```haskell
--Uebung 7
--Lilli Schuckert und Charlotte Seehagen

import Data.Char

binom1 :: Integer -> Integer -> Integer
binom1 n k = fac (n) `div` (fac (k) * fac(n-k)) --Binomialkoeffizient wird
                                                --berechnet indem man n! durch
                                                --(k!* (n-k)!) rechnet
     where                                      --Hilfsfunktion zur Berechnung
                                                --der Fakultät.
       fac :: Integer -> Integer
       fac n
         | n == 0 = 1                    --Rekursionsanker 0! = 1
         | n > 0 = n * fac(n-1)          --Wenn n>0 ist, dann rechnet man n*n-1

binom2 :: Integer -> Integer -> Integer
binom2 n k
 |k == 0 = 1              --Wenn Rekursionsanker k=0, ist Binomialkoeffizient 1
 |n == k = 1              --Wenn n gleich k, dann ist Binomialkoeffizient 1
 |otherwise = binom2 (n-1) k + binom2 (n-1) (k-1) --Rekursionschritt: Binomial-
                                                  --koeffizient gleich
                                                  --(n-1) über k + (n-1) über
                                                  --(k-1)

binomLaufzeit :: Integer -> Integer -> Integer
binomLaufzeit n k = binomLaufzeit (n-1) k + binomLaufzeit (n-1)(k-1) + 1
--ergibt sich aus der rekursiven Darstellung der Binomialkoeffizienten

simpleShift0 :: Int -> Int -> Int
simpleShift0 l n
  | n == l = 0          --Die Funktion wird 1 Mal geshifted, was bedeutet, dass
                        --wenn die obere Intervallgrenze l und die zu
                        --verschiebende Zahl n gleich sind, dann bildet n auf 0
                        --ab.
  | otherwise = (n + 1) --Ansonsten bildet n auf n+1 ab.

jShift0 :: Int -> Int -> Int -> Int
jShift0 j l n
  | j < 1 || l < 1 || n > l = error "Fehlermeldung"
  --j darf nicht kleiner als 1 sein, da sonst kein Shift stattfindet, l darf
  --nicht kleiner als 1 sein, da die obere Intervallgrenze sonst 0 sein würde
  --und n darf nicht größer als l sein, da die zu verschiebende Zahl sonst
  --außerhalb des Intervalls liegen würde
  | j == 1 && n == l = 0 --1 mal geschifted und n = l, dann bildet n auf 0 ab.
  | j == l = n-1 -- entspricht j der oberen Intervallgrenze, dann bildet n auf
                 -- n-1 ab.
  | (j+n) <= l = (j+n) -- ist j+n kleiner gleich l dann bildet n auf j+n ab.
  | otherwise = jShift0 (j-1) l (simpleShift0 l n) -- Sonst

universalShift :: Int -> Int -> Int -> Int -> Int
universalShift j k l n
  | j < 1 || k < 0 || l <= k || n > l = error "Fehlermeldung"
  -- Fehlermeldung bei n>l, da die zu verschiebende Zahl sonst nicht im
  -- Intervall liegt
  | otherwise = mod(-k+n+j) (l-k+1) + k
  --(-k), damit, dass Intervall bei 0 anfängt. Dann wird die zu verschiebende
  --Zahl n um j Stellen in dem Intervall verschoben, was jetzt bei k=0 anfängt.
  --(l-k+1) ist die neue obere Intervallgrenze. Das gesamte Intervall wird also
  --um k Stellen verschoben. Die Modulo Funktion beschreibt die zyklische
  --Verschiebung von n in dem neuen Intervall [0,(l-k+1)]. Um wieder zum
  --ursprünglichen Intervall [l,k] zurückzukommen und somit die Abbildung der
  --zur verschiebenden Zahl in diesem Intervall darzustellen, muss man die
  --untere Intervallgrenze k wieder hinzufügen.

caesarE :: Char -> Char
```

```haskell
caesar5 :: Char -> Char
caesar5 a
  |ord a >= 65 && ord a <= 90  = chr(universalShift 5 65 90 (ord a))
  -- Großbuchstaben sind im ASCII im Intervall von 65 bis 90 als Dezimalzahl
  -- angegeben. Liegt a in desem Bereich, dann wandelt, die Funktion den
  -- Buchstaben a in eine Zahl um (mit ord a) damit der universalShift
  -- angewendet werden kann, mit j=5. Dann wandelt die Funktion die Zahl wieder
  -- in einen Buchstaben um.
  |ord a >= 97 && ord a <= 122 = chr(universalShift 5 97 122 (ord a))
  -- liegt der Buchstabe im ASCII zwischen 97 und 122, ist es ein Kleinbuchstabe
  --  und die Funktion wandelt wieder erst a in eine Zahl um, um
  -- den Universalshift mit j=5 anwenden zu können. Dann wird die Zahl wieder
  -- in einen Buchstaben umgewandelt mit der vordefinierten Funktion chr.
  |otherwise = error "Fehlermeldung" --wird ein anderes Zeichen eingegeben,
                                      --dass nicht in einem der 2 Zahlenbereiche
                                      --liegt, dann error.

jTwistedCaesar :: Int -> Char -> Char
jTwistedCaesar j a
  |ord a >= 65 && ord a <= 90  = toLower(chr(universalShift j 65 90 (ord a)))
  --gleiches Prinzip, wie bei  caesar5, nur dass um j Stellen geshifted wird
  --und nicht um 5. Der verschobene Großbuchstabe wird dann noch durch die
  --vordefinierte Funktion toLower in einen Kleinbuchstaben umgewandelt.
  |ord a >= 97 && ord a <= 122 = toUpper(chr(universalShift j 97 122 (ord a)))
  --Es wird wieder um j Stellen geshifted und der verschobene Kleinbuchstabe
  --wird durch die vordefinierte Funktion toUpper in einen Großbuchstaben
  --umgewandelt.
  |otherwise = error "Fehlermeldung"

```