Sample Solution

Assigment 6

CSCI 3136 — Winter 2018

Question 1

The two factors that determine the answer are: What does the free variable `y` in `f` refer to? What does `x` refer to in the function call `f x`?

The answer to the first question is determined by deep vs shallow binding. Using deep binding, we look for the closest binding for `y` from the function call `g f` because this is where `f` gets passed as a function argument for the first time. Using either lexical or dynamic scoping, the closest `y` to this code position is the value `y = 20` defined at the top-level scope. Using shallow binding, we look for the closest binding for `y` from the place where we actually invoke `f`, that is, from the expression `f x` inside `g`. Since `g` locally sets `y = 5`, we have `y = 5` using shallow binding, no matter whether lexical or dynamic scoping is used.

The answer to the second question is determined by lexical vs dynamic scoping. If we use lexical scoping, then `x` in the function call `f x` refers to the value `x = 10` defined at the top-level scope because `g` has no locally defined value `x` and the global scope is the smallest enclosing scope of `g`. Using dynamic scoping, we search the call stack for the most recent definition of a variable `x`. Since `h` calls `g` and introduces the variable `x = 5`, the `x` in the function call `f x` from `g` thus refers to the value `x = 5`. This gives the following table of results using the different combinations of lexical/dynamic scoping and deep/shallow binding:

		Deep binding $y = 20$	**Shallow binding** $y = 5$
Lexical scoping	$x = 10$	30	15
Dynamic scoping	$x = 5$	25	10

Question 2

Let the nesting depth of a procedure or function be defined recursively as follows: The top-level scope has nesting depth 0. If a procedure or function definition is nested immediately inside a depth-$(d-1)$ scope, its nesting depth is d. We call this procedure or function a child of the immediately enclosing depth-$(d-1)$ scope.

Such a procedure P can only call its immediate children (at depth $d+1$) or any immediate child of any of its ancestor scopes. Now assume P calls a procedure or function P' at depth $d' \leq d+1$. We observe that the static chains of the invocations of P and P' agree in the first $d'-1$ stack frames. Moreover, the compiler can analyze the nesting of procedure and function definitions to determine the nesting depth of each procedure or function, that is, the values of d and d' are known at compile time.

Thus, to implement the invocation of P' from P, the compiler generates code that follows $d-d'+1$ pointers in the static chain of P's invocation. This results in the address of the $(d'-1)$st frame in P's

static chain. It then sets up the static chain pointer in the stack frame of P' to point to this $(d'-1)$st stack frame in P's static chain.