1. Consider the following C function:

```
long foo(long i, long j)
{
    long a = i + j;
    long b = i * j;
    return a + b;
}
```

and its X86Lite assembly code:

```
foo(long, long):
        pushq       %rbp
        movq        %rsp,      %rbp
        subq         $32,      %rsp
        movq        %rdi,   -8(%rbp)
        movq        %rsi,  -16(%rbp)          (a)
        movq     -8(%rbp),     %rax
        addq    -16(%rbp),     %rax
        movq        %rax,  -24(%rbp)
        movq     -8(%rbp),     %rax
        imulq   -16(%rbp),     %rax
        movq        %rax,  -32(%rbp)          (b)
        movq    -24(%rbp),     %rax
        addq    -32(%rbp),     %rax
        addq          $32,     %rsp          (c)
        popq        %rbp
        retq
```

Draw the states of the stack after each of the annotated instructions have been executed. The System V ABI x86-64 calling conventions are used.

2. Write an X86Lite function with one integer argument N. If N is non-positive then the program returns 0. Else, it returns $\sum_{i=0}^{N-1} i$. The sum should be computed with a loop and not the closed form formula. The System V ABI x86-64 calling conventions are used and you can ignore integer overflows.

3. The X86Lite address space is divided into three parts: a) code & data, b) the heap, and c) the stack. Two of the uses of stack are a) to store bookkeeping information for function calls and b) to store local and temporary data. A stack overflow occurs if the stack "grows" too much (the limit varies depending on the operating system, configuration, etc, but it typically is a handful of megabytes). Write an X86Lite program that crashes because of a stack overflow.