1. Consider an imperative language similar to IMP that has C-like pointers, as follows. The syntax for expressions is:

$$e \ ::= \ n \ | \ \mathsf{true} \ | \ \mathsf{false} \ | \ x \ | \ \&x \ | \ *x \ | \ \mathsf{malloc}(t) \ | \ \mathsf{null}$$

   Expressions include integers, booleans, and pointers. Expression $\&x$ represents a pointer that points to x. Expression null denotes a null pointer. The dereference $*x$ requires x to hold non-null pointer; then, $*x$ represents the variable that x points to. Finally, $\mathsf{malloc}(t)$ allocates a new heap cell for a value of type t, and returns a pointer to that cell. The commands are the same as in IMP, except for the assignment:

$$c \ ::= \ ... \ | \ e_1 := e_2$$

   The assignment requires e_1 to be an l-value: either a variable, or a dereference of a non-null pointer. It then updates the variable that e_1 represents with the value of e_2.

   (a) What are the values in this language?

   (b) Indicate the form of the small-step operational evaluation relations for this language.

   (c) Write all of the evaluation rules for expressions and assignments in this language.

   (d) Write appropriate typing rules for all expressions in the language. Your typing rules must enforce the absence of all type errors, except null pointer dereference errors.

2. Consider the subtyping rule for function types.

   (a) If we change the rule such that both the arguments and the return values are co-variant, the rule is unsound. Show the modified subtyping rule and write a program that type-checks in the presence of this rule, but yields run-time errors.

   (b) If the arguments and the return values are both contra-variant, the rule is again unsound. Write a program proving that this rule would not be safe.

   For each question, indicate where you use the modified subtyping rules, and where the run-time error occurs.