Remarks: Keep the answers compact, yet precise and to-the-point. Long-winded answers that do not address the key points are of limited value. Binary answers that give little indication of understanding are no good either. Time is not meant to be plentiful. Make sure not to get bogged down on a single problem.

PROBLEM 1 (36 pts)
(a) The code snippet
   ```c
   int a, *b, **c; printf("%d", a);
   ```
when compiled and executed prints the value of variable a (whatever it may be) to stdout. Add further code so that the value of a is output to stdout using variable b, and separately variable c, instead of a.

(b) The code snippet
   ```c
   int x, *y, *z; x = 8; *y = 9; z = &x; printf("%x", x); printf("%d", *y); printf("%d", *z);
   ```
is likely to encounter a segmentation fault. Where in the code (be precise) is segmentation fault likely to occur, and why? Why can we not be 100% certain that seg fault will occur?

(c) Suppose we want to use a 2-D array of type char to store up to 10 strings, each string of length up to 20 characters. Provide a declaration of a 2-D array that is just big enough to accomplish this goal. Write a call to printf() that outputs to stdout the third string. Code the call to scanf() that reads the first string from stdin. In both cases, use pointer notation instead of array notation. (Hint: Printing to stdout a 1-D array of type char is done through printf("%s", X) where X is the 1-D array. The same goes for scanf().)

PROBLEM 2 (32 pts)
(a) Suppose a function, readinput(), is called by main() to read from stdin 10 integer values using scanf() into 1-D array, int Y[10]. In one case, array Y is declared as global, in another as local to main(). Write the code of main() and readinput() when Y is declared global. Repeat the same when Y is declared local to main().

(b) Suppose x is a variable of type unsigned int and the goal is to inspect its i’th bit where i = 0 specifies the least significant bit, i = 31 the most significant bit. Code a function, int ibit(unsigned int x, int i), that accomplishes this goal by using a 32-bit mask and bit processing methods discussed in class. ibit() returns the value of the i’th bit.

PROBLEM 3 (32 pts)
(a) The function
   ```c
   void readfilename(void) { char f[20]; scanf("%s", f); }
   ```
reads from stdin a sequence of characters (e.g., a filename) and stores them into 1-D array f. Assuming the function compiles and is called by another function at run-time, what can potentially go wrong during execution? Suppose f is made global. Why is debugging more difficult when f is global than local? (Hint: We looked at a related example in class where gcc played an active role in helping detect silent run-time errors.)

(b) Modify the code of readfilename() in Problem 2(a) by calling getchar() in place of scanf() to fix the run-time bug. Assume the end of a filename entered on stdin is indicated by pressing the ENTER/RETURN key. Make sure that the file name is stored as a string—not just a sequence of characters—in f.

BONUS PROBLEM (10 pts)
A variation of Problem 2(b), suppose we wanted to flip the i’th bit of x so that 0 becomes 1, and 1 becomes 0. All other bits remain unchanged. Without providing actual code, describe in words how you would go about accomplishing this task using bit processing techniques. (Hint: Use the fact that performing XOR with 1 flips the value of a bit while XOR’ing with 0 keeps it unchanged.)