1.) Consider the following C functions. For each, indicate what variables would need to be saved on to the stack in each function. Please label each answer with the function name.

Hint: Keep in mind that a pointer enables code to modify the memory that the pointer points to.

```c
void function(int a, int b, int c){
    int result = 0;
    result = bar(a, b*c);
    return result;
}

void function2(int a, int b, int c){
    int result = 0;
    result = a + bar(a, b*c);
    return result;
}

void function3(int a, int b, int c){
    int result = 0;
    result = bar(a, b) + bar(a, c);
    return result;
}

void function4(int *a, int b, int c){
    int result = *a;
    result = bar(a, b*c) + bar(a+1, b*c);
    return result;
}
```

These items are needed after one of the function calls so must be saved on the stack.
function: nothing
function2: a
function3: a, b OR c, and result of one call to bar
function4: a, b*c OR b and c, result of one of the calls
Rewrite the following C function in x86-64 assembly. Make sure to follow the register and procedure call conventions.

```c
long foo(long a, long b) {
    if(a <= 0){
        return b;
    } else{
        return b + foo(a-1, b);
    }
}
```

Notes: using the standard C calling conventions, the first function argument is always stored in register `%rdi`, the second function argument in register `%rsi`, and the return value in register `%rax`.

foo:
```
subq $16, %rsp       # grow stack
movq %rsi, (%rsp)    # store b on stack
movq %rsi, %rax      # result = b.
cmpq $0, %rdi        # a-0
jle done             # (a-0) <= 0, done
subq $1, %rdi        # a-1.
callq foo
movq (%rsp), %rsi    # b from stack.
addq %rsi, %rax      # b + foo(a-1,b)
```

done:
```
addq $16, %rsp       # shrink stack.
ret
```
Consider the following C source code where a two dimensional array of longs has been declared in the following manner.

\[
\text{long array}[M][N];
\]

This declaration of array is followed by its initialization (not shown). Then the C code (left) summing elements in column \( j \) is executed. The x86-64 assembly code (right) is a translation of that C code.

\[
\begin{align*}
\text{long result} &= 0; \\
\text{int } j &= 1; \\
\text{for}(\text{int } i = 0; i < M; i+=1)\{ \\
&\quad \text{result} += \text{array}[i][j]; \\
\}
\end{align*}
\]

\[
\begin{align*}
\text{movl } &0, \text{ } r8 \quad \# \text{ result} \\
\text{movl } &0, \text{ } rdx \quad \# i \\
\text{jmp } &\text{cnd} \\
\text{loop:} \\
&\quad \text{movl } 1, \text{ } rax \quad \# j \\
&\quad \text{movq } \text{rdx}, \text{ } rsi \\
&\quad \text{shl } 0x2, \text{ } rsi \\
&\quad \text{add } rsi, \text{ }rax \\
&\quad \text{addq } \text{array}(, \text{ } rax, 8), \text{ } r8 \\
&\quad \text{addl } 1, \text{ } rdx \\
\text{cnd:} \\
&\quad \text{cmpl } 0x2, \text{ } rdx \\
&\quad \text{jle } \text{loop}
\end{align*}
\]

What are the values of \( M \) and \( N \)?

\( M=3, \ N=4 \)