The language used in problem 1 is similar to the extended lambda-calculi considered in class and homework 3. The key addition is access modifiers on record fields; see point 6.

1. Variables, function application, integers, conditionals, and addition are just like in homework 3. (Type-checking conditionals in a language with subtyping is interesting; see the problem description and the extra credit.)

2. Lambdas are like in class except we simplify type-checking by requiring a type annotation that is the type of the argument.

3. Closures use an env ref instead of an env just to make Letrec easier to interpret. You do not need to understand this for the homework, but look at this case of interp if curious.

4. Letrec(t1,f,x,t2,e) describes a recursive function named f with body e, argument name x, return type t1 and argument type t2. (So the body assumes f if a function from t2 to t1.)

5. Records are a collection of mutable fields. Strings are field names (the type-checker ensures they are unique). A RecordE appears in source programs and is not a value (its fields must be evaluated). A RecordV is a value (and all its fields are references to values). The Get operation gets the contents of a record-value field and the Set operation changes the contents. The type-checker can raise an exception if it encounters a RecordV. Type-checking does not need mutation.

6. A record type lists the fields, their types, and access modifiers. A Get e operation with field l type-checks only if e has a record type that includes field l and where l’s access is not Write (i.e., write-only). Similarly, a Set e operation with field l type-checks only if e has a record type that includes field l and where l’s access is not Read (i.e., read-only). A RecordE has a type where every field has access Both — explicit type annotations (in casts and functions) can have types with other access modifiers. The relevant subtyping is for you to figure out as described in the problem statement.

7. A cast allows only casts to supertypes and has no run-time effect. It type-checks if the type of the expression is a subtype of the explicit annotation.