Anhang A
Lösungen der Aufgaben

Kapitel 2, Aufgabe 1

```
and, or :: Bool -> Bool -> Bool
and x y = x && y
or  x y = x || y

and, or :: Bool -> Bool -> Bool
and True True = True
and _    _    = False

or True _     = True
or _    True  = True
or _    _     = False
```

Kapitel 3, Aufgabe 2

```
greater :: Int -> Int -> Int
greater a b = if (a>b) then a else b
```

Kapitel 3, Aufgabe 4

```
nand :: Bool -> Bool -> Bool
nand True True = False
nand _    _    = True
```

NAND und NOR stellen jeweils eine vollständige Basis dar. Das bedeutet, dass alle logischen Funktionen durch eine Kombination mit einem der beiden Funktionen darstellbar sind.

Kapitel 4, Aufgabe 1

```
ggT :: Int -> Int -> Int
ggT a b
   | a>b       = ggT (a-b) b
   | a<b       = ggT a (b-a)
   | otherwise = a
```

Kapitel 4, Aufgabe 2

```
isPan :: Int -> Int -> Bool
isPan x y = "123456789" == sort digits
   where digits = (show x) ++ (show y) ++ (show (x*y))

euler32 :: Int
euler32 = sum (nub (pan1 ++ pan2))
   where
      pan1 = [x*y | x<-[1..9]   ,y<-[x..9999],(isPan x y)]
      pan2 = [x*y | x<-[10..99],y<-[x..999]  ,(isPan x y)]
```

Kapitel 4, Aufgabe 3

```
euler4 :: Integer
euler4 = foldr max 0 [x*y | x<-[100..999],
                            y<-[100..999],
                            isPalindrom (x*y)]
   where isPalindrom n = show n == (reverse.show) n
```

Kapitel 5, Aufgabe 1

```
skalarP :: [Int] -> [Int] -> Int
skalarP []     []     = 0
skalarP (x:xs) (y:ys) = x*y + skalarP xs ys
skalarP _      _      = error "Listen nicht gleich lang"
```

Kapitel 5, Aufgabe 2

```
last :: [a] -> a
last xs = xs!!(length xs - 1)
```

Kapitel 5, Aufgabe 3

```
reverse :: [a] -> [a]
reverse []     = []
reverse (x:xs) = reverse xs ++ [x]
```

Kapitel 5, Aufgabe 4

```
euler1 :: Int -> Int
euler1 = sum.teil
  where teil n = [x | x<-[1..n-1], mod x 3==0 || mod x 5==0]
```

Kapitel 5, Aufgabe 5

```
euler2 :: Integer
euler2 = (sum.(filter even).(takeWhile (<4000000))) fibo
   where fibo = 1:2:zipWith (+) fibo (tail fibo)
```

Kapitel 6, Aufgabe 1

```
tausche :: (a->b->c) -> (b->a->c)
tausche f a b = f b a

umdrehen = foldl (tausche (:)) []
summe    = foldl (+) 0
produkt  = foldl (*) 1
```

Kapitel 6, Aufgabe 2

```
machStringGross :: String -> String
machStringGross = map machGross
```

Kapitel 6, Aufgabe 3

```
import Data.List
import Data.Char

primes :: [Int]
primes = 2:3:primes'
  where
    1:p:candidates = [6*k+r | k <- [0..], r <- [1,5]]
    primes'        = p:filter isPrime candidates
    divides n p    = mod n p == 0

isPrime n = all(not .(\p->(mod n p)==0))
            $ takeWhile (\p -> p*p<=n) primes

mirp :: Integer
mirp = [x | x<-[2..], isPrime x, isPrime (toDigit.reverse.show) x]
```

Kapitel 6, Aufgabe 4

```
euler5 :: Integer
euler5 = foldl (lcm) 1 [1..20]
```

Kapitel 6, Aufgabe 5

```
import Char

euler6 = sum (map (digitToInt) (show (2^1000)))
```

Kapitel 7, Aufgabe 3

```
euler = head (pFactors 317584931803 [])
   where
      pFactors 1 xs = xs
      pFactors p xs = pFactors (div p (save p)) xs ++ [save p]
      save x        = head [y | y <- [2..], mod x y == 0]
```

Kapitel 7, Aufgabe 4

```
import List
import Char

zahlen = map digitToInt
            "73167176531330624919225119674426574742355349194934
             96983520312774506326239578318016984801869478851843
             85861560789112949495459501737958331952853208805511
             12540698747158523863050715693290963295227443043557
             66896648950445244523161731856403098711121722383113
             62229893423380308135336276614282806444486645238749
             30358907296290491560440772390713810515859307960866
             70172427121883998797908792274921901699720888093776
             65727333001053367881220235421809751254540594752243
             52584907711670556013604839586446706324415722155397
             53697817977846174064955149290862569321978468622482
             83972241375657056057490261407972968652414535100474
             82166370484403199890008895243450658541227588666881
             16427171479924442928230863465674813919123162824586
             17866458359124566529476545682848912883142607690042
             24219022671055626321111109370544217506941658960408
             07198403850962455444362981230987879927244284909188
             84580156166097919133875499200524063689912560717606
             05886116467109405077541002256983155200055935729725
             71636269561882670428252483600823257530420752963450"

euler = print.maximum.map (product.take 5).tails $ zahlen
```

Kapitel 8, Aufgabe 2

```
import Data.List

primes :: [Int]
primes = 2:3:primes'
```

```
  where
    1:p:candidates = [6*k+r | k <- [0..], r <- [1,5]]
    primes'        = p:filter isPrime candidates
    divides n p    = mod n p == 0

isPrime n = all(not .(\p->(mod n p)==0)) $
            takeWhile (\p -> p*p<=n) primes

primefactors :: Int -> [Int]
primefactors n = primefactors' n n [] primes where
    primefactors' n m l (p:ps)
        | p>n       = l
        | mo == 0   = primefactors' n d (p:l) (p:ps)
        | otherwise = primefactors' n n l ps where
            (d,mo) = divMod m p

numOfFactors n   = product (map ((+1).length)
       (group (primefactors n)))
triangleNumber n = n*(n+1) `div` 2

-- Achtung: Codezeile umgebrochen
euler12 = triangleNumber (head (filter (\n -> 500 <
          (numOfFactors.triangleNumber) n) [1..]))
```

Kapitel 8, Aufgabe 3

```
euler15 = div product [21..40] product [2..20]
```

Kapitel 9, Aufgabe 5

```
fac n = snd (until ((>n) . fst) (\(i,m) -> (i+1, i*m)) (1,1))
p34 = sum liste
    where liste = [x|x<-[2..1000000],
       x==(sum.map (fac.digitToInt).show) x]
```

Kapitel 10, Aufgabe 1

```
buSort :: Ord a => (a -> a -> Bool) -> [a]->[[a]]
buSort f = bSort' [] where
```

```
    bSort' buckets [] = buckets
    bSort' buckets (x:xs) = bSort' (insert x buckets) xs
    insert x [] = [[x]]
    insert x ((e:b):bs)
        | f x e  && f e x= (x:e:b):bs
        | f x e = [x] :(e:b) : bs
        | otherwise = (e:b) : (insert x bs)
```

Kapitel 16, Aufgabe 2

```
type Knoten     = Int
type AdjListe e = DiffArray Knoten [(Knoten,e)]
type Kante e    = ((Knoten,Knoten),e)
newtype ListGraph e = G (AdjListe e)

-- Übung
erzeug :: [Knoten] -> [Kante e] ->  ListGraph e
erzeug [] _ = G (array (0,0) [])
erzeug k l  = G $ accumArray merge [] bounds edges where
    bounds = (lower,upper)
    lower  = minimum k
    upper  = maximum k
    edges  = map (\((a,b),e) -> (a,(b,e))) l
    merge l (n,e)
        | isJust (find (\(m,_) -> n==m) l) = l
        | otherwise                        = (n,e):l

adj :: Knoten -> Knoten -> ListGraph e -> Bool
adj u v (G arr) = elem v $ map fst (arr!u)

adjL :: Knoten -> ListGraph e -> [(Knoten,e)]
adjL u (G arr)  = arr!u

anzKnoten (G arr) = snd.bounds $ arr

istLeer g = 0 == anzKnoten g

knoten :: ListGraph e -> [Knoten]
knoten (G arr) = indices arr

kanten :: ListGraph e -> [Kante e]
kanten g = concatMap (\k -> map (\(n,e) -> ((k,n),e))
           (adjL k g)) (knoten g)

label :: Knoten -> Knoten -> ListGraph e -> Maybe e
label u v (G arr) = fmap snd $ find (\(v',_) -> v==v') (arr!u)

setzLabel :: e -> Knoten -> Knoten -> ListGraph e -> ListGraph e
setzLabel l u v (G arr) = G (arr//[(u,list)]) where
```

```
    list = foldr update [] (arr!u)
    update n@(v',_) r
        | v' == v = (v',l):r
        | otherwise = n:r
```

Kapitel 16, Aufgabe 6

```
import Graph
import Data.List

topSort :: ListGraph e -> [Knoten]
topSort g = reverse $ topSort' g where
    topSort' g
        | istLeer g = []
        | otherwise = let s = senke g in s : topSort'
                              (loeschKnoten s g)

-- Übung
topSort' :: ListGraph e -> [Knoten]
topSort' = reverse.unfoldr f where
    f g
        | istLeer g = Nothing
        | otherwise = let s = senke g in Just
                              (s, loeschKnoten s g)

senke g = case find (\k -> null (adjL k g)) (knoten g) of
    Nothing -> error "keine Senke gefunden"
    Just x  -> x

-- Achtung: Codezeile umgebrochen
loeschKnoten s g = erzeug (filter (/=s) (knoten g))
    [((n,m),e) | ((n,m),e)<-kanten g, n /=s && m /= s]

exampleGraph = erzeug [1..5] kanten
   where kanten =  [((1,2),1), ((1,3),4), ((1,4),4), ((2,5),4),
                    ((2,3),1), ((3,4),1), ((3,5),4), ((4,5),1)]
```

Literaturverzeichnis

1. O'Sullivan B, Goerzen J, Stewart D (2009) Real World Haskell, O'Reilly-Verlag
2. Block M (2009) Java Intensivkurs – In 14 Tagen lernen Projekte erfolgreich zu realisieren, Springer-Verlag, 2. Aufl
3. Neumann A (2009) Programmieren im Kranich-Stil, Bachelorarbeit an der Freien Universität Berlin
4. Block M, Rojas R (2009) Local Contrast Segmentation to Binarize Images, The Third International Conference on Digital Society (ICDS 2009), ISBN:978-1-4244-3550-6, Vol. 1, No. 1, pp. 294–299, Cancun/Mexiko
5. Schöning U (2008) Theoretische Informatik - kurz gefasst, 5. Aufl, Spektrum Akademischer Verlag
6. O'Neill ME (2008) The Genuine Sieve of Eratosthenes, Under consideration for publication in J. Functional Programming, http://www.cs.hmc.edu/~oneill/papers/Sieve-JFP.pdf
7. von Luxburg U (2007) A Tutorial on Spectral Clustering, in Statistics and Computing, Vol. 17, No. 4, pp. 395–416
8. Staab F (2007) Logik und Algebra: Eine praxisbezogene Einführung für Informatiker und Wirtschaftsinformatiker, Oldenbourg-Verlag
9. Dankmeier D (2006) Grundkurs Codierung: Verschlüsselung, Kompression, Fehlerbeseitigung, 3. Aufl, Vieweg-Verlag
10. Lang HW (2006) Algorithmen: in Java, 2. Aufl, Oldenbourg-Verlag
11. Bauer FL (2005) PYTHAGORÄISCHE TRIPEL, Informatik-Spektrum, Vol. 28, No. 5, pp. 417–423, Springer-Verlag
12. Kirmse A (Hrsg) (2004) Spieleprogrammierung Gems 4, Hanser-Verlag
13. Schulz R-H (2003) Codierungstheorie: Eine Einführung, 2. Aufl, Vieweg+Teubner
14. Okasaki C (2003) Fun with binary heap trees, The Fun of Programming, pp. 1–16
15. Claessen K, Hughes J (2003) Specification-based testing with QuickCheck, The Fun of Programming, pp. 17–39
16. Jones SP (Ed) (2002) Haskell 98, Report, http://www.haskell.org/onlinereport/
17. Schöning U (2001) Algorithmik, ISBN-13: 978-3827410924, Spektrum Akademischer Verlag
18. Pope B (2001) A tour of the Haskell Prelude, unpublished, http://www.cs.mu.oz.au/~bjpop/
19. Siefkes D (2000) Historische Paradigmenwechsel in der Informatik, Skript zur gleichnamigen Veranstaltung, TU Berlin, SoSe 2000
20. Hudak P, Peterson J, Fasel J (2000) A gentle introduction to Haskell Version 98, unpublished, http://www.haskell.org/tutorial/
21. Cormen TH, Leiserson CE, Rivest RL (2000) Introduction to Algorithms, MIT-Press
22. Bratko I (2000) Prolog Programming for Artifical Intelligence, 3. Aufl, Addison-Wesley-Verlag
23. Heun V (2000) Grundlegende Algorithmen, Vieweg-Verlag

24. Gibbons J, Jones G (1998) The Under-Appreciated Unfold, Proceedings of the third ACM SIGPLAN international conference on Functional programming, pp. 273–279, United States
25. Knuth DE (1998) The Art of Computer Programming, Vol. 2, ISBN 0-201-89684-2, Addison-Wesley-Verlag
26. Brodal GS (1996) Worst-Case Efficient Priority Queues, Proceedings of the seventh annual ACM-SIAM Symposium on Discrete Algorithms, pp. 52–58
27. Okasaki C (1996) Purely Functional Data Structures, Doktorarbeit, Carnegie Mellon University
28. Okasaki C (1995), Purely Functional Random-Access Lists, Functional Programming Languages and Computer Architecture, pp. 86–95
29. Broder A, Stolfi J (1986) Pessimal Algorithms and Simplexity Analysis, ACM SIGACT News
30. Barendregt HP (1985) The Lambda Calculus, Its Syntax and Semantics, North Holland, Revised edition
31. Gardner M (1985) Magic numbers of Dr. Matrix, Prometheus Books
32. Tarjan RE (1985) Amortized computational complexity, SIAM Journal on Algebraic and Discrete Methods, Vol. 6, No. 2, pp. 306–318
33. Church A (1985) The calculi of lambda-conversion, Princeton University Press
34. Bayer R (1972) Symmetric binary B-Trees: Data structure and maintenance algorithms, Acta Informatica. 1, pp. 290–306
35. Adelson-Velsky GM, Landis EM (1962) An algorithm for the organization of information (Englische Übersetzung aus Doklady Akademii Nauk SSSR 146), Soviet Math. Doklady. 3, pp. 1259–1263
36. McCarthy J (1960) Recursive functions of symbolic expressions and their computation by machine, Part I, Communications of the ACM archive, Vol. 3, No. 4, pp. 184–195
37. Church A, Rosser JB (1936) Some properties of conversion, Transactions of the American Mathematical Society, Vol. 39, No. 3, pp. 472–482
38. Webseite des Euler-Projekts: http://projecteuler.net/
39. Webseite zu GHC: http://www.haskell.org/ghc
40. Webseite zur Funktionenübersicht: http://www.haskell.org/ghc/docs/latest/html/libraries
41. Webseite zu Opal: http://user.cs.tu-berlin.de/~opal/
42. Webseite zu Meta Language: http://www.smlnj.org/
43. Webseite zu Gofer: http://web.cecs.pdx.edu/~mpj/goferarc/index.html
44. Webseite zu Miranda: http://miranda.org.uk/
45. Webseite zu Scala: http://www.scala-lang.org/
46. Webseite zu NotePad++: http://notepad-plus.sourceforge.net/de/site.htm
47. Webseite zu TextWrangler: http://www.barebones.com/products/textwrangler/
48. Webseite zum Kate-Editor: http://kate-editor.org/
49. Webseite der Helium-IDE: http://www.cs.uu.nl/wiki/Helium
50. Wikibook zur Datenkompression: http://de.wikibooks.org/wiki/Datenkompression
51. Haskell-Funktionen: http://www.zvon.org/other/haskell/Outputglobal/index.html
52. Webseite Haskellprojekte: http://hackage.haskell.org/packages/archive/pkg-list.html
53. Projektwebseite Frag: http://haskell.org/haskellwiki/Frag
54. Projektwebseite Monadius: http://www.geocities.jp/takascience/haskell/monadius_en.html
55. Haskell-Suchmaschine Hoogle: http://www.haskell.org/hoogle/
56. Dokumentation von Hugs: http://cvs.haskell.org/Hugs/pages/documentation.htm
57. Bibliotheken von GHC: http://www.haskell.org/ghc/docs/latest/html/libraries/
58. Standardklassen von Haskell: http://www.haskell.org/onlinereport/basic.html#standard-classes
59. Monaden: http://www.haskell.org/onlinereport/exps.html#sect3.14
60. Steuerzeichen: http://www.haskell.org/onlinereport/lexemes.html#sect2.6
61. Übersicht zu Monaden: http://www.haskell.org/all_about_monads/html/introII.html
62. Buffering-Bug: http://hackage.haskell.org/trac/ghc/ticket/2189
63. Webseite Isabell: http://www.cl.cam.ac.uk/research/hvg/Isabelle/
64. Webseite Coq: http://coq.inria.fr/
65. Webseite QuickCheck: http://www.cs.chalmers.se/~rjmh/QuickCheck/manual.html

66. Wikipedia: http://www.wikipedia.com
67. Wikipedia Bärtierchen: http://de.wikipedia.org/wiki/B%C3%A4rtierchen
68. Wikipedia ASCII-Tabelle:
http://de.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange
69. Webseite zum Unicode: http://www.unicode.org/
70. Webseite zu verschiedenen Fakultätsfunktionen:
http://www.willamette.edu/~fruehr/haskell/evolution.html
71. Blog zu Monaden:
http://blog.sigfpe.com/2006/08/you-could-have-invented-monads-and.html
72. Webseite zu Parsec: http://hackage.haskell.org/package/parsec
73. Webseite zu den Alligatoren: http://worrydream.com/AlligatorEggs/
74. Webseite A Neighborhood of Infinity:
http://blog.sigfpe.com/2006/08/you-could-have-invented-monads-and.html

Sachverzeichnis