```
// ---------------------------------------------------------------
//        Homework 5
//        Adam Levy, Alejandro Palacios, & Alicia Rodriguez
//        November 23, 2016
// ---------------------------------------------------------------

// This sets F# to read from whatever directory contains this source file.
System.Environment.set_CurrentDirectory __SOURCE_DIRECTORY__;;
#load "parser.fsx"

// Refer to "Parser.Parse.parsefile" simply as "parsefile",
// and to constructors like "Parser.Parse.APP" simply as "APP".
open Parser.Parse

// e is the body, x is the term to substitute, t is the substitution term
let rec subst e x t= match e with
                     |ID y              -> if x = y then t else ID y
                     |APP (e1, e2)      -> APP ( subst e1 x t, subst e2 x t)
                     |IF (e1, e2, e3)   -> IF (subst e1 x t, subst e2 x t, subst e3 x t)
                     |FUN (y, e1)       -> if x = y then FUN (y, e1) else FUN (y, subst e1 x t)
                     |REC (y, e1)       -> if x = y then REC (y, e1) else  REC (y, subst e1 x t)
                     |e                 -> e // Catch all

let rec interp = function
    | APP (e1, e2) ->
        match (interp e1, interp e2) with
        | (ERROR s, _)          -> ERROR s              // ERRORs are propagated
        | (_, ERROR s)          -> ERROR s
        | (SUCC, NUM n)         -> NUM (n+1)            // Rule (6)
        | (SUCC, v)             -> ERROR (sprintf "'SUCC' needs INT argument, not '%A'" v)
        | (PRED, NUM 0)         -> NUM 0
        | (PRED, NUM n)         -> NUM (n - 1)
        | (PRED, v)             -> ERROR (sprintf "'PRED' needs INT argument, not '%A'" v)
        | (ISZERO, NUM 0)       -> BOOL true
        | (ISZERO, NUM n)       -> BOOL false
        | (ISZERO, v)           -> ERROR (sprintf "'ISZERO' needs INT argument, not '%A'" v)
        | (FUN (x, e), v1)      -> interp (subst e x v1) // Rule (10)
        | (REC (x, f), e)       -> interp (APP (subst f x (REC (x, f)), e)) // Rule (11)
    | IF (b, e1, e2) ->     // Rule (4 & 5)
        match (interp b, e1, e2) with
        | (ERROR s, _, _)       -> ERROR s
        | (_, ERROR s, _)       -> ERROR s
        | (_, _, ERROR s)       -> ERROR s
        | (BOOL true, e, _)     -> interp e
        | (BOOL false, _, e)    -> interp e
        | (b, _, _) -> ERROR "'IF' needs BOOL expression."
    | e -> e // Catch all case NUM -> NUM, BOOL -> BOOL, FUN (x,e) -> FUN (x,e)  etc..

// Two convenient abbreviations for using the interpreter
let interpfile filename = filename |> parsefile |> interp
let interpstr sourcecode = sourcecode |> parsestr |> interp
```

This handling of REC more or less works, but it is not what rule (11) specifies.
Also, you need a default ERROR case for things like (17 true), where a non-function is being applied.

You are missing cases for ID and REC, which do NOT evaluate to themselves!