1. (10) This question uses a notional C-like language, rather than an actual programming language. The intent is to get you to demonstrate your knowledge of type checking, not worry about a minor syntactic nit. Consider the following code fragments ...  
   Text

   Description automatically generated  
   What types are allowable for variables x, y, and s? If no legal typing can exist, modify the code to allow a legal typing and then specify the legal types.

ASSUMING THAT USER INPUTS ARE LIMITED TO ONLY BEING ABLE TO BE INTEGERS (such as by user inputs being done through a number pad like on an ATM machine) s must be an int. Granted, most likely this is just taking any user input which most likely we would want to be a string and as such s should actually be a string and we should modify foo to be “foo(string m, string n)”. Doing this shouldn’t cause any issues as the only other piece of code that would be affected would be the makeString() call, but it says to assume that no matter what the parameter type is, assume it’s converted to a string.

x and y are required to be strings. This is as such because the function foo() specifies types that must be inputted and returned and setting x equal to it means that x must return a string and so on for y and s.

Basically, to simplify it all, x, y and s should all be initialized as type “string”, and we should adjust the foo() method to just take two string parameters.

1. (10) The lecture slides for denotational and operational semantics discussed how the semantic analysis of programs arises from the semantic analysis of statements. They did this by providing a rule for how two sequential statements are treated in the analysis. Create a rule for two sequential program statements for axiomatic semantics.

(Not really sure that I understand what exactly is being asked here but I will try and explain how it works as I understand.) An axiomatic system is based on the idea of sequentially confirming the truth/validity of statements by using basic unarguable statements, axioms, to describe statements. Given two sequential statements “s1;s2”; s1 and s2 have axioms to describe basic truths involved before they are executed (such as variables used in a method are initialized properly), the axioms that s1 and s2 are in fact program statements, and finally the new axioms for each statement after execution of each statement (such as variables having their new values). Basically, for s1 and s2, if the initial axiom (precondition) is true and the program successfully terminates, then the ending axiom (postcondition) must also be true.

1. (10) My best friend Bob says he has developed a language that changes type checking from a semantic to a syntactic check. His language requires that every variable name be prefaced with its type. So 'x' becomes 'intx' or 'doublex' (depending on the type) and so on. Justify why this is or is not sufficient to allow type checking to be performed in the lexing/syntax compilation phase.

As I understand it now, theoretically, this should be sufficient to allow type checking to be performed in the compilation phase. I believe this is true because Java is a language that does type-checking during compile-time and basically looks the same as the above example initialized variables. The bigger issue seems to be that you may run into syntax issues but honestly assuming the compiler detects the type and is able to internally separate the type declaration from the rest of the characters used to name variables, it would work identically to java.