P1(a) 12 pts

b = &a;
printf("%d", *b);
6 pts

c = &b;
b = &a;
printf("%d", **c);
6 pts

P1(b) 12 pts

At statement *y = 9.
4 pts

The reason is that y is a pointer and can contain an address. The code has not stored
a valid address in y, hence following that address using *y is likely to result in
accessing a location that is not allowed.
4 pts

Since we did not assign an address to y, it is unlikely but possible that whatever
value resides in y happens to be a valid address by chance.
4 pts

P1(c) 12 pts

char A[10][21];
// second dimension is 21 since we need space to stored EOS '\0' to make it a string
4 pts

printf("%s", *(A+2));
4 pts

scanf("%s", *A);
4 pts

P2(a) 16 pts

#include <stdio.h>
int Y[10];
void readinput(void); // can be declared inside main()

int main() {
    readinput();
}

void readinput() {
    int i;
    for(i=0; i<10; i++)
        scanf("%d", &Y[i]);
}
8 pts

#include <stdio.h>
void readinput(int *); // can be declared inside main()

int main() {
    int Y[10];
    readinput(Y);
}

void readinput(int *Z) {
    int i;
for(i=0; i<10; i++)
    scanf("%d", &Z[i]);
}

8 pts

P2(b) 16 pts

int ibit(unsigned int x, int i) {
    unsigned int m, y;
    m = ~(~0 << 1);  // m = 1 will also work
    y = x >> i;
    return y & m;
}

16 pts

P3(a) 16 pts

If the user provides a string that is longer than 20 characters, 1-D array will overflow.
8 pts

As we saw in class, when f is local, gcc (by default in our lab) inserts code (canary) to help detect overflow of f which otherwise may cause silent run-time error. When overflow is detected, the running program is terminated which helps discover the run-time error. If f is global, then gcc does not insert such code, hence detecting the resultant silent run-time error becomes more difficult.
8 pts

P3(b) 16 pts

void readfilename() {
    char f[20], c;
    int i = 0;
    while((c = getchar()) != '\n') {
        if(i == 19) {
            printf("input too long\n");
            exit(1);
        }
        f[i++] = c;
    }
    f[i] = '\0';
}

16 pts

Bonus 10 pts

Step 1. Set up a mask m so that its i'th bit is 1 and all other bits are 0. To accomplish this, set m to 1 (first bit is 1 and all other bits are 0), then shift it left by i positions.
6 pts

Step 2. Perform bit-wise XOR of x and m.
4 pts