Combining C and Assembly Language programs

1. Compile and launch: Easy way, write *make* on terminal.
    a. make file working:
        i. create object files for each function (A, B, C)
        ii. B is compiled with *nasm* elf64 mode
        iii. C has inline assembly so it also takes *-masm=intel*
        iv. Final the linker links A, B, C and returns the executable named ***b***
2. How it works
    a. Objective:
        i. Main calls A
        ii. A calls B with argument as 64 bit integer
        iii. B modifies it stack to *return* to C
        iv. C exits zero.
    b. Execution:
        i. The main function calls **A** in the most standard way possible (i.e. **A()**)
        ii. **A** declares a 64-bit integer. The Integer corresponds to "Hello" when printed. **A** then calls **B** with the argument (like B(argument)). **B** has been externed so that it doesn't throw error when making an object file.
        iii. **B** starts by saving rbp, greets using a write syscall and processes the 8byte long. Just like how a string is printed (byte by byte). When the write syscall is used on the 64-bit argument (whose address is passed via *lea* opcode), it is also instructed to only print 8 bytes…. So that's how "Hello" is printed.
        iv. Then, **B** calls **C** by modifying its stack.
            As per the calling conventions of x86_64, [rbp+8] needs to be overwritten with the memory address of the function *to return to*. Finally, after ret, we jump to **C.**
        v. **C** performs the toughest job of exiting the function. rax takes the syscall number of exit (i.e. 60), rdi takes the exit code (0). After syscall the program terminates.