CS240 Midterm Solution, summer 2025

P1(a) 16 pts

Run-time bug: *y = 5
6 pts

Segmentation fault (or violation).
4 pts

y will likely contain an address that does not belong to the running program;
trying to access such an address causes a segmentation fault.
6 pts

P1(b) 16 pts

1 (the value of x) // but not the value of y.
7 pts

Even though printf("%d", y) would have executed correctly, the value of y (2)
is temporarily held in main memory and before it can be flushed to the display
statement *z = 3 causes a segmentation fault. Thus the value of y is not seen
on the display.
9 pts

P1(c) 16 pts

s is a pointer that holds the beginning address of 10 contiguous locations
where 10 integers may be stored. *s dereferences (i.e., follows the address
contained in s), hence its value is equivalent to s[0] (the content at the
first of the 10 contiguous locations).
10 pts

By precedence of operation * over operation +, *s + 2 is equivalent to
(*s) + 2 which is equal to s[0] + 2.
6 pts

P2(a) 17 pts

If the input entered on stdin is too long scanf("%s", v) will cause overflow
of 1-D char array v[10]. Since v[10] is local to main() this may overwrite
and corrupt the canary that gcc (by default) placed below the return address
of main.
5 pts

Since the code added by gcc checks if the canary has changed before executing
return, the corrupted canary will cause a stack smashing message to be
output before terminating the running program.
5 pts

printf() will likely succeed since the return address of printf() to its
caller main() has not been affected by overwriting v[10].
3 pts

Change scanf() to scanf("%9s", v) so that no more than 9 characters are read
(10th character for EOS).
// Allow scanf("%10s", v) without point deduction.
4 pts

P2(b) 17 pts

fgetc() needs to be able to signal the caller that the end of a file has

been reached which is accomplished by returning EOF (i.e., -1). If the return
type were char then all 8 bits would be needed to communicate the 8-bit
content of a data byte read from a file. This would leave no room to
notify the caller that the end of file has been reached through the return
value.
4 pts

```
int x;
while((x = fgetc(fp)) != EOF) {
4 pts
  if(x > 127) {
          printf("Not ASCII text file.\n");
          exit(1);
          // exit(0) and return are fine too.
  }
  4 pts
}
printf("ASCII text file.\n");
2 pts
```

Read byte-by-byte until EOF. If a byte value exceeds 127 (i.e., most significant
bit is not 0) then not ASCII character, thus file is not ASCII text. Only if
all bytes of the file are ASCII is it an ASCII text file.
3 pts

P3 18 pts

```
// set mask to 0x00000001
int m;
m = 1;
// unsigned int for m is fine.
5 pts

int y;
// unsigned int for y is fine.
int x;
while((x = fgetc(fp)) != EOF) {
  y = x >> 7;                // Move 8'th bit to first bit position.
  6 pts
  if((y & m) == 1) {         // Not ASCII since 8'th bit is 1.
                             // (y & m) != 0 or equivalents are fine too.
          printf("Not ASCII text file.\n");
          exit(1);
          // exit(0) and return are fine too.
  }
  7 pts
}
printf("ASCII text file.\n");
```

Bonus 10 pts

11 and 13
6 pts

Since r is static its value will be remembered across function calls. Since
return r++ returns the current value of r (11) first before incrementing it
the first value is 11. At the second call, r starts out 12 due to increment
after first return, then statement r++ increases it to 13 before returning
to caller.
4 pts