PHILIPPS-UNIVERSITÄT MARBURG
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

Übungen zur „Praktischen Informatik III“, WS 2006/07

Nr. 1, Abgabe: 24. Oktober 2006 vor der Vorlesung

Hinweise: Die Lösungen sollten grundsätzlich schriftlich, Programme zusätzlich auf Diskette oder per E-Mail an Ihren Tutor oder Ihre Tutorin abgegeben werden. Die Abgabe ist in Gruppen bis zu zwei Personen erlaubt.

---

1. Primzahltest — 6 Punkte

   Sei $n > 1$ eine natürliche Zahl. $ld(n)$ bezeichne die kleinste natürliche Zahl $k > 1$, die n (ohne Rest) teilt, d.h. n ‘mod‘ $k == 0$.

   (a) Zeigen Sie mit einem Widerspruchsbeweis: Für $n > 1$ ist $ld(n)$ Primzahl. / 1

   (b) Beweisen Sie direkt: Falls $n > 1$ und n ist keine Primzahl, so gilt: $ld(n)^2 \leq n$. / 2

   (c) Definieren Sie eine Funktion `ld :: Int -> Int`, die zu einer Zahl $n > 1$ die Zahl $ld(n)$ berechnet. / 2
   Hinweis: Definieren Sie eine Hilfsfunktion `ldf :: Int -> Int -> Int`, die zu n und k mit $n \geq k$ den kleinsten Teiler l von n mit $l \geq k$ bestimmt.

   (d) Schreiben Sie eine Funktion `is_prime :: Int -> Bool`, die testet, ob ihr Argument eine Primzahl ist. / 1

2. Bildtransformationen — 6 Punkte

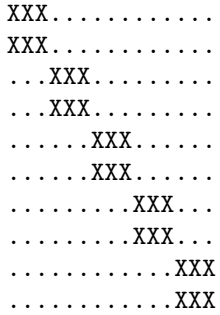
   Das Modul `Picture` in der auf der Vorlesungsseite bereitgestellten Datei `picture.hs` stellt folgende Funktionen für einfache Bildtransformationen zur Verfügung:

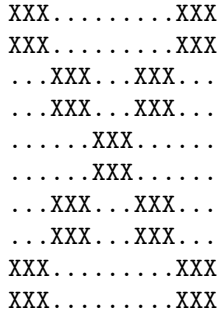
```
module Picture where type Picture = ...

printPic :: Picture -> IO()    -- Bildschirmausgabe von Bildern
flipH, flipV, flipD :: Picture -> Picture
                               -- Spiegelung horizontal, vertikal, diagonal
invertColour  :: Picture -> Picture -- Farbinversion
superimpose, above, aside  :: Picture -> Picture -> Picture
                               -- Bildanordnung ueber-, unter-, nebeneinander
white, lambda :: Picture       -- weisses Rechteck und Lambda
```

   Durch Angabe von `import Picture` am Beginn Ihres Programms können Sie das Modul importieren und die Funktionen verwenden.

   (a) Definieren Sie eine Funktion `picCol :: Int -> Picture -> Picture`, die ein Bild so oft untereinander anordnet wie durch den ersten Parameter angegeben. / 2

   (b) Definieren Sie Funktionen `diag` und `cross` mit dem Typ `Int -> Picture`, die zu einer ganzen Zahl n eine $n \times n$ Matrix mit weißen und schwarzen Rechtecken bilden, in der die schwarzen Rechtecke diagonal von links oben nach rechts unten angeordnet sind (links) bzw. in der die schwarzen Rechtecke ein Kreuz bilden, das diagonal entgegengesetzte Ecken verbindet (rechts). / 4





Vordefinierte Typen, Typkonstruktoren und Datenkonstruktoren

Typkonstruktor	Datenkonstruktor(en)	Beschreibung
`Int`	0, 1, 2, ...	ganze Zahlen
`Float`	0.0 ...	Gleitkommazahlen
`Char`	' ', '!', ..., 'A', 'B', ...	ASCII Zeichen
`Bool`	`True, False`	Wahrheitswerte
`[ _ ]`	`[ ], [ el`$_1$`, ..., el`$_n$`]`	Listen
	`[ ], _ : _`	Listenkonstruktoren
`(_, ..., _)`	`(_, ..., _)`	Paare, Tupel
`_ -> _`	(entfällt)	Funktionen

Für Konstanten des Typs `[Char]` kann anstelle von `['s', 't', 'r', 'i', 'n', 'g']` auch die Schreibweise `"string"` benutzt werden. Der Typkonstruktor `->` assoziiert nach rechts, d.h. `a -> b -> c` ist gleichbedeutend zu `a -> (b -> c)`.

Vordefinierte Funktionen

Funktion	Typ	Beschreibung
`not`	`Bool -> Bool`	logische Negation
`ord`	`Char -> Int`	Codenummer im ASCII Zeichensatz
`chr`	`Int -> Char`	invers zu ord
`show`	`Text a => a -> String`	Umwandlung von Druckbarem in String
`error`	`String -> alpha`	Erzeugt einen Laufzeitfehler
`fst`	`(a, b) -> a`	erste Komponente eines Paares
`snd`	`(a, b) -> b`	zweite Komponente eines Paares
	arithmetische Operationen	
`abs`	`(Num a, Ord a) => a -> a`	Absolutbetrag
`acos`	`Float -> Float`	Arcus Cosinus
`asin`	`Float -> Float`	Arcus Sinus
`atan`	`Float -> Float`	Arcus Tangens
`atan2`	`Float -> Float -> Float`	`atan2 y x`: Arcus Tangens von `y/x`
`cos`	`Float -> Float`	Cosinus
`exp`	`Float -> Float`	Exponentialfunktion
`log`	`Float -> Float`	Logarithmus zur Basis e
`log10`	`Float -> Float`	Logarithmus zur Basis 10
`negate`	`Num a => a -> a`	Vorzeichenwechsel
`pi`	`Float`	die Zahl π
`signum`	`(Num a, Ord a) => a -> Int`	Vorzeichen
`sin`	`Float -> Float`	Sinus
`sqrt`	`Float -> Float`	Quadratwurzel

Vordefinierte Operatoren

In der folgenden Tabelle der vordefinierten Infixoperatoren geht die Assoziativität der Operatoren aus der Spalte „A" hervor. Ein r steht für rechtsassoziativ, n für nicht-assoziativ und l für linksassoziativ.

Priorität	A	Operator	Beschreibung, Typ
stark	l	`_ _`	Funktionsanwendung
	r	`if, let`, Lambda (\), `case`	(nach links)
	r	`case`	(nach rechts)
Infix 9	r	`.`	Funktionskomposition
	l	`!!`	Zugriff auf Listenelement
Infix 8	r	`^`	Exponentiation
Infix 7	l	`*`	Multiplikation
	n	`/, 'div', quot`	Division
	n	`'mod', 'rem'`	Rest
Infix 6	l	`+, -`	Addition, Subtraktion
Infix 5	r	`++`	Listenverkettung, `[a] -> [a] -> [a]`
	r	`:`	Listenkonstruktion, `a -> [a] -> [a]`
	n	`\\`	Listendifferenz, `[a] -> [a] -> [a]`
Infix 4	l	`'elem', 'notElem'`	Elementtest
			`Eq a => a -> [a] -> Bool`
	n	`/=, ==`	(Un-) Gleichheit
			`Eq a => a -> a -> Bool`
	n	`<, <=, >, >=`	Vergleichsoperationen
			`Ord a => a -> a -> Bool`
Infix 3	r	`&&`	logisches Und, `Bool -> Bool -> Bool`
Infix 2	r	`||`	logisches Oder, `Bool -> Bool -> Bool`
Infix 1			
Infix 0			
schwach	r	`->`	Funktionstypen
	—	`=>`	Kontexte
	—	`::`	Typeinschränkungen
	r	`if, let`, Lambda (\)	(nach rechts)
	—	`..`	Sequenzen
	—	`<-`	Generatoren
	—	`(_, ..., _)`	Tupelkonstruktion
	—	`|`	Bedingungen, Wachen
	—	`->`	Alternativen (`case`)
	—	`=`	Definitionen
	—	`;`	Trennung von Deklarationen