There are three tasks today. You must complete all of them.

Task 1:

- Run the code given below.
- Try to understand what it does and how it works
- Go through the register descriptions and operations given below the code, which may help you with this code.

```
extern     printf
extern     scanf

SECTION .data

a:    dq    5
b:    dq    2
c:    dq    0

enter:     db "Enter two numbers: ",0
out_fmt:   db "%ld + %ld =%ld", 10, 0
out_fmt_2: db "%s",10,0
in_fmt:          db "%d",0
SECTION .text

global main
main:
        push    rbp

        mov rax,0
        mov rdi,out_fmt_2
        mov rsi,enter
        call printf

        mov rax, 0
      mov rdi, in_fmt
      mov rsi, a
      call scanf


        mov rax, 0
      mov rdi, in_fmt
      mov rsi, b
      call scanf
```

```
mov   rax,[a]
mov   rbx,[b]
add   rax,rbx
mov   [c],rax
mov   rdi,out_fmt
mov   rsi,[a]
mov   rdx,[b]
mov   rcx,[c]
mov   rax,0
   call    printf

pop   rbp

mov   rax,0
ret
```

General-Purpose Registers

64-bit	32-bit	16-bit	8-bit (high/low)	Purpose / Usage
RAX	EAX	AX	AH/AL	Accumulator; arithmetic, logic, I/O
RBX	EBX	BX	BH/BL	Base; data storage, memory addressing
RCX	ECX	CX	CH/CL	Counter for loops, string operations
RDX	EDX	DX	DH/DL	I/O, multiplication/division
RSI	ESI	SI	SIL	Source index for string/data operations
RDI	EDI	DI	DIL	Destination index for string/data operations
RBP	EBP	BP	—	Base pointer for stack frames
RSP	ESP	SP	—	Stack pointer
R8–R15	R8D–R15D	R8W–R15W	R8B–R15B	Additional general-purpose registers (64-bit only)

Data Movement Instructions

Instruction	Syntax	Purpose
`mov dest, src`	`mov rax, rbx`	Copies data from `src` to `dest`. Can be register, memory, or immediate.
`push reg/mem`	`push rbp`	Pushes value onto stack; decrements `rsp` by 8 (64-bit).
`pop reg`	`pop rbp`	Pops value from stack into register; increments `rsp` by 8.

Arithmetic Instructions

Instruction	Syntax	Purpose
`add dest, src`	`add rax, rbx`	Adds `src` to `dest` and stores result in `dest`.
`sub dest, src`	`sub rax, rbx`	Subtracts `src` from `dest`.
`mul src`	`mul rbx`	Unsigned multiply `rax * src`. Result stored in `rdx:rax`.
`imul src`	`imul rbx`	Signed multiply. Can store result in `rax` or another register.
`div src`	`div rbx`	Unsigned divide `rdx:rax / src`. Quotient → `rax`, remainder → `rdx`.
`idiv src`	`idiv rbx`	Signed division.

Task 2:

Scan three variables a,b, and c. Print the value of 2a + 3b + c

Task 3:

Scan a variable x. Print the value of the sum of the numbers from 1 to x. You may assume x is a positive integer.