

isPrimeAssembly: // Same as primeIterator()
	// Your code for iterating through arrays here
	// The Base addresses of arrays in X0 - X2, length in X3s
	// X0 has the base address of Original array
	// X1 has the base address of Prime array
	// X2 has the base address of Composite array
	// X3 has the length of the Original array
	nop    //first line of execution when called by main.c
	mov x20, x30 // Save the return register for later
	mov x4, #0 // Initialize as x4=index=0

iterate:
	ldr x19, [x0, x4, lsl #3] // Load value from original array (x0), using lsl to use a
multiple of 8
	// x19 is going to be our n value
	bl isPrime // Call isPrime sub-function, branch to label

	cbz x6, storeComposite // If the returnZero label is used and X6 is 0
	str x19, [x1, x4, LSL #3] // Else store x19 into the Prime array at the current index
	b nextIndex // Branch to calculate next index

storeComposite:
	str x19, [x2, x4, lsl #3] // Store x19 into the Composite array at the current index
	b nextIndex // Branch to calculate next index

nextIndex:
	add x4, x4, #1 // Index=x4+=1, increase index by 1
	cmp x3, x4 // Compare length of original array to altered x4 index (Index that is not a
multiple of 8)
	b.ne iterate // Continue iterating until all array values are accessed
	mov x30, x20 // Reload value of branch to x30
	ret // Return to C code


isPrime: // Same as isPrime()
	// Your code for detecting a prime number here
	mov x9, #2 // Set x9=i=2
	lsr x10, x19, #1 // Set x10=n/2 using logical shift right by 1

loopOne:
	cmp x9, x10 // Compare i with n/2 and raise flags
	bgt returnOne // If i > n/2, bgt with register equal to 1

	udiv x11, x19, x9 // This line computes x11 = quotient = n / i
	msub x12, x11, x9, x19 // This line computes x12 = x19 - x11 * x9 = n - q*i
	cbz x12, returnZero // If x12 reaches 0, go to returnZero label
```

```
        add x9, x9, #1 // x9=i+=1, increase i by 1
        b loopOne // Go back to loopOne label to keep calculating

returnOne:
        mov x6, #1 // Set x6 to 1 for storing into array
        br x30 // Branch to last link to store a Prime

returnZero:
        mov x6, #0 // Set x6 to 0 for storing into array
        br x30 // Branch to last link to store a Composite
```

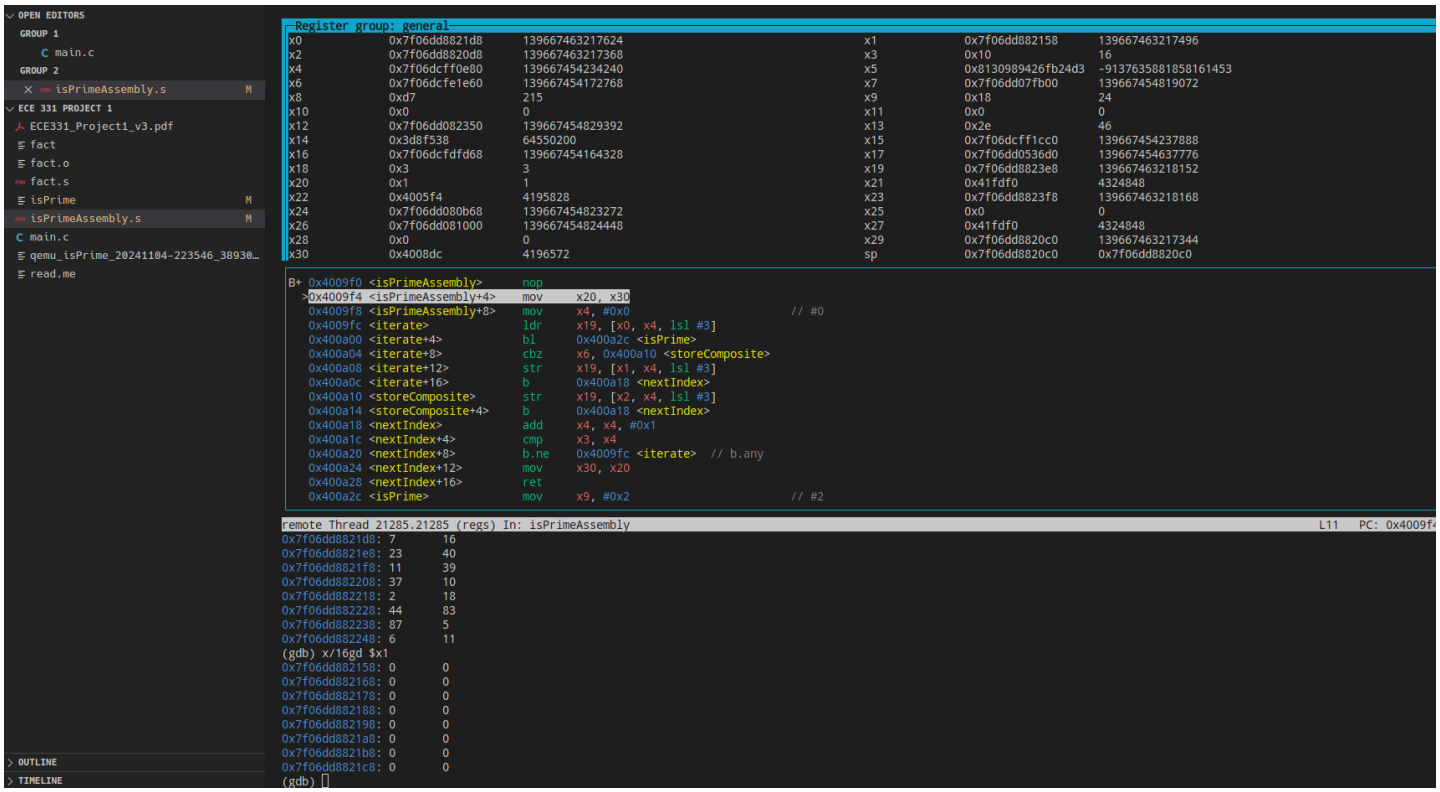
Images:

(Some commands for me were different as I was running this on my native Arch linux install instead of the provided Ubuntu Virtual machine)

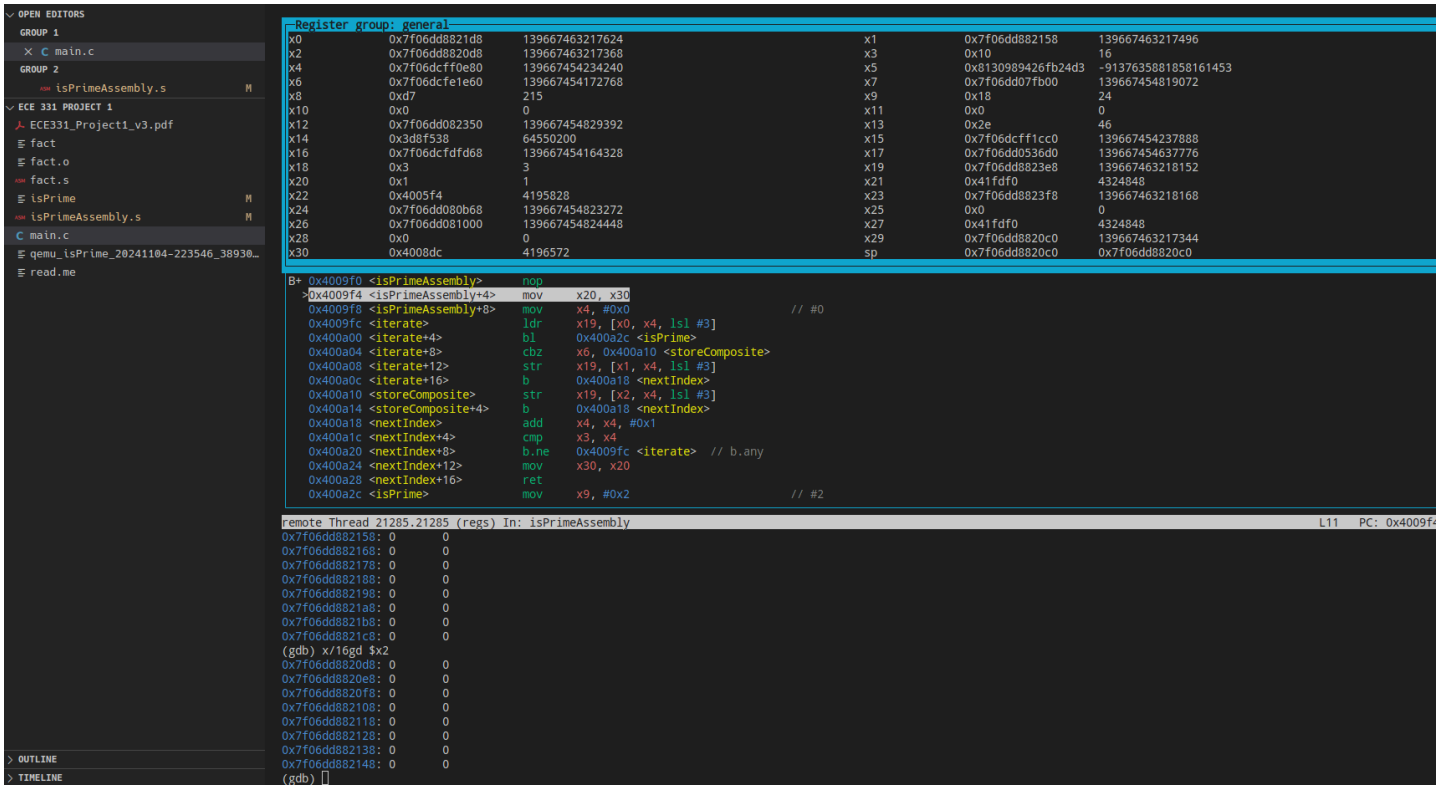
This arrayA stored at x0 as seen using x/16gd $x0 in gdb:



This arrayPrime stored at x1 as seen using x/16gd $x1 in gdb:

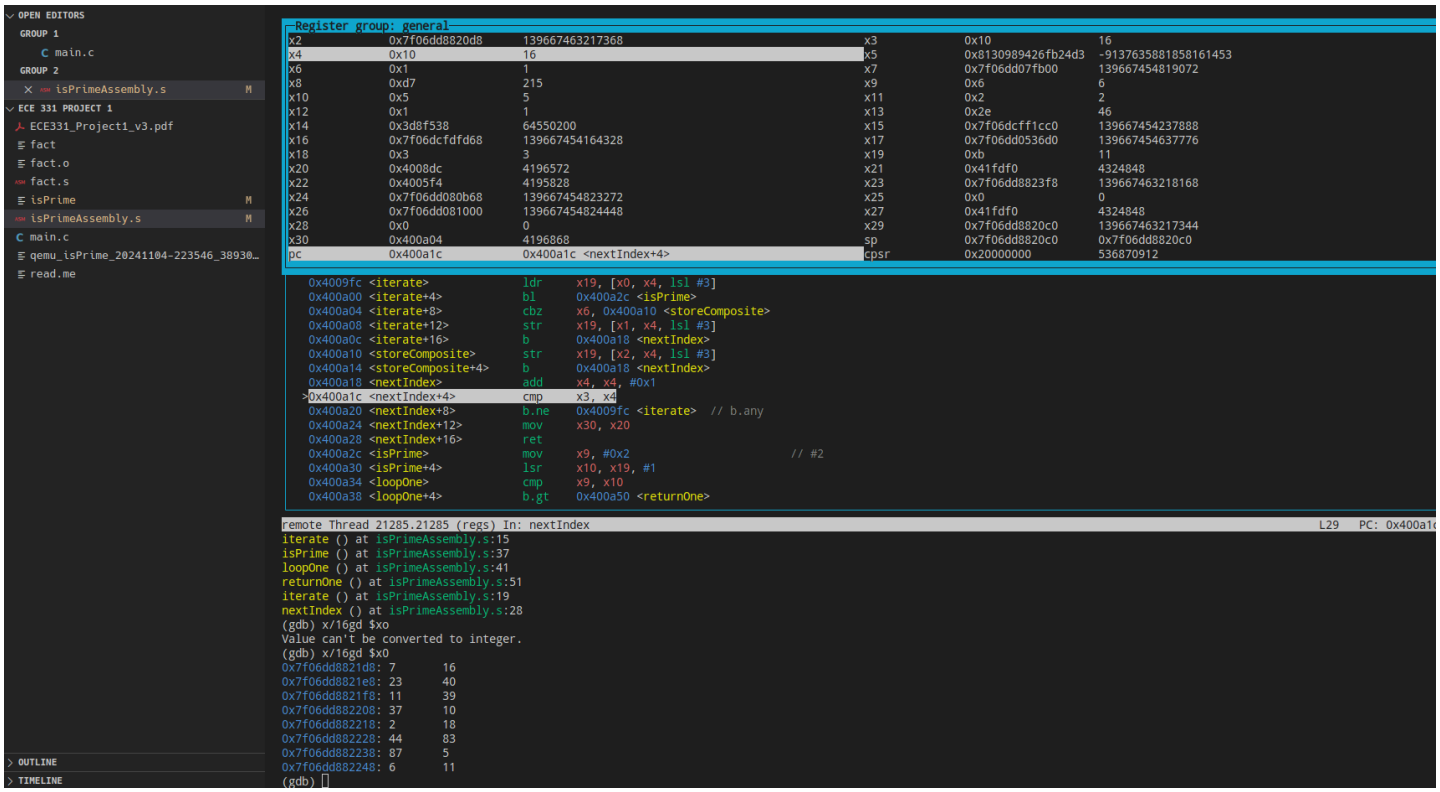


This arrayComposite stored at x2 as seen using x/16gd $x2 in gdb:

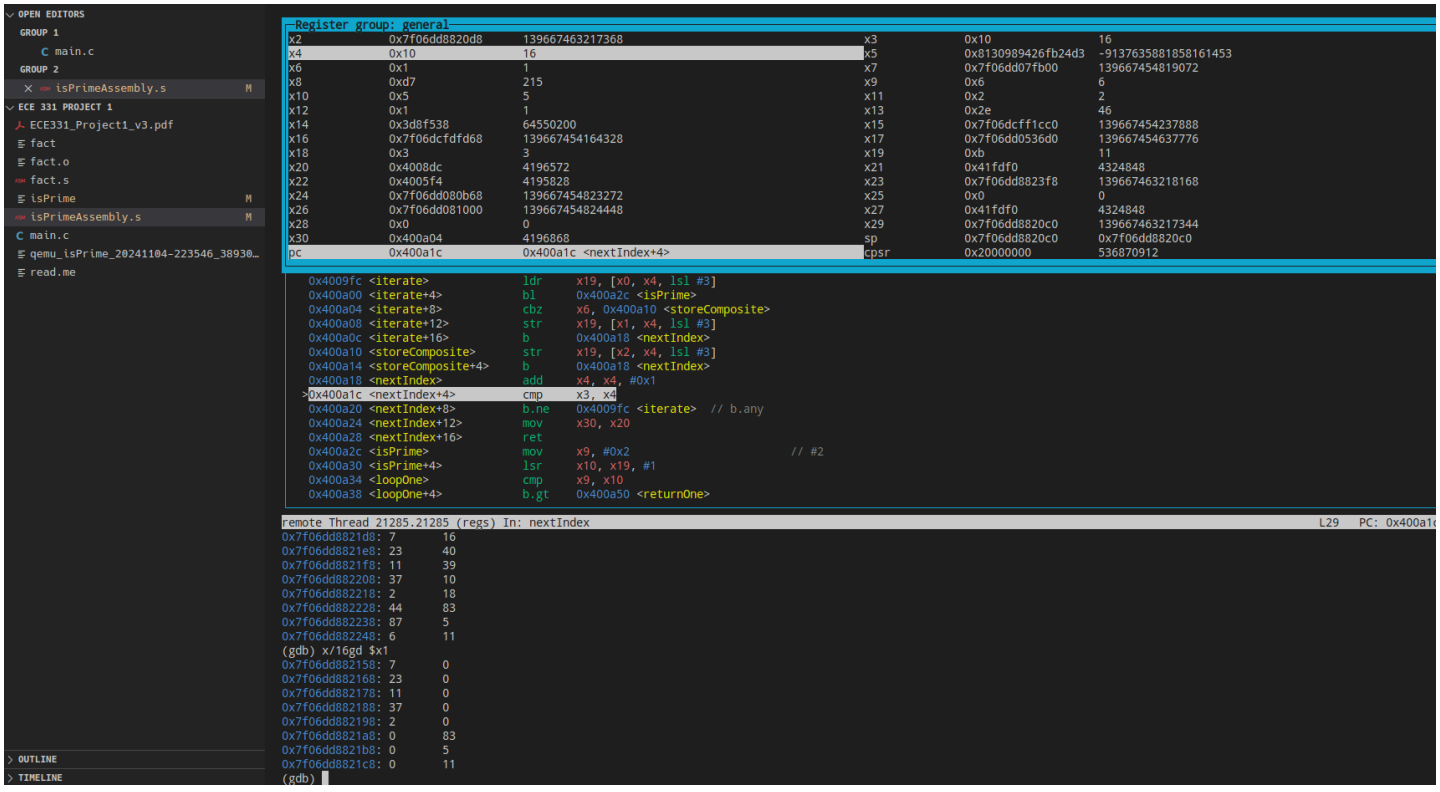


After stepping through the code:

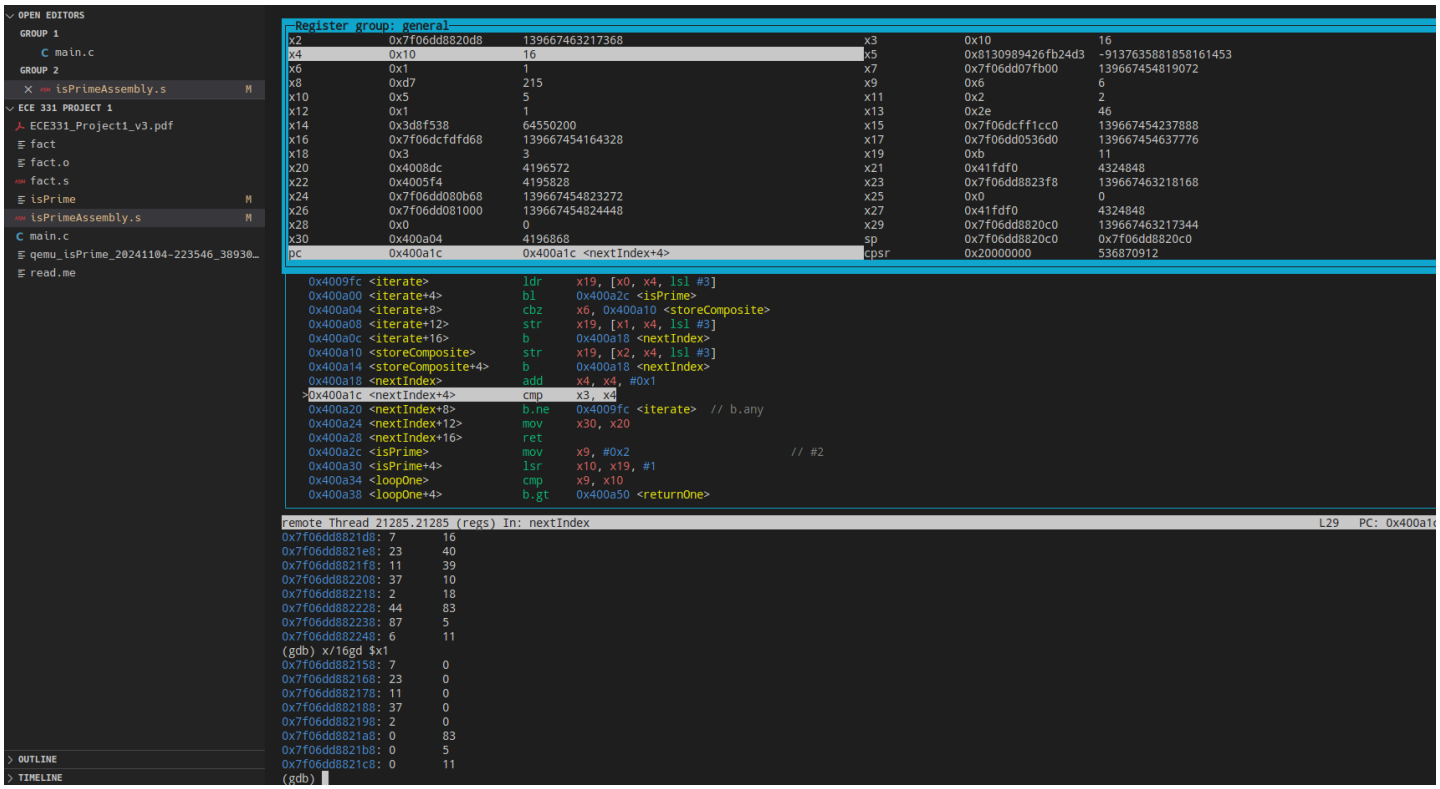
This arrayA stored at x0 as seen using x/16gd $x0 in gdb (Should not have changed):



This arrayPrime stored at x1 as seen using x/16gd $x1 in gdb:



This arrayComposite stored at x2 as seen using x/16gd $x2 in gdb:



Bonus:

Here's me running my code without a server, the C function is commented out just like the above gdb images!