Recall the simple typed language E52 from homework 5, which we’ll now call “E6.” An E6 interpreter in Java (only) with fully working typechecker has been provided (`hw6_1.java`).

Modify the interpreter to support a new language feature: binary disjoint sum (discriminated union) types. The syntactic extensions are as follows:

```
typ := ...  
    | '(' 'Sum' typ typ ')'  
exp := ...  
    | '(' 'inleft' typ exp ')'  
    | '(' 'inright' typ exp ')'  
    | '(' 'case' exp var exp var exp ')'  
```

The type `Sum t_l t_r` is a sum type with two tags, identified as “left” and “right;” the value carried with the left tag is of type `t_l` and the value carried with the right tag is of type `t_r`.

The injection expression `(inleft t e)` creates a value of type `t` (which must be a `Sum` type) tagged “left” and carrying the value of `e`. Similarly, `(inright t e)` creates a value tagged “right.” (Note that it is necessary to give an explicit type parameter to these expressions because the full sum type cannot be inferred from the type of `e`.) The expression `(case e v_l e_l v_r e_r)` is evaluated as follows. First `e` is evaluated; the result must belong to a sum type. If the result is tagged “left” then the current environment is extended by binding the associated value to `v_l` and `e_l` is evaluated in the resulting environment, yielding the result of the entire case. Similarly, if the result is tagged “right” then the associated value is bound to `v_r` and `e_r` is evaluated to yield the case result.

Also, `write` should work on `Sum` values; for example, the value corresponding to `(inleft 42)` should display as “(L 42)” and similarly for `inright`.

Here are the typing rules for the new expressions:

```
\frac{TE \vdash e : t_l \quad t = (Sum t_l t_r)}{TE \vdash (inleft t e) : t} \text{ inleft}
```

```
\frac{TE \vdash e : t_r \quad t = (Sum t_l t_r)}{TE \vdash (inright t e) : t} \text{ inright}
```

```
\frac{TE \vdash e : (Sum t_l t_r) \quad TE + \{v_l \mapsto t_l\} \vdash e_l : t \quad TE + \{v_r \mapsto t_r\} \vdash e_r : t}{TE \vdash (case e v_l e_l v_r e_r) : t} \text{ case}
```

To give more intuition about these expressions, suppose we had an ML type definition
datatype ('a,'b) sum = Left of 'a | Right of 'b

Then inleft and inright play the roles of the constructors Left and Right, and (case e v1 e1 v2 e2) is similar to

    case e of
        Left v1 => e1
    | Right v2 => e2

The necessary parsing support for this extension is already present in hw6_1.java; you just need to uncomment it. You’ll need to add a new subclass SumTyp of Typ, a new subclass SumValue of Value (and perhaps a new function sumValue) and the AST, printing, and evaluation code for the new expression forms.

Put your modified interpreter into a file called sol6_1.java and submit it.

2. Recall the ML interpreter for language E51 in assignment 5. It makes use of a structure Env that implements an abstract data type (ADT) for environments. The signature of this ADT is given between the keywords sig and end:

    structure Env :>
        sig
        type 'a env
        exception NotFound of string
        val empty : 'a env
        val extend : 'a env -> (string * 'a) -> 'a env
        val lookup : 'a env -> string -> 'a
        end =
        struct
            ...
        end

Your task is to write a new structure implementation (replacing the ... ) that implements the same functionality as the existing Env, but uses (unbalanced) binary search trees rather than lists. (Consult your favorite algorithms text for more details about binary search trees if you need them.) Use the following definition of trees

    datatype 'a env = LEAF
        | NODE of string * 'a * 'a env * 'a env

Do not alter the existing Env interface and don’t accidentally alter the behavior of the operators. In particular, remember that if an environment is extended twice with the same identifier, the more recent extension “hides” the previous one.

Put your new structure definition (only) into a file sol6_2.sml and submit it. Note: You may wish to test your solution by using it in the context of the hw5 interpreter, but don’t submit the interpreter code – just the structure definition.