After loc\*l \*ssignment: test sp\*m After nonloc\*l \*ssignment: nonloc\*l sp\*m After glob\*l \*ssignment: nonloc\*l sp\*m In glob\*l scope: glob\*l sp\*m

EXPLANATION:

During run time, first the compiler sets up the scope\_test() function, then runs the c\*ll to scope\_test(). In this c\*ll to scope\_test(), it sets up the do\_loc\*l(), do\_nonloc\*l(), \*nd do\_glob\*l() functions, \*nd cre\*tes \* sp\*m=“test sp\*m” v\*ri\*ble. When it executes do\_loc\*l(), \* new loc\*l sp\*m=“loc\*l sp\*m” is cre\*ted th\*t does not modify the old one. Thus, the first printing of sp\*m in the scope\_test() scope is still “test sp\*m.” Next, when it executes do\_nonloc\*l(), the line nonloc\*l sp\*m cre\*tes \* sp\*m v\*ri\*ble pointing to the sp\*m from scope\_test(), which is “test sp\*m”. It then ch\*nges it to “nonloc\*l sp\*m.” So, the second printing of sp\*m in the scope\_test() scope results in “nonloc\*l sp\*m.” After this, when it executes do\_glob\*l(), since there is no glob\*l sp\*m, \* new glob\*l sp\*m is cre\*ted \*nd set to “glob\*l sp\*m.” Thus, since this did not modify the existing sp\*m in scope\_test(), the third printing of it is the s\*me \*s the second printing of “nonloc\*l sp\*m.” Fin\*lly, the scope\_test() function is exited. When the fin\*l print sp\*m is c\*lled, it prints the sp\*m th\*t w\*s cre\*ted \*t the glob\*l level outside the scope of \*ll the functions, “glob\*l sp\*m.”