

1. [15] The two parts of this question can be answered independently.
- a. Define `insert :: Int -> [Int] -> [Int]` recursively so that if `bs` is a sorted list (i.e., one whose elements are in nondescending order), `insert x bs` is a sorted list containing all the elements of `bs` and also `x`.

For example,

```
insert 3 [1,4,9]  => [1,3,4,9]
insert 3 [3,3]   => [3,3,3]
```

A solution:

```
> insert :: Int -> [Int] -> [Int]
> insert x []      = [x]
> insert x (y:ys)
>   | x<=y        = x : y :ys
>   | otherwise   = y : insert x ys
```

- b. Define a list-sorting function `insertSort :: [Int] -> [Int]` using `insert` and `foldr`. For example,

```
insertSort [3,2,4,5,2]  => [2,2,3,4,5]
```

A solution:

```
> insertSort = foldr insert []
```

2. [15] Define a Haskell IO action `cal` which reads, on successive lines, decimal numerals of the form `(+|-) Digit {Digit}`, until it reads a line beginning with "=", when it outputs the sum of the preceding numerals and quits. A line that does not conform to the prescribed grammar should be noted with an error message, and should not be included in the sum.

A sample session:

```
CS345MT2> cal
+99
-33
12
illegal operator
-6
=
60
```

A solution:

```
> cal :: IO ()
> cal = do sum <- readAdd 0
>         putStr (show sum)
>       where
> readAdd :: Int -> IO Int
> readAdd m = do (op:num) <- getLine
>                 if op == '=' then return m
>                 else if not (null num) && and (map isDigit num)
>                     then do let n = read num
>                               case op of
>                                 '+' -> readAdd (m+n)
>                                 '-' -> readAdd (m-n)
>                                 _   -> do putStr "illegal operator"
>                                         readAdd m
>                 else do putStr "not a number"
>                         readAdd m
```

3. [15] Define a Haskell parser `p11` for lists of lowercase letters enclosed in braces and separated by commas. The parser should deliver a string consisting of the letters parsed.

For example.

input	delivered
{a,z,w}	"azw"
{}	""
{r}	"r"

A solution:

```
> p11 :: Parser String
> p11 = do char '{'
>         letters <- pLets
>         char '}'
>         return letters
>   where
>     pLets :: Parser String
>     pLets = do lettr <- lower
>                lettrs <- many (do char ','; lower)
>                return (lettr:lettrs)
>
>         +++
>         return []
```

4. [15] Prove that for any finite list `xs`,

$$\text{reverse } (xs ++ ys) = \text{reverse } ys ++ \text{reverse } xs.$$

The definitions of `reverse` and `(++)` (with line numbers for reference in your proof):

$$\begin{aligned} \text{reverse } [] &= [] && (1) \\ \text{reverse } (c:cs) &= \text{reverse } cs ++ [c] && (2) \end{aligned}$$

$$[] ++ bs = bs \quad (3)$$

$$(a:as) ++ bs = a : (as ++ bs) \quad (4)$$

You may use any theorems proved in class or in published homework solutions.

The proof, by induction over `xs`:

**Case** `[]`:

$$\begin{aligned} &\text{reverse } ([] ++ ys) = \text{reverse } ys ++ \text{reverse } [] \\ &= \{ 3, 1 \} \\ &\text{reverse } ys = \text{reverse } ys ++ [] \\ &= \{ \text{Homework 5, Exercise 6: } xs ++ [] = xs \} \\ &\text{reverse } ys = \text{reverse } ys \\ &= \{ \text{identity} \} \\ &\text{true} \end{aligned}$$

**Case** `(x:xs)`

$$\begin{aligned} &\text{reverse } ((x:xs) ++ ys) \\ &= \{ 4 \} \\ &\text{reverse } (x : (xs ++ ys)) \\ &= \{ 2 \} \\ &\text{reverse } (xs ++ ys) ++ [x] \\ &= \{ \text{induction hypothesis} \} \\ &(\text{reverse } ys ++ \text{reverse } xs) ++ [x] \\ &= \{ \text{associativity of } (++) \text{ (Lecture slide 127)} \} \\ &\text{reverse } ys ++ (\text{reverse } xs ++ [x]) \\ &= \{ 2 \} \\ &\text{reverse } ys ++ \text{reverse } (x:xs) \quad \blacksquare \end{aligned}$$

5. [15] Find the weakest precondition such that execution of

**if**  $x > 3$  **□**  $a, b := b+2, c-3$  **fi**

establishes  $a \leq b$ .

$$\begin{aligned} & \text{wp "if } x > 3 \text{ □ } a, b := b+2, c-3 \text{ fi" } a \leq b \\ &= x > 3 \text{ □ wp " } a, b := b+2, c-3 \text{ " } a \leq b \\ &= x > 3 \text{ □ } (a \text{ □ } b)_{b+2, c-3}^{a, b} \\ &= x > 3 \text{ □ } b+2 \leq c-3 \\ &= x > 3 \text{ □ } b \leq c-5 \end{aligned}$$

6. [15] Translate the following fragment of GCN code into C, and the C fragment into GCN:

**a.**  $a, b := a+b, a+2$

$t1 = a+b; t2 = a+2; a = t1; b = t2;$

**b.** `for( i = 0; i < k; i++ )`

```
{
    A[i] = i+j;
    if( i == j ) break;
    A[j] = i;
}
```

$i := 0; \text{ break} := \text{false};$

**do**  $i < k$  **□**  $\text{break}$  **□**  $A[i] := i+j;$

**if**  $i = j$  **□**  $\text{break} := \text{true}$

**□**  $i \neq j$  **□**  $A[j] := i; i := i+1$

**fi**

**od**