The slip data structure introduces the name \textit{slip} with the following axioms:

\begin{align*}
\text{slip} &= [X; \text{slip}] \\
B &= [X; B] \Rightarrow B: \text{slip}
\end{align*}

where \( X \) is some given type. Can you implement it?

After trying the question, scroll down to the solution.
That second axiom is not induction; it is coinduction, defining *slip* to be the largest solution of the construction axiom. (If it were induction, the two axioms would define *slip* to be *null*.) If lists and recursive definition are implemented, as they are in some “lazy functional” languages like LazyML and Haskell, then *slip* is already implemented by the first axiom. It's strange because the recursion doesn't seem to have a base, so *slip* is an infinite structure:

\[ slip = \left[ X; [X; [X; \ldots \right) \right] \]

In C we have to use pointers.

```c
struct slip { X left; slip *right; }
```

Although recursive data types are seldom implemented, recursive functions usually are implemented. (This is a strange inconsistency in the design of programming languages; the reasons for recursion and the implementation of recursion are exactly the same for data types as for functions and procedures.) We can define

\[ slip = 0 \rightarrow X \mid 1 \rightarrow slip \]

or

\[ slip = \langle n: 0,1 \mid \text{if } n=0 \text{ then } X \text{ else } slip \text{ fi} \rangle \]

This function definition will be a problem in a language that wants you to state the result type. The number of further arguments depends on the values of previous arguments.