PROBLEM 1 (42 pts)

(a) When C functions call each other, they may pass arguments by value or reference. In what situation is passing by value preferred and when is passing by reference more adequate? For the stdio function printf(), is it feasible to rewrite it such that passing by reference is used? Discuss your reasoning.

(b) When a C function is called at run-time (i.e., when an app runs), a stack frame is created for the function which acts as a “scratch space.” What are the two principal uses of the stack frame discussed in class? What happens to the stack frame when the function returns to its caller? In the function

```c
int somefunc(void) { int x = 0; static int y; y = x + 1; return y;}
```

where in memory are the spaces for x and y allocated? Suppose main() calls somefunc() twice in succession. What returned values will it see?

(c) Suppose we have a 1-D array declared as int z[5]. How is the array organized in memory? Why is the statement *(z+2) = 37 equivalent to the statement z[2] = 37? Suppose you want to read a list of five integer values into the array z using scanf(). What is a correct way to pass as argument to scanf()?

PROBLEM 2 (30 pts)

(a) Consider the following snippet of code:

```c
int x, *y; x=5; y=&x; y=10; *y = 15;
```

What is the value of x after the last statement? Explain what is going on.

(b) Explain the difference between char *myfunc(int, int) and char (*myfunc)(int, int). Provide snippets of main() code that shows how to call the two functions correctly.

(c) Suppose we have the code:

```c
int x; main() {
    void abc(void); x = 55; abc(); printf("%d",x);
}
void abc(void) {
    int x; x = 77;
}
```

contained in the same file mycode.c. What value of x is printed by main() and why? Will running gcc on mycode.c compile correctly?

PROBLEM 3 (28 pts)

(a) Suppose we have the following code snippet:

```c
float *a, b[3]; *a = 10.2; b[3] = 5.5; a = &b;
```

Why does each of the three statements following the variable declaration result in a bug? Are all three run-time bugs or can some be detected at compilation?

(b) For the following code

```c
main() { int x, y; unsigned m; scanf("%d",&x); m = ~(~0 >> 1); y = x & m; if (y != 0) printf("neg"); }
```

explain, line-by-line, what main() does. Assume that right-shift fills the most significant bit with 0. What is an alternative (and a tad simpler) expression for creating the mask m?

BONUS PROBLEM (10 pts)

How is a 2-D array float h[3][5] organized in memory? Suppose we set h[1][2] = 12 and h[2][1] = 21. Using only pointer arithmetic (i.e., without using square brackets), rewrite the above two statements. (Hint: Recall that h[0][0] is equivalent to **h.) Given that array declarations and their manipulation (using square brackets) is straightforward, what is a practical motivation for declaring 2-D arrays as pointers and them using pointer arithmetic?