1) See endian.c
2) See simple_string.c
3) is this an ok version of simp_str_copy? if not why? No

```
simp_str* simp_str_copy(simp_str *in)
{
    simp_str *ret = malloc(sizeof(simp_str);   missing parenthesis  => malloc(sizeof(simp_str));
                <------ (should check here that malloc didn't return a NULL ptr)
    ret->len = in->len;
    ret->buf_len = in->buf_len;
    ret->str = malloc(ret->len);          => malloc(ret->buf_len);
    if (NULL == ret->str)
    {
       return NULL;
    }
    strncpy(ret->buf, in->buf, ret->len);          => strncpy(ret->str, in->str, ret->len);
                                                   (buf is not defined in simp_str)
    return ret;
     }
```

4) What is the value of b in the code fragment below?

```
char a[4];
unsigned short b;

a[0] = 1; a[1] = 2; a[2] = 3; a[3] = 4;

b = *(unsigned short *)a;
```

b will get 2 bytes out of a, so it'll get a[0] = '01' and a[1] = '02'

but since my PC is little endian, it gets stored as x'02'x'01'

so, b = x'201'

5) See nth_bit_checker.c
6) Consider the following snippet of (admittedly contrived) threaded code:

For gcc (gcc version 4.8.2 at least) when we compile it like this:

gcc thread_question.c -o thread_question -lpthread

and run it, everything seems to work fine. However when we compile it like this:

gcc thread_question.c -o thread_question -lpthread -O3

It seems to just hang forever! What's going on there? How can we fix it?

Ordering isn't really guaranteed and the optimization might have reordered it. Making the "g_keep_running" atomic will allow it to be updated from multiple threads and guarantee ordering.

Aside from removing the sleep(2); line of code, changing

"int g_keep_running = 1;" to "atomic_int g_keep_running = 1;" fixes the issue.

See thread_question.c