

Do not open this exam until you are told. Read these instructions:

1. This is a closed book exam. **No calculators, notes, or other aids are allowed.** If you have a question during the exam, please raise your hand.
2. **You must turn in your exam immediately when time is called at the end.**
3. 4 problems, 100 points, 75 minutes.
4. In order to be eligible for as much partial credit as possible, show all of your work for each problem, **write legibly**, and **clearly indicate** your answers. Credit **cannot** be given for illegible answers.
5. You will **lose credit** if **any** information above is incorrect or missing, or your name is missing from any page.
6. Minor syntax errors will be ignored in any code you have to write on this exam, as long as the concepts are correct.
7. Part of this page and another page are for scratch work. If you need extra scratch paper after you have filled these areas up, please raise your hand. Scratch paper must be turned in with your exam, with your name and UMD ID written on it. Scratch paper **will not** be graded.
8. To avoid distracting others, **no one** may leave until the exam is over.
9. The Campus Senate has adopted a policy asking students to include the following handwritten statement on each examination and assignment in every course: "*I pledge on my honor that I have not given or received any unauthorized assistance on this examination.*" Therefore, **just before turning in your exam**, you are requested to write this pledge **in full and sign it** below:

1. [20 pts.] Answer each question below **briefly**.

a. Briefly describe what the associativity of an operator refers to.

b. Briefly describe any *two* ways in which structures in C differ from classes in Java.

c. Assume that `s` and `t` are both valid strings and both end with a null character. Under what circumstances will the expression `strlen(s) - strlen(t)` produce inconsistent or incorrect results? And how can the problem be avoided?

2. [18 pts.] Consider the declarations below:

```
int x= 4, y= 7, z= 10;
int a[10]= {20, 19, 18, 17, 16};
int *p= a;
char name[10]= "harriet";
char *sp = name + 3;
```

Give the value that would be produced by evaluating each of the following expressions. Assume that each expression is independent, and its results don't affect the ones below it. You may assume, if needed, that an `int` is 4 bytes, a `char` is 1 byte, and a pointer is 8 bytes. Each of the expressions is valid C, and none of them have any errors. If it's not possible to know the exact value of any of the expressions (for instance, if one of them would produce a memory address somewhere as its value) just write "unknown".

a. `x < y ? ++z : --z` _____

b. `(x * 2 > z) || (x + y > z)` _____

c. `(a + 8) - (a + 6)` _____

d. `sp[-2]` _____

e. `*(name + 6)` _____

f. `++*p` _____

3. [32 pts.] Consider the structure types shown to the right. Implement the function `add_to_course` whose prototype appears following the structure definitions. It should add `num` students to every section of every course that has the same course number as the value of its parameter `course`, in the `University` structure that its parameter `univ` refers to. It should add the students to every course with the given course number, regardless of what department the course is in. Note there may be many courses in the parameter `univ` that have the same course number, like ENGL 100 and MATH 100 and PSYC 100. If `course` is 100 and `num` is 5, the function should add 5 students to each section of each course that has course number 100. The function should return the total number of courses that are modified in this way. If the parameter `univ` is `NULL`, then the function should return -1.

The function should only examine the elements in the array of the `University` structure that `univ` refers to that are actually in use, as indicated by its `in_use` field.

On the next page, write the complete function. Note that your function may not need to use all the fields of all the structures. Your function should work correctly in all cases, without any execution errors. Comments are unnecessary, but your function must be written neatly, and use good style and formatting.

```

/* A Section structure stores the data
   about one discussion section, namely
   its section number, and how many
   students are in that section. */

typedef struct {
    int section_nbr;
    int num_students;
} Section;

/* A Course stores the data about one
   course, and has a department ("CMSC",
   "ENGL", etc.), a number, and
   exactly two sections. */

typedef struct {
    char dept[5];
    int course_nbr;
    Section sections[2];
} Course;

/* A University structure has an array
   of Course structures, (containing
   courses that may be from different
   departments), and an integer
   indicating how many of the array's
   elements (starting from the beginning
   of the array) are in use. */

typedef struct {
    Course all_courses[1000];
    int in_use;
} University;

int add_to_course(University *univ,
                 int course, int num);

```


4. [30 pts.] Write the following function:

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

It is supposed to return the value of its parameter `num`, except that the bits from the `m`'th bit through the `n`'th bit of the value are reversed. In other words, whichever of those bits were 0 should now be 1 in the result, and vice versa. Bits are considered to be numbered from the right of a number, with the rightmost bit being numbered 1, so the call `swap_bits(x, 5, 8)` would invert each of the fifth, sixth, seventh, and eighth bits from the right of `x`. The values of all the other bits should be unchanged. For example, the call `swap_bits(0xaaa, 5, 8)` (0xaaa is 1010 1010 1010 in binary) should return an `unsigned int` with value (given in hexadecimal) `a5a` (a5a is 1010 0101 1010 in binary).

The function should return the value of `num` unchanged if the value of `m` is greater than that of `n`, or either of their values are zero or less, or `n` is greater than the number of bits in an `unsigned int` (see below). Your function **may not** call any other functions.

To receive full credit:

- Your function **should not** contain any assumptions about how much space an `unsigned int` occupies in memory. In other words, it should work correctly on different machines or compilers regardless of how large an `unsigned int` happens to be. (If you don't know how to satisfy this restriction, **clearly indicate** what you are assuming the size of an `unsigned int` is, and where this value is being used in your function.)
- Your function may use **either** the bitwise left-shift operator (`<<`) **or** the bitwise right-shift operator (`>>`), but it **cannot use both of them**. Any other bit operators may be used as desired without restriction.

You can get a reasonable score on this problem even if you can't satisfy one or both of these conditions. Comments are unnecessary, but your functions must be written neatly, and use good style and formatting.