```
1  -- Aufgabe 1
2  data Season = Spring | Summer | Automn | Winter
3   deriving (Eq, Ord, Enum, Show)
4
5  data Month = January | February | March | April | May |
       June | July | August | September | October | November |
       December
6   deriving (Eq, Ord, Enum, Show)
7
8  toSeason :: Month -> Season
9  toSeason January = Winter
10 toSeason February = Winter
11 toSeason March = Spring
12 toSeason April = Spring
13 toSeason May = Spring
14 toSeason June = Summer
15 toSeason July = Summer
16 toSeason August = Summer
17 toSeason September = Automn
18 toSeason October = Automn
19 toSeason November = Automn
20 toSeason December = Winter
21
22 -- Aufgabe 2
23 data Shape = Circle Float | Rectangle Float Float
24 -- a)
25 lengthOfEdge :: Shape -> Float
26 lengthOfEdge (Circle r) = 2*pi*r
27 lengthOfEdge (Rectangle a b) = 2*(a+b)
28
29 -- b)
30 instance Show Shape where
31   show (Rectangle a b) = take (round a) (repeat ('*')) ++ "
       \n" ++ concat (take ((round b)-2) hilf) ++ take (round
        a) (repeat ('*'))
32     where hilf = repeat ("*" ++ take ((round a)-2) (repeat
         (' ')) ++ "*\n")
33 --  show (Circle r) =
34
35 ausgabe a b = putStr (show (Rectangle a b))
```

```
36
37 -- Aufgabe 3
38 -- a)
39 data EAA = Atom Float | Add EAA EAA | Sub EAA EAA | Mult
       EAA EAA | Div EAA EAA | Min EAA
40
41 a,b :: EAA -- Beispielausdrücke
42 a = Sub (Atom 3) (Add (Atom 4) (Atom 9)) -- a = 3-(4+9)
43 b = Sub (Add (Atom 4) (Atom 9)) (Atom 3) -- b = (4+9)-3
44
45 value :: EAA -> Float
46 value (Atom z) = z
47 value (Add a1 a2) = value a1 + value a2
48 value (Sub a1 a2) = value a1 - value a2
49 value (Mult a1 a2) = value a1 * value a2
50 value (Div a1 a2) = value a1 / value a2
51 value (Min a) = - (value a)
52
53 instance Show EAA where
54     show (Atom z) = show z
55     show (Add a1 a2) = "(␣" ++ show a1 ++ "␣+␣" ++ show a2
           ++ "␣)"
56     show (Sub a1 a2) = "(␣" ++ show a1 ++ "␣-␣" ++ show a2
           ++ "␣)"
57     show (Mult a1 a2) = "(" ++ show a1 ++ "␣*␣" ++ show a2
           ++ "␣)"
58     show (Div a1 a2) = "(" ++ show a1 ++ "␣/␣" ++ show a2 ++
           "␣)"
59     show (Min a) = "(␣-␣" ++ show a ++ "␣)"
60
61 -- Stapelmaschine zur Auswertung von EAA's
62
63 data Com = PUSH Float | ADD | SUB | MULT | DIV | MIN --
       Befehlssatz der Stapelmaschine
64     deriving Show
65
66 type Prog = [Com]  -- Programme der Stapelmaschine
67
68 makeProg :: EAA -> Prog -- Übersetzer
69 makeProg(Atom z) = [PUSH z]
```

```
70 makeProg (Add a1 a2) = (makeProg a1) ++ (makeProg a2) ++ [
   ADD]
71 makeProg (Sub a1 a2) = (makeProg a1) ++ (makeProg a2) ++ [
   SUB]
72 makeProg (Mult a1 a2) = (makeProg a1) ++ (makeProg a2) ++ [
   MULT]
73 makeProg (Div a1 a2) = (makeProg a1) ++ (makeProg a2) ++ [
   DIV]
74 makeProg (Min a) = (makeProg a) ++ [MIN]
75
76 type Stack = [Float] -- Datenspeicher der Stapelmaschine
77
78 data StackMachine = State Prog Stack -- die Stapelmaschine
   hat ein Programm- und ein Datenspeicher
79    deriving Show
80
81 step :: StackMachine -> StackMachine --
   Zustandsüberführungsfunktion
82 step (State (c:cs) s) = State cs (apply c s)
83
84 apply :: Com -> Stack -> Stack -- Befehlsausführung
85 apply (PUSH z) s = z:s
86 apply ADD (z1:z2:s) = (z2+z1):s
87 apply SUB (z1:z2:s) = (z2-z1):s
88 apply MULT (z1:z2:s) = (z2*z1):s
89 apply DIV (z1:z2:s) = (z2/z1):s
90 apply MIN (z:s) = (-z):s
91
92 load :: Prog -> StackMachine -- Laden eines Programms
93 load p = State p []
94
95 run :: StackMachine -> StackMachine -- Ausführung eines
   Programms
96 run (State [] s) = State [] s
97 run sm = run (step sm)
98
99 dump :: StackMachine -> IO() -- Ausdrucken eines
   Programmlaufs
100 dump (State [] s) = putStr (show (State [] s))
101 dump sm = do putStr (show sm)
```

```
102             putStr "\n"
103             dump (step sm)
104
105 {- Der Wert eines Ausdrucks a kann berechnet werden, indem
    er in
106    ein Programm der Stapelmaschine übersetzt wird, das
       geladen und
107    auf der Stapelmaschine ausgeführt wird.
108    Anschließend liegt der Wert auf dem Keller.
109 -}
110
111 eval :: EAA -> Float
112 eval = readTop . run . load . makeProg
113
114 readTop :: StackMachine -> Float
115 readTop (State p (z:s)) = z
```