P1(a) 12 pts
When source code changes are performed on some files but not abc.c, there is no need to recompile abc.c which saves time.
4 pts
xyz.o must be linked with abc.o.
4 pts
make keeps track of time stamp information which shows that abc.c has a more recent update time than abc.o. It assumes the content of abc.c has changed.
4 pts
P1(b) 12 pts
&t = 3 assigns integer 3 to s since t is assigned the address of s. Therefore The first printf() outputs 3.
6 pts
The assignment *u = 5 is likely to result in segmentation fault since the address contained in u may not be valid.
4 pts
The second printf() is likely never reached due to segmentation fault.
2 pts
P1(c) 12 pts
mm() takes a single argument of type double and returns a value of type float.
2 pts
nn() takes a single argument of type double and returns a pointer (or address) to float.
2 pts
pp is a function pointer (i.e., contains an address to a function) to a function that takes a single argument of type double and returns a value of type float.
2 pts
nn = &mm is invalid
2 pts
pp = nn and pp = &nn are valid assignments but likely incorrect in that their return values (nn returns pointer to float and pp's function returns float) do not match up w.r.t. type.
2 pts
pp = &printf() is a valid assignment but likely incorrect since their arguments do not match up.
2 pts
P2(a) 16 pts
Array s[] is initialized to the string "hi".
2 pts
h = s makes h point to the string "hi".
2 pts
The first printf() outputs the string "i" since h+1 is the address of the second byte of "hi".
3 pts
The second printf() outputs the address stored in s.
3 pts
The third printf() outputs the character 'h' since \*h is the content at the first byte pointed to by h.
3 pts
By definition, a string in C must terminate with '\0' (i.e., EOS) which requires an additional byte of storage.
3 pts
P2(b) 16 pts
ddd contains an address that points to three consecutive addresses. Each of the three consecutive addresses is a 1-D array of three float values. That is, each points to memory where in three consecutive locations three float values can be stored.
2 pts
ddd+1 is the second of the three addresses that ddd points to. Hence it's the second row (1-D array) ddd[1][0], ddd[1][1], ddd[1][2].
4 pts
*(ddd+1) follows this address which leads to where ddd[1][0] is stored.
4 pts
Adding 2 leads to the third float in the 1-D array which is ddd[1][2].
4 pts
The second star specifies the content at the above address which is 1.2.
2 pts
P3(a) 16 pts
0 is returned all three times.
3 pts
Since a is local variable of bbb(), it is initialized to 0 each times bbb() is called and becomes invalid when bbb() returns.
2 pts
As we discussed in class, a++ increments a after the return value is set hence it remains 0.
2 pts
When variable a is declared as static int, its memory is not allocated in the stack frame of bbb() but similar to where global variables are allocated. Hence when bbb() returns the content of a is preserved.
3 pts
As a result, three consecutive calls to bbb() returns the values 0, 1, 2.
2 pts
When the variable a is declared as global, it has the same effect as declaring it as static. The only difference in the latter is that the variable name a can be reused by other functions when declaring local variables.
2 pts
When the variable a is declared as global, it has the same effect as declaring it as static. The difference is that in the static case, the variable name a can be reused by other functions as a local variable.
2 pts
P3(b) 16 pts
int main(void)
{

int *s;
  *s = 5;
}
6 pts

Memory for s has been allocated which can hold an address that contains a value of int. However, the address stored in s has not been assigned. As a result, what is contained in s is likely to be invalid which leads to segmentation fault.

6 pts

int main(void)
{
  int *s, t;

  s = &t;
  *s = 5;
}

s is assigned a valid address (the address of t which can store a value of int), hence *s = 5 will not trigger a segmentation fault.

4 pts

[This is but one of many possible solutions. They all count as equal as long as they contain a valid example and explanation of what is going on.]

Bonus 10 pts

Step 1:
Set up a mask that is 0 everywhere except in the first (least significant) bit position.
3 pts

Step 2:
Shift the unsigned int to the right by 19 positions so that the 20th bit becomes the first bit.
4 pts

Step 3:
AND the shifted data with the mask and print the result.
3 pts