CS240 Final Solution, summer 2022

P1(a) 15 pts

Problematic:
amigo->year = 2017
4 pts

strcpy(amigo->nickname, "fish")
4 pts

Augmentation:
amigo = (friend_t *) malloc(sizeof(friend_t)); // Type casting (friend_t *) can be omitted.
4 pts

amigo->nickname = (char *) malloc(strlen("fish")+1); // (char *) can be omitted.
// malloc(5) is fine too.
3 pts

P1(b) 15 pts

fun1 takes as argument a pointer to char and returns a pointer to char.
4 pts

fun2 is a function pointer to a function that takes as argument a pointer
to char and returns a value of type char.
4 pts

```
char fun3(char *s) {
  while(*s != '\0')
           s++;
  return *(s-1);
}
```
7 pts

P1(c) 15 pts

Input "/bin/cp file1 file2" is read from stdin.
4 pts

main(int argc, char *argv) of /bin/cp accesses the two file names via argv[1] and argv[2].
4 pts

Assuming a variable s is of type, char **s, a shell must allocate sufficient memory for
s and copy "/bin/cp" into s[0], "file1" into s[1], "file2" into s[2], and set s[3] to NULL.
4 pts

execv() is called as execv(s[0], s).
3 pts

P2(a) 15 pts

```
unsigned int countdbl(long x) {
int i;
unsigned int count = 0;
long m = 1;
// 3 pts

  for(i=0; i<64; i++) { // Check all bits of long value from lsb to msb.
        if((x & m) == 0) count++;
        x = x >> 1;
  }
  // 6 pts

  if((count & 1) == 0) return 0; // Check if count is even.
        else return 1;
  // 6 pts
}
```

P2(b) 15 pts

When a function is called by another function, gcc tries to detect if the
return address has been corrupted and, if so, terminate the running program.
This is to prevent the code from jumping to unintended code such as malware.
4 pts

A local variable of a function declared as a 1-D array overflows by input
whose length is not checked when reading from stdin (or file).
4 pts

Example: a function contains code
char buf[100];
scanf("%s", buf);
which may overflow buf[] since scanf() does not check for length of the input.
4 pts

Sound practice: use functions to read from stdin (or file) that check for length.
In the above example use fgets() instead of scanf().
3 pts

P3 25 pts

```
double multnums(char *a, ...) {
int x;
double y, val = 1;
va_list arglist;
// 4 pts

  va_start(arglist, a);
  // 2 pts
  while(*a != '\0') { // Check the format string, character by character until EOS.
  // 3 pts
            if (*a == 'd') { // Interpret argument as int.
              x = va_arg(arglist, int);
              val = val * x;
            }
            // 6 pts
            else { // Assumes must be 'f' since forgoing error checking.
              y = va_arg(arglist, double); // Interpret argument as double (not float).
              val = val * y;
            }
            // 6 pts
            a++;
            // 2 pts
  }
  va_end(arglist);
  // 2 pts
  return val;
}
```

Bonus 10 pts

Step 1: Open file, read byte by byte until EOF is reached while counting occurrences
of '\n' to determine the total number of lines (count plus 1). Denote this number of r.
Close file.
2 pts

Step 2: Use malloc() to allocate 1-D array, int *M, of size r of type int. Open file,
read byte by byte, counting for each line the number of bytes. Store the line length
in 1-D array M. Close file.
3 pts

Step 3: Using 1-D array M call malloc() for each line to allocate memory to store
the bytes of each line. Point x to the 1-D array of pointers to char.
3 pts

Step 4: Open file. Read byte by byte the content of each line into 1-D array of pointers
to char pointed to by x.
2 pts