1.) Consider the following C function that updates an array's contents:

```c
void fcn(long *array, long i, long j)
{
}
```

The compiler has generated the following incomplete version of code for this function:

```
fcn:
    movl $1, %ecx                  # tmp = 1
    addq 48(%rdi), %rcx           # tmp += array[6]
    movq (%rdi, %rdx, 8), %r8     # %r8 = &array[j]
    addq %r8, %rcx                # tmp += array[j]
    leaq __-32(%rdi, %rsi, 8), %r9 # & (array[i-4])
    movq %rcx, __(%r9)            # array[i-4] = tmp
    ret
```

Fill in the incomplete portions indicated by empty lines to complete the code.

**Notes:** using the standard C calling conventions, the first argument is always stored in register %rdi, the second in %rsi, and third in %rdx. On this system, a long is 8 bytes.
2.) Consider the assembly code (left) that corresponds to the partially complete C function \texttt{fcn} (right). Please fill in the missing lines of the C code.

<table>
<thead>
<tr>
<th>fcn:</th>
<th>long \texttt{fcn}(long a, long b) {</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{movl} $0, %eax}</td>
<td>long result = 0;</td>
</tr>
<tr>
<td>\texttt{cmpq} %rdi, %rsi</td>
<td>if( (b - a) &lt; 0){</td>
</tr>
<tr>
<td>\texttt{jl} .L3</td>
<td>\hspace{1cm} return result;</td>
</tr>
<tr>
<td>\hspace{1cm} subq %rdi, %rsi</td>
<td>}</td>
</tr>
<tr>
<td>\hspace{1cm} addq %rdi, %rax</td>
<td>\hspace{1cm} b = b - a;</td>
</tr>
<tr>
<td>\hspace{1cm} cmpq %rdi, %rsi</td>
<td>\hspace{1cm} result += a;</td>
</tr>
<tr>
<td>\hspace{1cm} jge .L1</td>
<td>\hspace{1cm} } while( (b - a) &gt;= 0);</td>
</tr>
<tr>
<td>\hspace{1cm} ret</td>
<td>\hspace{1cm} return result;</td>
</tr>
<tr>
<td>.L3:</td>
<td>}</td>
</tr>
</tbody>
</table>

Notes: using the standard C calling conventions, the first function argument is always stored in register $\texttt{rdi}$, the second function argument in register $\texttt{rsi}$, and the return value in register $\texttt{rax}$. 
3.) Consider the following C functions. For each, indicate which variables would need to be saved on to the stack in each function, assuming no callee saved registers are used. (Hint: this question is about register conventions at function calls.) Please label each answer with the function name.

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

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

void function3(int *a, int b, int c){
    int result = 0;
    result = *a + bar(a, b, c) + bar(a+1, b, c);
    return result;
}
```

The question should have been clearer that values not variables were to be specified. Values returned by the function calls will also need to be stored on the stack before making the second function call. (I did not take off points for not mentioning the function return values.)

function1:
result or a must be stored on the stack since one of them will be used after the function call.

function2:
If bar(a, b) occurs first, then b and c need to be saved on the stack. The result of bar(a, b) will also need to be stored on the stack before making the call to bar(b, c) if bar(b, c) occurs first, then a and b need to be stored on the stack. Then the result of bar(b, c) needs to be stored onto the stack as does a and b.

function3:
The result of the first function call and a, b, and c need to be stored on the stack.
4.) Rewrite the following C function in x86-64 assembly. Make sure to follow the register and procedure call conventions.

```c
long callee(long *ptr, long i);
long caller(long *ptr, long i) {
    if(ptr == 0){
        return i;
    } else{
        return ptr[i] + callee(ptr, i);
    }
}
```

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.

caller:
    testq %rdi, %rdi
    jne else
    movq %rsi, %rax
    retq
else:
    movq (%rdi, %rsi, 8), %r8
    pushq %r8
    callq callee
    popq %r8
    addq %r8, %rax
    retq