```
Sep 12, 21 22:05
                                         example.c
                                                                               Page 1/1
   /* CS202 -- handout 1
       compile and run this code with:
         $ gcc -g -Wall -o example example.c
         $ ./example
4
         examine its assembly with:
         $ gcc -00 -S example.c
         $ [editor] example.s
8
q
   #include <stdio.h>
11
12
   #include <stdint.h>
13
   uint64_t f(uint64_t* ptr);
15
   uint64 t q(uint64 t a);
16
   uint64_t* q;
   int main (void)
18
19
        uint64_t x = 0;
20
       uint64_t arg = 8;
21
22
23
       x = f(\&arg);
24
25
        printf("x: %lu\n", x);
       printf("dereference q: %lu\n", *q);
26
27
        return 0:
28
29
30
   uint64_t f(uint64_t* ptr)
31
32
33
        uint64_t x = 0;
34
       x = q(*ptr);
        return x + 1;
35
36
37
38
   uint64_t g(uint64_t a)
39
        uint64_t x = 2*a;
       q = &x; // <-- THIS IS AN ERROR (AKA BUG)
41
42
        return x;
43
```

```
as.txt
Sep 12, 21 22:05
                                                                             Page 1/1
   2. A look at the assembly...
        To see the assembly code that the C compiler (gcc) produces:
           $ gcc -00 -S example.c
        (then look at example.s.)
        NOTE: what we show below is not exactly what gcc produces. We have
        simplified, omitted, and modified certain things.
8
9
        main:
                                         # prologue: store caller's frame pointer
10
            pushq
                    %rbp
                    %rsp, %rbp
                                        # prologue: set frame pointer for new frame
11
            movq
12
                    $16, %rsp
                                        # make stack space
            subq
13
14
15
            mova
                    $0, -8(%rbp)
                                        \# x = 0 (x lives at address rbp - 8)
16
            movq
                    $8, -16(%rbp)
                                        # arg = 8 (arg lives at address rbp - 16)
17
                   -16(%rbp), %rdi
                                        # load the address of (rbp-16) into %rdi
18
            leag
19
                                        # this implements "get ready to pass (&arg)
                                        # to f"
20
21
                                        # invoke f
            call
                    f
22
23
            mova
                    %rax, -8(%rbp)
                                        \# x = (return value of f)
24
25
            # eliding the rest of main()
26
27
        f:
28
                                        # prologue: store caller's frame pointer
29
            pushq
                    %rbp
30
            movq
                    %rsp, %rbp
                                        # prologue: set frame pointer for new frame
31
32
            subq
                   $32, %rsp
                                        # make stack space
33
            movq
                   %rdi, -24(%rbp)
                                        # Move ptr to the stack
                                        # (ptr now lives at rbp - 24)
34
                                        \# x = 0 (x's address is rbp - 8)
            movq
                   $0, -8(%rbp)
35
37
            mova
                    -24(%rbp), %r8
                                        # move 'ptr' to %r8
38
                    (%r8), %r9
                                        # dereference 'ptr' and save value to %r9
            movq
                                        # Move the value of *ptr to rdi,
39
                    %r9, %rdi
            movq
                                        # so we can call q
41
42
            call
                    g
                                        # invoke g
43
44
            movq
                   %rax, -8(%rbp)
                                        \# x = (return value of g)
                   -8(%rbp), %r10
                                        # compute x + 1, part I
45
            movq
                    $1. %r10
                                        # compute x + 1, part II
            addq
46
47
            movq
                    %r10, %rax
                                         # Get ready to return x + 1
48
49
            movq
                    %rpb, %rsp
                                         # epilogue: undo stack frame
50
            popq
                    %rbp
                                        # epilogue: restore frame pointer from caller
51
            ret
52
53
        g:
            pushq
                    %rbp
                                        # prologue: store caller's frame pointer
54
            movq
55
                    %rsp, %rbp
                                        # prologue: set frame pointer for new frame
56
57
            . . . .
58
                                        # epilogue: undo stack frame
59
            movq
                    %rbp, %rsp
60
                                         # epiloque: restore frame pointer from caller
            popq
                    %rbp
                                         # return
            ret.
```