

HW 7

50 points

You can work on your own or in a team of two

Posted Tuesday, December 2

Due Thursday, December 11 at midnight

Submit your solutions in file HW7Solutions.pdf. Size limit in Submittly is set to 2.5MB.

Problem 1 (5pts). Consider the *twice* combinator:

$$twice = \lambda f. \lambda x. f(f\ x)$$

Reduce expression *twice twice f x* into *normal form* using *normal order reduction*. For full credit, show each step on a separate line.

Problem 2 (8pts). Now consider the Haskell implementation of *twice*:

```
twice f x = f (f x)
```

- (a) What is the type of `twice`?
- (b) What is the type of expression `twice twice`?
- (c) If the type of `fun` is `Int->Int`, what is the type of expression `twice twice fun`?
- (d) If the type of `fun` is `Int->Int` and expression `twice twice fun v` is well-typed, what is the type of `twice twice fun v`?

Note: You do not need to justify your answer, just write the corresponding type expression.

Problem 3 (3pts). Fill in the blanks in the Haskell quicksort with list comprehensions:

```
quicksort [] = []
quicksort (x:xs) = quicksort ... ++ [x] ++ quicksort ...
```

Problem 4 (12pts). Write `merge`, the function that takes two sorted lists of numbers and produces a single sorted list in

- (a) Prolog
- (b) Scheme
- (c) Haskell

Problem 5 (5pts). In the following code, which of the variables will a compiler consider to have compatible types under structural equivalence? Under strict name equivalence? Under loose name equivalence?

```
type A = array [1..10] of integer
type B = A
a : A
b : A
c : B
d : array [1..10] of integer
```

Problem 6 (10pts). Show the type trees for the following C declarations:

```
double *a[n];
double (*a)[n];
double (*a[n])();
double (*a()) [n];
double (*a(int, double(*) (double, double[])))(double);
```

Problem 7 (7pts). Consider the Pascal-like code for function `compute`. Assume that the programming language allows a mixture of parameter passing mechanisms as shown in the definition.

```
double compute(first : integer /*by value*/, last : integer /*by value*/,
  incr : integer /*by value*/, i : integer /*by name*/, term : double /*by name*/)

  result : double := 0.0
  i := first
  while i <= last do
    result := result + term
    i := i + incr
  endwhile
  return result
```

- (2pts) What is returned by call `compute(1, 10, 1, i, A[i])`?
- (2pts) What is returned by call `compute(1, 5, 2, j, 1/A[j])`?
- (3pts) `compute` is a classic example of *Jensen's device*, a technique that exploits call by name and side effects. Write `max`, which uses Jensen's device to compute the maximum value in a set of values based off of an array `A`.