CSCI 3155: Lab 1 (due January 28)

Ian Smith & Noah Leuthaeuser

1. For each the following uses of names, give the line where that name is bound. Briefly explain your reasoning (in no more than 1 to 2 sentences).

   (a) *Line 1:*
   pi : Double is bound
   Line 2:
   r : Double is bound
   circumference : Double =>Double is bound
   Line 3:
   pi: Double is bound
   Line 6:
   r: Double is bound
   area : Double =>Double is bound

   The use of **pi** on line 4 is bound at line 3 because this is the **pi** inside the scope that contains line 4. The scope is marked by the block with the open curly bracket at the end of line 2 and the close bracket on line 5. The **pi** definition inside this block "shadows" the **pi** outside of this block
   The use of **pi** on line 7 is bound at line 1 because it is outside of the inner block that contains the (shadow) **pi** bound at line 3. No new block is created for area so it referes to original binding.

   (b) *Line 1:*
   x : Int is bound
   Line 2:
   x : Int is bound
   f : Int =>Int is bound
   Line 5:
   x : Int is bound
   Line 6:
   y: Int is bound
   Line 8:
   x : Int is bound
   Line 13:
   y : Int is bound

   The use of **x** at line 3 is bound at the function definition on line 2. Function parameters are bound to the value passed in.
   The use of **x** at line 6 is bound to the **x** on line 5. The name is rebound at the case statement and the **x** on line 6 is in the scope of that case statement.

The use of **x** at line 10 is bound at line 5 because it is outside the block on line 7 to 10 but it is still inside the case statement block.
The use of **x** at line 13 is bound at line 1 (the original global binding of **x**). The function block is closed by line 13, so **x** at line 13 is in global scope.

2. Is the body of *g* well-typed? If so, give the return type of *g* and explain how you determined this type.

   Function *g* is well-typed and it has return type **((Int,Int), Int)**. This type was determined by looking at both return statements (**(b,1)** and **(b, a+2)**) and working backwards. All returns will return this type and we can work backwards to see that the tuples returned are actually this type. In line 2, a pattern-matching binds **a** to integer 1 and binds **b** to a tuple (**x**:Int, 3), where 3 is also an Int. So on line 2, types of **a** and **b** are **a : Int** and **b: (Int, Int)**. Then in either evaluation of the if, the first element in the tuple returned is **b** with type (Int, Int) and the second element is either the Int 1 or the Int from evaluation of **a** + 2.

   (**b**,1) : ((Int,Int), Int) because
   - **b** : (Int, Int) because
     - (**x**,3): (Int, Int) because
       - **x** : Int
       - **3** : Int
   - 1 : Int

   (**b**, **a** + 2) : ((Int,Int),Int) because
   - **b** : (Int,Int) because
     - (**x**,3): (Int, Int) because
       - **x** : Int
       - **3** : Int
   - (**a** + 2) : Int because
     - **a** : Int
     - 1 : Int