CS 240 (Park) **Midterm** **July 10 (Mon.), 2017**

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) What are the three steps performed by gcc per our discussion in class? In what way is the -c option useful when coding and updating large C programs? How does the *make* utility help programmers manage large C apps?

(b) For the code snippet

```
int s = 5, *t, **u; t = &s; u = &t; **u = 7; printf("%d", s);
```

explain what happens when we compile and execute the code.

(c) Given the code snippet

```
float a, *b, *c; a = 3.3; c = &a; *c = 5.5; printf("%f", a); printf("%f", *c); printf("%p", b); *b = 4.4;
printf("%f", *b);
```

explain happens when we compile and execute the code.

PROBLEM 2 (30 pts)

(a) Suppose x is a variable of type char, we read its value from stdin, and we wish to determine what its 5'th bit (starting from the least significant bit) is. First, describe in words the steps needed to carry out this task. Second, write a complete code of main() that performs this task.

(b) Explain the difference between the two declarations:

```
float *f(int), (*g)(int);
```

How can we call function f with integer argument 5 by using g? Write the complete code of a function h that uses a function pointer and integer passed as arguments to run f on behalf of the calling function main(). h returns to main() what f would have returned had it been directly called by main().

PROBLEM 3 (34 pts)

(a) In Problem 9 of lab2, you encountered a stack smashing run-time error when running a for-loop that iterates on a 1-D array declared as int a[5]. Explain what happened. Without gcc's assistance, how can a segmentation fault arise? What is a potentially worse outcome than segmentation fault stemming from the stack smashing bug? Why can gcc not guarantee that stack smashing is always detected before it causes further harm?

(b) Describe how a 2-D array, int d[3][3], is organized in main memory (i.e., "RAM") by using a drawing as discussed in class. Suppose we store the values 1 2 3 in the first row of d[][], 4 5 6 in its second row, and 7 8 9 in the third row. Which values do *(*d+2) and *(*(d+1)+2) refer to, and why? Suppose d[][] is declared as local to main() and main() calls a function read_array() to read the values of d[][] from a file (similar to Problem 4 in lab3). Assuming read_array() is of type void (it does not return anything), specify the declaration of read_array() with respect to its argument and how main() should call read_array() so that the values read from a file are stored in main()'s local data structure d[][].

BONUS PROBLEM (10 pts)

When performing file I/O, if opening a file failed we terminated the program by calling exit(1) instead of executing return. How are they different, and what is gained by doing the former?