

1. [10] The Prelude function `elem` is defined so that `elem n ns` is `True` if and only if the value of `n` occurs in list `ns`. Use `foldr` to define `elem` nonrecursively for Integer lists.

```
> elem :: Integer -> [Integer] -> Bool
> elem a = foldr (\x b -> x==a || b) False
```

2. [10] Define `getParagraph` as an IO action which gets successive lines from the keyboard, stopping when it gets an empty line and delivering the concatenation of the preceding lines. The concatenated lines should be separated by spaces, but the last line should not be followed by a space.

```
> getParagraph :: IO String
> getParagraph = do s <- getLine
>                 if null s then return []
>                 else do ss <- getParagraph
>                         if null ss then return s
>                         else return (s ++ (' ' : ss))
```

3. [15] Two related questions on parsing.

- a. Define a function `pString` which, applied to a string, returns a parser that recognizes that string. The parser should deliver the string it recognizes. Make good use of one or more of the parser combinators introduced in class.

```
> pString :: String -> Parser String
> pString [] = return []
> pString (c:cs) = do t <- char c; ts <- pString cs; return (t:ts)
```

- b. Assuming that

```
expr :: Parser Expr    is a parser that recognizes expressions
stmL :: Parser StmL    is a parser that recognizes statement lists
```

define a parser `m2IF` which recognizes Modula-2's IF-THEN-[ELSE]-END construct (your parser doesn't need to handle ELSEIF clauses). The type of the parser's recognized item should be `(Expr, [StmL])`.

```
> m2IF :: Parser (Exp, [StmL])
> m2IF = do pString "IF"
>          e <- expr
>          pString "THEN"
>          s0 <- stmL
>          (do pString "ELSE"
>             s1 <- stmL
>             pString "END"
>             return (e, [s0, s1])
>          +++
>          do pString "END"
>             return (e, [s0]))
```

4. [15] Prove that

$$\text{map } f (\text{map } g \text{ } xs) = \text{map } (f.g) \text{ } xs$$

given the definitions

$$\text{map } h \text{ } [] = [] \quad (1)$$

$$\text{map } h (x:xs) = h \text{ } x : \text{map } h \text{ } xs \quad (2)$$

$$(f1.f2) \text{ } n = f1(f2 \text{ } n) \quad (3)$$

Proof:

Case $[]$:

$$\begin{aligned} & \text{map } f (\text{map } g \text{ } []) = \text{map } (f.g) \text{ } [] \\ &= \{ (1) \} \\ & \text{map } f \text{ } [] = [] \\ &= \{ (1) \} \\ & [] = [] \\ &= \{ \text{identity of } (=) \} \\ & \text{true} \end{aligned}$$

Case $(a:as)$

$$\begin{aligned} & \text{map } f (\text{map } g (a:as)) \\ &= \{ (2) \} \\ & \text{map } f (g \text{ } a : \text{map } g \text{ } as) \\ &= \{ (2) \} \\ & f(g \text{ } a) : \text{map } f (\text{map } g \text{ } as) \\ &= \{ (3) \} \\ & (f.g) \text{ } a : \text{map } f (\text{map } g \text{ } as) \\ &= \{ \text{induction hypothesis} \} \\ & (f.g) \text{ } a : \text{map } (f.g) \text{ } as \\ &= \{ (2) \} \\ & \text{map } (f.g) (a:as) \quad \blacksquare \end{aligned}$$

5. [10] Use the method of weakest preconditions to determine the truth or falsehood of

$$\{x < y - 10\} \ x, y := y - 1, x + 5 \ \{y < x + 2\}.$$

$$\begin{aligned} & \text{wp}("x, y := y - 1, x + 5", y < x + 2) \\ &= \{ \text{def. ':='} \} \\ & (y < x + 2)_{y-1, x+5}^{x, y} \\ &= \{ \text{substitution} \} \\ & x + 5 < y - 1 + 2 \\ &= \{ \text{simplification} \} \\ & x < y - 4 \\ & \square \{ \text{predicate calculus} \} \\ & x < y - 10 \end{aligned}$$

So the proposition is true. \blacksquare

6. [15] Translate each of the following into Guarded-Command notation:

a. Pascal:

```

case y of
  0: x := x+a;
  1: x := x+b;
  2: x := x+c
end

```

```

if y=0  $\square$  x := x+a
   $\square$  y=1  $\square$  x := x+b
   $\square$  y=2  $\square$  x := x+c
fi

```

b. C/C++:

```

switch( y )
{
  case 0: x = x+a;
  case 1: x = x+b;
  case 2: x = x+c;
}

```

```

if y=0  $\square$  x := x+a+b+c
   $\square$  y=1  $\square$  x := x+b+c
   $\square$  y=2  $\square$  x := x+c
   $\square$  y $\neq$ 0 $\square$ y $\neq$ 1 $\square$ y $\neq$ 2  $\square$  skip
fi

```

c. Modula-2:

```

FOR k := a TO b BY d DO
  A[k] := B[k] + C[k]
END

```

```

k := a;
do (d>0  $\square$  k $\leq$ b) (d<0  $\square$  k $\geq$ b)  $\square$ 
  A[k] := B[k] + C[k]; k := k+d
od

```

7. [10] Determine the type of

```

f x y z
  | null y = [x = "123"]
  | otherwise = z (head y)

```

Solution:

```

x :: a
y :: b
z :: c
f :: a -> b -> c -> d

```

```

y :: [e]          null y
x :: String      x = "123"
z :: e -> d      f x y z = z (head y), head :: [e] -> e
d = [Bool]      [x = "123"], (==) :: g -> g -> Bool
b = [e]
c = e -> d
a = String

```

```

f :: String -> [e] -> (e -> [Bool]) -> [Bool]

```