COMP 212 : Functional Programming, Fall 2024

Homework 02

Name: ____________________

Wes Email: ____________________

Question	Points	Score
1	20	
2	15	
Total:	35	

If possible, please type/write your answers on this sheet and upload a copy of the PDF to your google drive handin folder. Otherwise, please write the answers in some sort of word processor and upload a PDF. Please name the file `hw02-written.pdf`.

1. **Basics**

   The type `real` represents numbers with decimal points/fractions using a floating point representation. The built-in function `real` : `int -> real` returns the `real` value corresponding to a given `int` input; for example, `real 4` evaluates to `4.0`. Conversely, the built-in function `trunc` : `real -> int` returns the integral part (intuitively, the digits before the decimal point) of its input; for example, `trunc 2.7` evaluates to `2`. Feel free to try these functions out in `smlnj`.

   Once you understand these functions, you should solve the questions in this section in your head, *without* first trying them out in `smlnj`. The type of mental reasoning involved in answering these questions should become second nature.

(3) (a) Consider the following code fragment:

```
fun square (x : real) : real = x * x
fun square (x : int) : int = x * x
val z : real = square 7.0
```

Does this typecheck? Briefly explain why or why not.

Solution:

(b) In class, we went over SML's syntax for let-bindings. It is possible to write `val` declarations in the middle of other expressions with the syntax `let ... in ... end`. See the end of the Lecture 3 notes for more details. Consider the following code fragment (the line-numbers are for reference, not part of the code itself):

```
 (1)  val x : int = 12
 (2)
 (3)  fun assemble (x : int, y : real) : int =
 (4)    let val q : real = let val x : int = 3
 (5)                           val p : real = 5.2 * (real x)
 (6)                           val y : real = p * y
 (7)                           val x : int = 123
 (8)                       in p + y
 (9)                       end
(10)    in
(11)      x + (trunc q)
(12)    end
(13)
(14)  val z = assemble (x, 2.0)
```

(2) i. What gets substituted for the variable `x` in line (5)? Briefly explain why.

Solution:

(2) ii. What gets substituted for the variable `p` in line (8)? Briefly explain why.

Solution:

(2) iii. What gets substituted for the variable `x` in line (11)? Briefly explain why.

Solution:

(2) iv. What value does the expression `assemble (x, 2.0)` evaluate to in line (14)?

Solution:

(6) (c) Consider the following code fragment:

```
fun square (x : int) : int = x * x
val z : int =
  let
    val x : real = real (square 6)
  in
    3 + (trunc x)
  end
```

Provide a step-by-step sequential evaluation trace of the right-hand-side of the declaration of `z` (that is, `let val x : real = real (square 6) in 3 + (trunc x) end`). You may assume that, for values `i : int`, the expression `real i` evaluates in one step to the corresponding `real` value, and similarly for `trunc r` given a value `r : real`.

Solution:

(3) (d) Recall the `fact` function from Lecture 3. Define

```
fun f (x : int) : int = f(x) + f(x)
```

Are the following two expressions behaviorally equivalent?[1] Explain why or why not. (Recall that ~ 1 is the SML notation for negative 1.)

$$\texttt{fact (}\sim\texttt{ 1)} \overset{?}{\cong} \texttt{f 10}$$

[1]In Tuesday's class, we wrote $e_1 = e_2$ for behavioral equivalence. Since there are multiple possible notions of equivalence of programs (e.g. "has the same behavior" or "has the same behavior and the same running time"), we'll sometimes use $\cong$ to emphasize that we're refering specifically to behavioral equivalence.

Solution:

2. **Induction**

(15) (a) The sum of the natural numbers from 0 to n can be calculated quickly like this

$$0 + 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

We will prove the `summ` function that you implemented in lab correct by relating it to this formula.

```
fun summ (n : int) : int =
    case n of
      0 => 0
    | _ => n + (summ (n - 1))
```

Theorem 1. *For all natural numbers* `n`*,* `summ n` $\cong$ `(n*(n+1)) div 2`*.*

The proof is by induction on the natural number n. Your equality reasoning should include each individual step of evaluation necessary to prove the equivalence. In the inductive case, you will need to do some algebraic manipulation; you may assume basic properties of arithmetic (associativity, distributivity of $*$ over $+$, commutativity, etc.).

Solution: The proof is by induction on `n`.

Case for 0

To show:

Proof:

Case for $1 + k$

Inductive hypothesis:

To show:

Proof: