Problem 1

A macro declaration consists of a template specification and the definition of a replacement string. Each occurrence of the template in the scope of the declaration is replaced by the string. This process is called macro-expansion.

Assume that your favorite lexically scoped language provides macros and functions. For instance, it is possible to implement a bounded increment operation as either a macro or a function. Note that our example language allows conditional expressions.

Program probl0
integer max = 100; // variable max
macro incr-macro(x)
    (if x > max then max
        else x+1)
function incr-function(x)
    (if x > max then return max
        else return x+1)
procedure test ...

begin // program
call test
end // program

1. Write procedure test that produces different results for the macro and function implementation of the increment operation.

2. Discuss the general benefits and draw-backs of macros and functions.

Problem 2

1. Assume that your favorite lexically scoped language allows you to pass procedures and functions as actual parameters (downward exposure). Sketch an implementation based on access links and an implementation based on displays that will ensure correct access of local and non-local variables.

2. Assume that your favorite lexically scoped language allows you to pass procedures and functions as actual parameters (downward exposure) and return function or procedure values
(upward exposure). In addition, assume that the lexically scoped language has only two
levels, namely local and global, i.e., procedure and function declarations can not be nested.
Sketch an implementation that will ensure correct access of local and global variables.

**Problem 3**

Although you have been using your favorite lexically scoped programming language for some time,
you forgot which single parameter passing style the language is using. You know that it must be
one of the following: call-by-value, call-by-reference, copy-restore, or call-by-name (see Aho, Sethi,
Ullman Section 7.5).

Write a procedure `test` that allows you to determine which parameter passing style is used based
on a single call to `test`. The procedure `test` should not have more than 3 formal parameters.

An answer to this problem contains the procedure `test` in pseudo code, the actual parameters of the
single call to `test`, and the conditions that have to hold after the call that uniquely identify each of
the four different message passing styles.

Here is a coarse template for your answer:

```plaintext
program what-passing-style
integer a, b, c  // declare as many variables or functions as
                 // you need

... call test(a, b, c)

if (a = ? and b = ? and c = ? and ...) then
    write(call-by-value)

if (a = ? and b = ? and c = ? and ...) then
    write(call-by-reference)

if (a = ? and b = ? and c = ? and ...) then
    write(copy-restore)

if (a = ? and b = ? and c = ? and ...) then
    write(call-by-name)
```