1. from handout week2.a

1.a C code

```
...
uint64_t f(uint64_t* ptr)
{
    uint64_t x = 0;
    x = g(*ptr);
    return x + 1;
}

uint64_t g(uint64_t a)
{
    uint64_t x = 2*a;
    q = &x; // <-- THIS IS AN ERROR (AKA BUG)
    return x;
}
...
```

1.b assembly code (by “gcc -O0”)

```
...
  f:
      pushq %rbp           # prologue: store caller’s frame pointer
      movq %rsp, %rbp      # prologue: set frame pointer for new frame

      subq $32, %rsp       # make stack space
      movq %rdi, -24(%rbp) # Move ptr to the stack
                           # (ptr now lives at rbp - 24)
      movq $0, -8(%rbp)    # x = 0 (x’s address is rbp - 8)

      movq -24(%rbp), %r8  # move ’ptr’ to %r8
      movq (%r8), %r9      # dereference ’ptr’ and save value to %r9
      movq %r9, %rdi       # Move the value of *ptr to rdi,
                           # so we can call g

      call g               # invoke g

      movq %rax, -8(%rbp)  # x = (return value of g)
      movq -8(%rbp), %r10  # compute x + 1, part I
      addq $1, %r10        # compute x + 1, part II
      movq %r10, %rax      # Get ready to return x + 1

      movq %rpb, %rsp      # epilogue: undo stack frame
      popq %rbp            # epilogue: restore frame pointer from caller
      ret                  # return
...
```

2. "gcc -O3 -S example.c"

```
...
f:
    subq    $24, %rsp       // push frame: allocate stack frame 24B

                            // [Security]
    movq    %fs:40, %rax    // canary value (%fs:40) for detecting stack
                            // smashing attacks
    movq    %rax, 8(%rsp)   // put canary at 8(%rsp)

    xorl    %eax, %eax      // %eax=0 [%eax is the low 32bit of %rax]
    movq    %rsp, %rdx      // %rdx = %rsp
    movq    (%rdi), %rax    // %rax = *ptr (%rdi contains the first arg to
                            // function f, which is "ptr")
    movq    %rdx, q(%rip)   // "q" is the global variable; set "q" to %rdx:

                            // [Security]
    movq    8(%rsp), %rcx   // copy the canary at stack
    xorq    %fs:40, %rcx    // check if the value has been changed
    jne    .L9              // if so, alert!!!

    leaq    1(%rax,%rax), %rax  // %rax = %rax + %rax + 1
                                // (!!! this is function g!)
    addq    $24, %rsp           // pop frame
    ret                         // return to main
...
```