Problem 1:

1a) "f1" is on the stack because it is a function call.
"i" is in data because it is a static variable.
"ws" is in data because it is an initialized global variable.
"z" is in bss because it is an unitialized global variable.

1b) This program outputs:

10

11

11

12

→ This is the output because the first time f1() is called, i = 10 and then "10" is printed to STD_OUT and i is incremented to 11. When fork() is called inside the if statement, a child process begins. The child process calls the f1() call that is inside the if statement. i is a static variable, so in this call to f1, i maintains its value of 11 that was inherited by the child process from the parent process, and "11" is printed to STD_OUT. Then, in the child process, i is incremented to 12. Simultaneously, the parent process calls the f1() function call outside the if statement and because i preserves its value of 11, "11" is printed to STD_OUT. This is why there are two "11"s outputted, but we do not know which one is from the parent process and which one is from the child process. The parent process then waits for the child to terminate before returning while the child process calls f1() again after the if statement and here, i maintains its value of 12 from within the child process and "12" is printed to STD_OUT. When the child process encounters wait, it returns "-1" because it has no child processes of its own, after which the parent process continues because the child process terminated. The parent process then returns.

1C) The output is <u>not</u> deterministic because it is not known whether after the fork system call completes, the parent runs first, the child runs first or both processes run simultaneously. However, in this example, the out put is the same regardless.

1d) The output of "echo $?" is "255". This is because when the child process encounters the wait system call, it fails because the child process has no child processes. As a result, wait returns -1, and the program calls exit(-1). Because exit status codes fall within 0-255, "-1" becomes "255".

<u>Problem 2:</u>

2A) The output will always be "XY" <u>with certainty</u>. This is because buf is an unitialized global variable which is stored in bss, so it automatically gets initialized to 0. Therefore, "X""buf""Y" just outputs XY.

2B) "/usr/bin/python" is the program that next occupies the address space and the argument vector is: [-B, "/tmp/osps3.py", "osps3.py", "myinput.txt", NULL].

2C) When execvp is called, the colon-separated list of directory pathnames specified in the path environment variable is searched for the file (i.e. the first argument of execvp).