

1. [20 pts.]

- a. When two operators with the same precedence are applied to an operand, their associativity dictates whether they're performed left-to-right or right-to-left.

Note that associativity is not the same thing as precedence.

b. Here are several possibilities:

- Structures contain data only, not methods.
- Everything in a structure is public.
- Structures don't have inheritance.
- When a structure in C is declared you have the structure itself, while when an object in Java is created you have a reference to it.
- Along the same lines, assigning a structure will make a copy of all the structure's fields in C, while assigning a class object in Java will just copy the reference (the two references will refer to the same object).
- Along similar lines, trying to compare structure in C won't compile, but trying to compare references to objects in Java will work (but just tests whether the references are referring to the same object).

- c. `strlen()`'s return type is defined to be `size_t`, which is an unsigned integer type. An expression that has only `unsigned int` operands will have an `unsigned int` type result. But if the string `t` is longer than the string `s`, the result of the expression will be negative, which can't be represented as an `unsigned int`. (It will appear as a large positive number.)

The solution is to cast the return type of `strlen()` to `int`.

2. [18 pts.]

a. 11

c. 2

e. 't'

b. 1

d. 'a'

f. 21

3. [32 pts.] Here is one solution:

```
int add_to_course(University *univ, int course, int num) {
    int i;
    int num_found= 0;
    if (univ == NULL)
        return -1;
    for (i= 0; i < univ->in_use; i++)
        if (univ->all_courses[i].course_nbr == course) {
            univ->all_courses[i].sections[0].num_students += num;
            univ->all_courses[i].sections[1].num_students += num;
            num_found++;
        }
    return num_found;
}
```

4. [30 pts.] Here are three versions, which all use the left-shift operator (<<) (and other bitwise operators), but not the right-shift operator (>>). A correct solution could also be written using the right-shift but not the left-shift operator.

Note that `sizeof` is an operator (covered in the chapter of the text on operators), not a function, so the restriction in the problem does not apply to avoiding use of `sizeof`.

```
unsigned int swap_bits(unsigned int num, int m, int n) {
    int i;

    if (m <= n && m > 0 && n > 0 && n <= sizeof(unsigned int) * 8)
        for (i= m; i <= n; i++)
            num ^= 1 << (i - 1);

    return num;
}
```

```
unsigned int swap_bits2(unsigned int num, int m, int n) {
    int i;
    unsigned int mask= 0;

    if (m <= n && m > 0 && n > 0 && n <= sizeof(unsigned int) * 8) {

        for (i= m - 1; i < n; i++)
            mask |= (1 << i);

        num= (num & ~mask) | (~num & mask);

    }

    return num;
}
```

```
unsigned int swap_bits3(unsigned int num, int m, int n) {
    int i;
    unsigned int bit, ans= 0;

    if (m <= n && m > 0 && n > 0 && n <= sizeof(unsigned int) * 8) {

        for (i= 1; i <= sizeof(unsigned int) * 8; i++) {

            if (i < m || i > n)
                bit= num & (1 << (i - 1));
            else bit= ~num & (1 << (i - 1));

            ans |= bit;
        }

        } else ans= num;

    return ans;
}
```