```
{-

Beh 1.

1)

LET True = \ x y . x
LET False = \ x y . y

LET EQU = \ a b . a b (b F T)

EQU F F =
  (\ a b . a b (b F T)) F F =
  (\ b . F b (b F T)) F =
  F F (F F T) =
  (\ x y . y) F (F F T) =
  (\ y . y) (F F T) =
  F F T =
  (\ x y . y) F T =
  (\ y . y) T =
  T

-}

-- 2)

takeN _ [] = []
takeN n (x:xs)
  | n <= 0 = []
  | True   = x : takeN (n-1) xs


-- dropN bylo definováno

dropN = drop

replac p c1 c2 [] = []
replac p c1 c2 (l:ls) = (fsx ++ rx nxx) : replac p c1 c2 ls
  where
      fsx = takeN p l    -- případně (p+1) nebo (p-1), jak to kdo bere
      nxx = dropN p l    -- případně (p+1) nebo (p-1), jak to kdo bere
      rx [] = []
      rx (c:cs) = if c1==c then c2:cs else c:cs


-- 3)

data Expr
  = Val Int
  | Add Expr Expr
  | Mul Expr Expr
  deriving (Show,Eq)

eval (Val v) = v
eval (Add (Val 0) e) = eval e
eval (Add e (Val 0)) = eval e
eval (Add e1 e2) = eval e1 + eval e2
eval (Mul (Val 0) _) = 0
eval (Mul _ (Val 0)) = 0
eval (Mul e1 e2) =
  if ee1==0 then 0 else ee1 * eval e2
  where
    ee1 = eval e1


-- ############################################################################


{-

Beh 2.

1)
```

```
LET True = \ x y . x
LET False = \ x y . y

LET NEQ = \ a b . a (b F T) b

NEQ T T =
  (\ a b . a (b F T) b) T T =
  (\ b . T (b F T) b) T =
  T (T F T) T =
  (\ x y . x) (T F T) T =
  (\ y . (T F T)) T =
  T F T =
  (\ x y . x) F T =
  (\ y . F) T =
  F

-}


-- 2)

splitAT _ [] = ([],[])
splitAT n l@(x:xs) =
  if n<=0 then ([],l)
  else let
    (p,s) = splitAT (n-1) xs
   in (x:p,s)

replac' p s1 s2 [] = []
replac' p s1 s2 (l:ls) =
  if s1==ss1 then (f++s2++ss2) : replac' p s1 s2 ls
  else l : replac' p s1 s2 ls
  where
    (f,s) = splitAT p l
    (ss1,ss2) = splitAT (length s1) s


-- 3)

data BE
  = BTrue
  | BFalse
  | BAnd BE BE
  | BOr BE BE
  deriving (Show,Eq)

eVal BTrue = True
eVal BFalse = False
eVal (BAnd BTrue e) = eVal e
eVal (BAnd e BTrue) = eVal e
eVal (BAnd BFalse _) = False
eVal (BAnd _ BFalse) = False
eVal (BAnd e1 e2) =
  if eVal e1 then eVal e2 else False
eVal (BOr BTrue _) = True
eVal (BOr _ BTrue) = True
eVal (BOr BFalse e) = eVal e
eVal (BOr e BFalse) = eVal e
eVal (BOr e1 e2) =
  if eVal e1 then True else eVal e2


-- ani není nutné tak rozsáhle
{-
eval BTrue = True
eval BFalse = False
eval (BAnd BFalse _) = False
eval (BAnd _ BFalse) = False
eval (BAnd e1 e2) =
  if eval e1 then eval e2 else False
eval (BOr BTrue _) = True
eval (BOr _ BTrue) = True
eval (BOr e1 e2) =
  if eval e1 then True else eval e2
```

```
-}


-- EOF
```