The exam was a 50 minute, open-book, open-notes exam. Electronic devices were not allowed.

**Question 1a**

\[
64\_{10} = 1000000_2 \\
63\_{10} = 0111111_2
\]

So, bitwise AND produces a 0 in each bit position, resulting in an output of 0.

**Question 1b**

A two's complement number starting with a 1 bit is a negative number. To convert to the equivalent positive number, flip each bit and add 1. Flipping the bits results in 00000001. Adding 1 results in 000000010, or 2. So, the decimal integer is -2.

**Question 1c**

The largest unsigned integer is 11111111. Rather than converting to decimal (255) and then to octal, it is possible to convert the binary representation directly to octal. Grouping sets of three bits leads to 011 111 111, or 377 in octal.

**Question 1d**

Dividing by 64 \(2^6\) is the same as shifting right by 6 bits. Shifting left by 3 bits and right by 6 bits is equivalent to shifting right by 3 bits. Three bits corresponds to one octal digit. So, the result of shifting is 53\(_8\). This is equivalent to 5*8+3, or 43 in decimal.

**Question 1e**

\[
76_{10} = 00..01001100_2 \\
15_{10} = 00..00001111_2
\]

The result of bitwise XOR is 00...01000011\(_2\). The result of bitwise NOT is 11...10111100\(_2\). This is a signed negative number in two's complement representation. To convert to an equivalent positive number, flip each bit and add 1. So, the result is 00...01000011\(_2\) + 1, or 00...01000100\(_2\). In base ten, this is 64+4, or 68, so the answer is -68.

A common mistake was to omit the leading 0s and get an answer of 60.

**Question 2a**

The expression within the "if" statement compares a pair of chars; it should compare a pair of strings. As such, the function would produce the wrong result for the input parameter "d". A correct implementation would use `strcmp()`.

**Question 2b**

The program produces the wrong output when the input has an odd number of characters. Here is a correct implementation:

```c
void q2b(void) {
    int c;
    for (;;) { 
        c = getchar();
        if (c == EOF) return;
```
c = getchar();
if (c == EOF) return;
putchar(c);
}
}

Question 2c

The program produces an incorrect return value of 0 when all elements of the array are negative. One fix would be to initialize currmax to the constant INT_MIN. Another option would be to assign currmax to a[0], and change the for loop to start at i=1.

Question 2d

At the end of the loop the array index i+1 is out of bounds. That error will manifest itself in a way that, from the application programmer's viewpoint, is unpredictable. The function could access bad data, or could generate a seg fault. Change i < n to i < n-1.

Some students noted the absence of an assert(a != NULL). It is true that it is better for testing to include the assertion, but it is not a bug per se -- it would be a defensive check for client code that wrongly passes a NULL pointer. Some students said that memory would be corrupted, but that is not true because the function does not write any data into a[n].

Question 3a

Reverses the characters of the string.

Question 3b

Converts a string of decimal digits to the corresponding integer.

Question 3c

Prints the last name, followed by a comma and space, and the first initial (e.g., "Turing, A").

Question 3d

Binary inputs ending in 011.

Question 4a

A declaration informs the compiler of the existence of a construct with a given name. A definition informs the compiler of the meaning of that name.

Question 4b

No. Changing stack.c does not imply changing stack.h, and so does not imply that client.c has changed, and so does not imply that client.c must be re-compiled.

Question 4c

The ADT uses void* so it is generic. The ADT uses const so the corresponding argument can be a pointer to a constant or an ordinary pointer. A common mistake was to say that const keeps the client from modifying the item, but it actually keeps the stack from modifying the item. Another common error was to say that this ensures the item remains unmodified throughout the program, but this is not true because the client part of the program can still modify the item.

Question 4d

So the client can assign the returned value to either an ordinary pointer or a pointer to a
constant.

**Question 4e**

The proposed alternative compares the addresses of the two Stack objects. It doesn't compare items of the two Stack objects.

**Question 4f**

```c
void *Stack_top(Stack_T s) {
    void *item;
    if (Stack_isEmpty(s))
        return NULL;
    item = Stack_pop(s);
    Stack_push(item);
    return item;
}
```

Note that it is not correct to simply call `Stack_pop(s)` and check whether the return value is the NULL pointer. This is because `Stack_pop(s)` can return a NULL for two reasons: (i) the stack is empty or (ii) the top item in the stack is a NULL pointer. Calling `Stack_isEmpty(s)` is necessary to differentiate between these two cases.