The exam was closed-book and closed-notes. It was a 50 minute exam.

**Question 1**

NOTE: answer key gives explanations, but only true/false really needed

**Question 1 Part a**
true - a number plus its one's complement is all ones

**Question 1 Part b**
true - even a tiny floating-point number is nonzero

**Question 1 Part c**
false - the left side is just the conversion of the octal number to decimal

**Question 1 Part d**
false - the 1 bitwise AND with 2 is zero, and the 3 bitwise AND with 4 is zero

**Question 1 Part e**
false - logical not of a nonzero is zero

**Question 2 Part a**
it produces zero, since the char is a single-byte data type, and it therefore gets the low-order byte of 0x5700, which is 0x00.

**Question 2 Part b**
the logical AND will produce zeros whenever either value is zero. However, the bitwise AND will produce zero in all of those cases, as well as a few others when the bits don’t line up, like 0xf0 AND 0x0f. So, the logical AND has a higher chance of producing nonzero values.

**Question 2 Part c**
int getint(int *hadEOF);
The integer read is the return value, unless the end of file is reached, in which case a nonzero value is returned in the space pointed to by hadEOF. Since we need to return all possible integers and an extra value, we cannot simply use the return value for both.

**Question 2 Part d**
return y->x != x;

**Question 2 Part e**
return streq(p[0], p[1]) && streq(p[1], p[2]);
**Question 2 Part f**

(a) this is not a good hash function because the hash is being kept in an unsigned char, limiting its range. Worse, each iteration basically discards the hash value and just replaces it with a new character from the string.

(b) the range of values is 0 to 255

(c) you could just return the last character of the string or zero if the string is empty

**Question 3 Part a**

checks that the first parameter exists, and then stores its length in k

**Question 3 Part b**

allocates space to hold a string twice as long as the input, and terminates the string

**Question 3 Part c**

copies the original string into the first half of the new space

**Question 3 Part d**

copies a reversed version of the original string into the second half of the new space

**Question 3 Part e**

creates a palindrome of it, and prints it out

**Question 4**

**Bug 1**
original statement: char counts[MAX VAL]; or MAX VAL = 1000
fixed statement: char counts[MAX VAL+1]; or increasing MAX VAL to 1001
explanation: the input is 0 to 1000 inclusive, which means that we have 1001 possible valid values. So we need to have one more space

**Bug 2**
original statement: scanf("%s"
fixed statement: scanf("%d"
explanation: we want to read in an integer, not a string, so we need %d

**Bug 3**
original statement: for (i = 1; i < c; i++)
fixed statement: for (i = 1; i <= MAX VAL; i++)
explanation: we need to go through all of the possible values, which are spread across the counts array. The original statement is only iterating as much as the number of input items, but is not checking where they actually are.

**Bug 4**
original statement: &counts[freak] in printf
fixed statement: counts[freak]
explanation: we want to print the value, so we don’t need a pointer to it

**Bug 5**
original statement: return;
fixed statement: return 0; or return EXIT SUCCESS;
explanation: should return a value of some kind

**Bug 6**
original statement: char i;
fixed statement: int i;
explanation: this counter is too small to iterate over the range, since a signed char is at most 127 and an unsigned char is at most 255. This needs to be at least a short, but an int is easier.