```
--Übungsblatt05 ; Thomas Alessandro Buse ; 192959 ; Gruppe: 17
--                                Paul Rüssmann ; 196683
import Expr

--Aufgabe 6.1 a

a :: Exp String
a = Sum [2:*(Var"x":^3),5:*(Var"y"),Con 2]

b :: Exp String
b = (Var"z":^2)

--Aufgabe 6.1 b

--Aufgabe 6.2
type BStore x = x -> Bool
bexp2store :: BExp x -> Store x -> BStore x -> Bool

data BExp x = True_ | False_ | BVar x | Or [BExp x] |
                                And [BExp x] | Not (BExp x) | Exp x := Exp x |
                                Exp x :<= Exp x

bexp2store True_ _ _                    = True
bexp2store False_ _ _                   = False
bexp2store (BVar x) _ bst               = bst x
bexp2store Or [BExp x] st bst           = Or $ [bexp2store BExp x st bst]
bexp2store And [BExp x] st bst          = And $ [bexp2store BExp x st bst]
bexp2store Not (Bexp x) st bst          = if bexp2store BExp x st bst == True then False else True
bexp2store (Exp x := Exp x) st bst      = bst $ (exp2store x $ st == exp2store x $ st)
bexp2store (Exp x :<= Exp x) st bst = bst $ (exp2store x $ st <= exp2store x $ st)
```

```
--Aufgabe 6.3


data Colist a = Colist { split :: Maybe (a, Colist a)}

data Stream a = (:<) { hd :: a, tl :: Stream a}


class A a where

        drop' :: Int -> a -> a


instance A [a] where

        drop' 0 s = s

        drop' n (a:s) | n > 0 = drop' (n-1) s

        drop' _ [] = []



instance A (Colist a) where

        drop' 0 z = z

        drop' _ (Colist(Nothing)) = (Colist(Nothing))

        drop' t (Colist(Just(x,y))) = drop' (t-1) y



instance A (Stream a) where

        drop' 0 z = z

        drop' z (a Main.:<s) = drop' (z-1) s


--Aufgabe 6.4

data Bintree a = Empty | Fork a (Bintree a) (Bintree a) deriving Show

data Edge = Links | Rechts deriving Show

type Node = [Edge]


btreeExample = Fork 5 (Fork 3 (Fork 1 Empty (Fork 2 Empty Empty)) (Fork 4 Empty Empty))(Fork 8
(Fork 6 Empty (Fork 7 Empty Empty)) (Fork 10 (Fork 9 Empty Empty)Empty))

nodeExmaple = [Links]
```

```
--Aufgabe 6.4 a

value :: Node -> Bintree a -> Maybe a
value _ Empty = Nothing
value [] (Fork a l r) = Just a
value n (Fork a l r) = if show(head n) ==show(Links) then value (drop 1 n)l else value (drop 1 n) r

--Aufgabe 6.4 b

search :: Eq a => a -> Bintree a -> Maybe Node
search v Empty = Nothing
search v (Fork a l r) = if v == a then Just [] else help (search v l) (search v r) where
        help :: Maybe Node -> Maybe Node -> Maybe Node
        help Nothing Nothing = Nothing
        help (Just xs) _ = Just (Links:xs)
        help  _ (Just xs) = Just (Rechts:xs)
```