Calling Conventions, Higher Order Functions, and Functional Programming

1. C macros use the **call by name** parameter passing convention. Consider the macro defined below (in C++). Note that ## in a C macro concatenates two tokens into a single token.

```
#define MAKE_SWAP(T)         \
void swap_##T(T &x, T &y) { \
    T tmp = x;               \
    x = y;                   \
    y = tmp;                 \
}
MAKE_SWAP(int)
MAKE_SWAP(string)
```

   What is the result of the two calls to `MAKE_SWAP`? Justify your answer by explaining how call by name works.

2. Assume the language below uses reference semantics. What is the output of the following code when the language uses **call by value-result**? What is the output when it uses **call by reference**?

```
class Foo {
    int x = 0
}

void bar(Foo f) {
    f.x = 10
}
```

```
Foo f = Foo()
Foo g = f

bar(f)

print(f.x, g.x)
```

Call by Value Result	Call by Reference

3. What is the output of `h(3)` and `g(f)` when the language uses each of the following scope and binding policy rules? Justify your answer.

```
int x = 1
int f(int y) { return x + y }

void g(int h(int b)) {
    int x = 2
    return h(3) + x
}

{
    int x = 4
    int z = g(f)
}
```

Static Scope + Deep Binding	Dynamic Scope + Deep Binding	Dynamic Scope + Shallow Binding

4. Write a function in Scheme that returns the number of ways to count change given an amount (in cents) and a list of denominations. *Hint:* you should have two base cases and two recursive calls.

```
(define (count-change amount denominations)
; write your solution here
```

)

```
(count-change 11 (list 1 5 10 25)) ; evaluates to 4
```