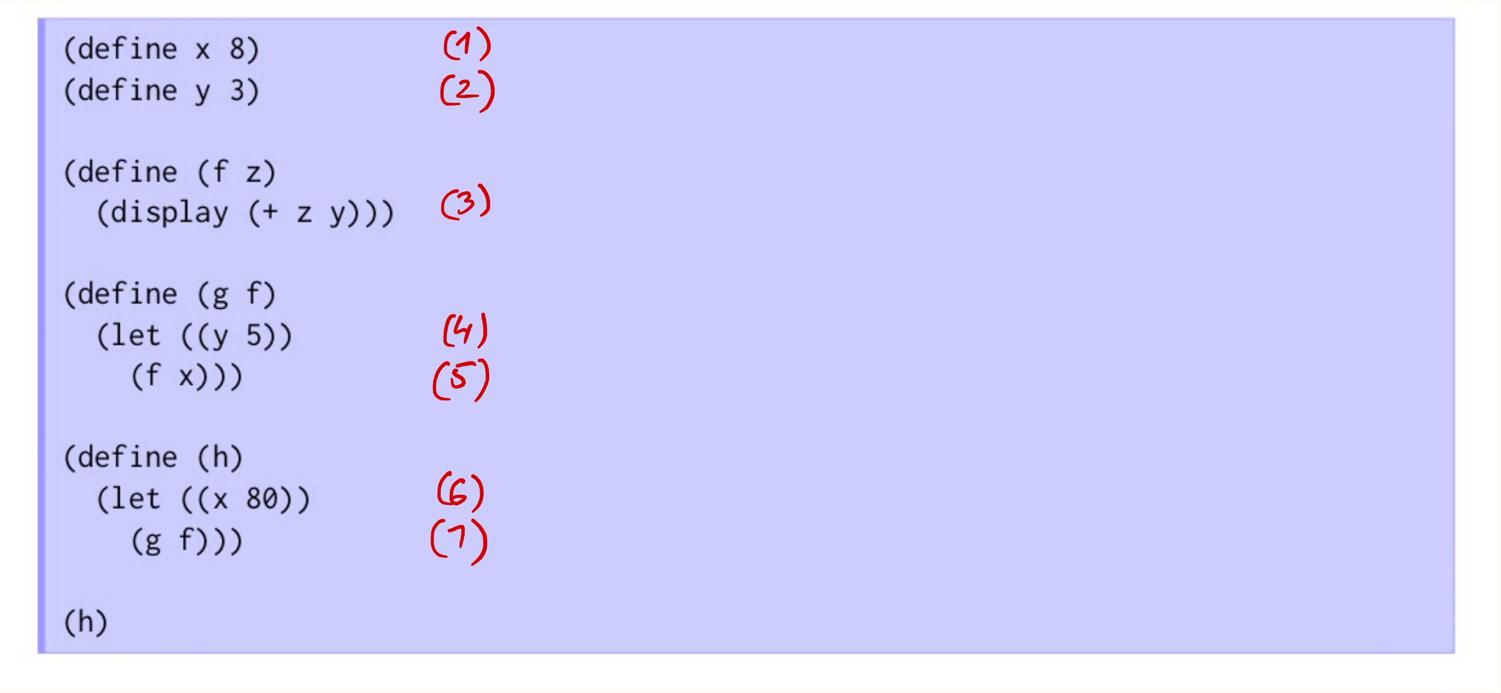
# Sample Solution - Assignment 6
# CSCI 3136

**Question 1** To explain the answer more easily, here is a copy of the code with some lines numbered.



```
(define x 8)                (1)
(define y 3)                (2)

(define (f z)
  (display (+ z y)))        (3)

(define (g f)
  (let ((y 5))              (4)
    (f x)))                 (5)

(define (h)
  (let ((x 80))             (6)
    (g f)))                 (7)

(h)
```

(a) Static binding. In this case, y is a free variable in line 3. The smallest enclosing lexical scope with a variable y is the top level (line 2). Thus, y is 3 in line 3. z is whatever argument is passed to f in line 5. Since x is a free variable in g, we once again look for the smallest enclosing scope that defines x. This is once again the top level (line 1). Thus, x is 8 in line 5, that is, z is 8 in line 3 and line 3 prints 8+3 = 11.

(b) Dynamic and deep binding. The first time $f$ is passed as a parameter to a function is in line 7. At this point, the dynamically most recent binding of $y$ (the only free variable in $f$) is the top-level definition in line 2. Thus, $y$ is bound to 3 in $f$. The function call $(f\ x)$ in line 5 uses the most recent binding for $x$ during the program execution. This is the local definition in line 6. Thus, $f$'s parameter $z$ is bound to 80 and line 3 prints $80+3=83$.

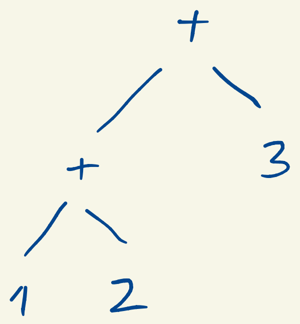
(c) Dynamic and shallow binding. $f$ is called in line 5. The most recent binding for $y$ is in line 4. Thus, $f$'s free variable $y$ is bound to 5. The most recent binding for $x$ in line 5 is once again the binding in line 6, so $f$'s parameter $z$ is 80 again. Line 3 prints $80+5=85$.

## Question 2

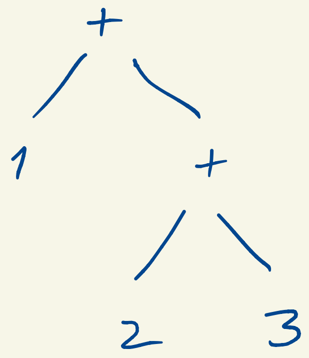
Consider some function $f$ calling another function $g$. Since we use static scoping, $g$ must be defined in $f$ or in some enclosing scope. Let us call $f$'s body scope 0, the immediately enclosing scope scope 1, and so on. $k$ hops along $f$'s static chain get us to the stack frame corresponding to scope $k$. If $g$ is defined in the $k$th scope, then this frame is the target of $g$'s static link. Thus, in the same way that the compiler counts enclosing scopes to determine the number of hops along $f$'s static chain necessary to reach a variable $x$ accessed from $f$, it can also count enclosing scopes until finding g and the number of these scopes is the number of hops along f's static chain that get us to the target frame of g's static link.

## Question 3

There is no contradiction. Associativity determines the shape of the expression tree: 1+2+3 has the expression tree



not



Analogously to the existence of many derivations corresponding to a parse tree, the nodes of an expression tree can be evaluated in any order as long as ancestors are evaluated after descendants.