1.

parameter transmission mechanism	output						
call–by–value	10	13	0	200	150	175	190
call–by–reference	5	8	0	200	20	175	190
call–by–name	5	8	0	200	150	175	190

2. a. The output would be `7`.

   b. The output would be `14`.

   c. The output would be `24`.

3. Using call–by–value, when the argument `func(1, 1)` is evaluated the function would recurse indefinitely, since the actual parameter `func(1, 1)` corresponding to the formal parameter `a` would first be evaluated. For the call `func(1, 1)` the parameter `b` is nonzero, and the function will call itself recursively with the exact same parameter values it was itself called with. This is not a good idea if you have anything productive you want to do after that.

   If call–by–need were used instead, `a` would never be evaluated for the call `func(1, 1)` since it's not used, so `func(1, 1)` would never be called. Evaluating `b` is safe, as `func(0, 0)` would cause a recursive call in which `b`'s value was 0, causing the function to simply return 0 instead of calling itself recursively again.

4. Any variable whose address can only be resolved at runtime may cause this procedure to not function correctly; specifically when the address of one parameter depends on the other's value. For example, the call `swap(i, c[i])` won't swap the values of `i` and `c[i]`.

5. The least restrictive condition: do not allow a variable whose address can only be resolved at runtime to be an actual parameter.

6. The output that would be produced if dynamic scoping were used would be:

```
e:  9  1  8  2
b:  8  2  6  3
a:  9  3  8  7
```

   The output that would be produced if static scoping were used would be:

```
e:  9  1  8  2
b:  9  6  8  8
a:  9  6  8  8
```