WS 2017/2018

Funktionale Programmierung

2. Übungsblatt

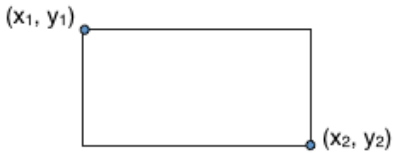
Prof. Dr. Margarita Esponda

1. Aufgabe (7 Punkte)

Nehmen Sie an, wir haben folgende Datentyp-Synonyme definiert:

```
type Point  = (Double, Double)
type Rect = (Point, Point)
```

Ein Rechteck des Datentyps "Rect" ist immer parallel zum Koordinatensystem und wird durch die Koordinaten der Ecken oben links und unten rechts $((x_1,y_1),(x_2,y_2))$ definiert.



Definiere Sie unter Verwendung des **Rect**-Datentyps folgende Funktionen:

```
area :: Rect -> Double
contain :: Rect -> Rect -> Bool   -- Testet, ob eines der Rechtecke das
                                     andere Rechteck beinhaltet.
overlap :: Rect -> Rect -> Bool   -- Testet, ob zwei Rechtecke sich überlappen.
```

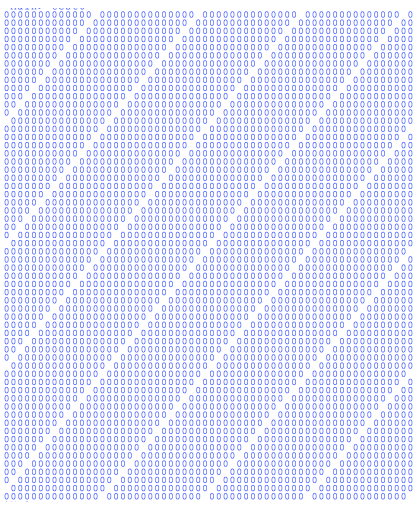
Anwendungsbeispiele:

```
area ((1.5, 3.0), (5.0, 7.0))  =>  14.0
contain ((1.0,1.0), (8.0, 8.0)) ((2.0, 2.0), (7.0, 7.0))  =>  True
overlap ((4.0,8.0), (6.0, -4.0)) ((1.0, 4.0), (8.0, 0.0))  => True
```

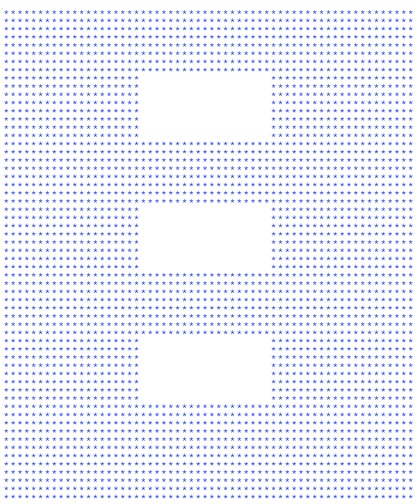
2. Aufgabe (12 Punkte)

In dieser Aufgabe sollen vier der folgenden 6 Funktionen definiert werden, die die unten stehenden Zeichenbilder ausgeben.

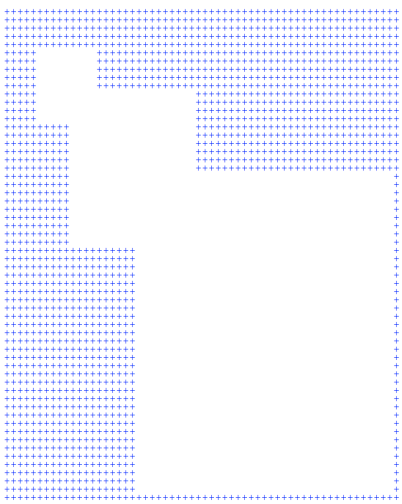
diags :: (Int, Int, Int) -> Char



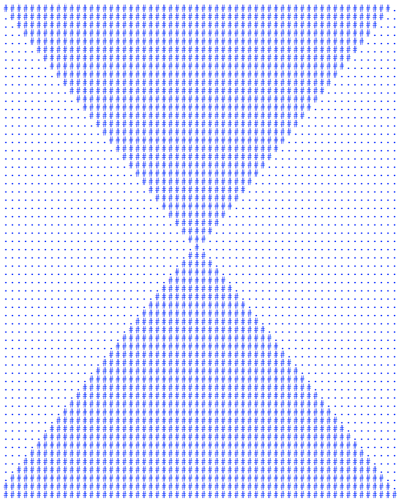
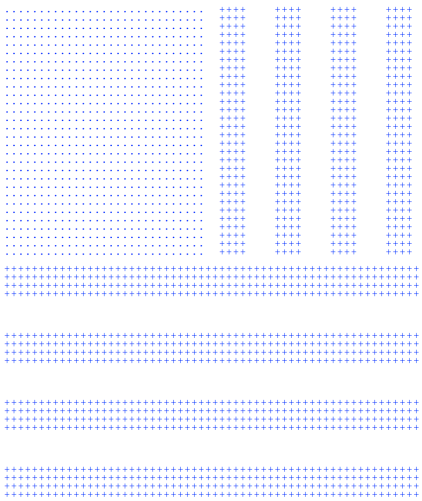
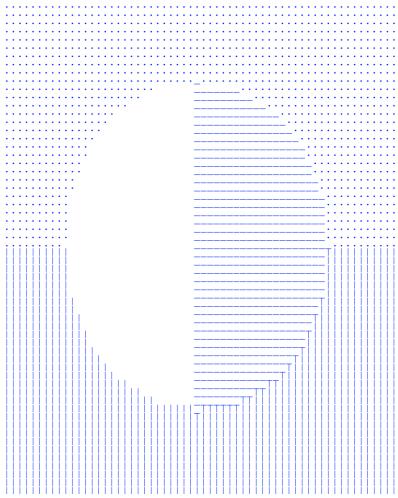
squares :: (Int, Int, Int) -> Char



squares2 :: (Int, Int, Int) -> Char



triangles :: (Int, Int, Int) -> Char **flag** :: (Int, Int, Int) -> Char **egg** :: (Int, Int, Int) -> Char

Eine **paintChars** Funktion, so wie drei Beispielsfunktionen sind aus der Veranstaltungsseite (siehe auf der KVV-Veranstaltungsseite unter **Ressourcen** -> **Material**) herunter zu laden.

Diese **paintChars** Funktion darf nicht verändert werden.

Die vier zu implementierenden Funktionen bekommen jeweils als Argumente ein (**x, y**, **size**) Tupel, das aus **x, y** Koordinaten innerhalb eines Bildes und **size**, der Seitenlänge des Bildes, besteht, und entscheidet, welches Zeichen an dieser bestimmten **x**, **y** Position geschrieben wird.

Innerhalb der zu implementierenden Funktionen darf keine Rekursion verwendet werden.

Vier **Bonuspunkte** können erworben werden, wenn alle sechs Funktionen definiert werden (zwei pro Zusatzfunktion).

3. Aufgabe (4 Punkte)
Eine Annäherung der Zahl π kann mit Hilfe folgender Reihe wie folgt berechnet werden:

$$R_n = 4 \cdot \sum_{k=0}^{n} \frac{(-1)^k}{2k+1} = 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots\right) = \pi$$

Definieren Sie eine Funktion **pi_leibniz**, die die Zahl π bei Eingabe von **n** annähert.

Anwendungsbeispiel:

pi_leibniz 3000000 => 3.141595986911832 (ungefähr)

4. Aufgabe (4 Punkte)
Schreiben Sie eine rekursive Funktion, die bei Eingabe einer positiven ganzen Zahl **n** alle Zahlen zwischen **1** und **n** aufsummiert, die durch zwei vorgegebene Zahlen teilbar sind.

Anwendungsbeispiel:

sum_mults 2 5 100 => 550

5. Aufgabe (2 Punkte)

Betrachten Sie folgende Funktionsdefinition.

```
collatz_Nums :: Integer -> [Integer]
collatz_Nums    1     =  [1]
collatz_Nums  (n+1)  =  (n+1): collatz_Nums (next (n+1))
                        where
                        next  n  | mod n 2 == 0  =  div n 2
                                 | otherwise     =  3*n + 1
```

Reduzieren Sie den folgenden Ausdruck, indem Sie alle einzelnen Schritte bis zur Normalform aufschreiben.

```
collatz_Nums  3
```

6. Aufgabe (3 Punkte)

Definieren Sie eine **deleteSpaces** Funktion, die alle Leerzeichen eines Textes löscht.

Anwendungsbeispiel:

```
deleteSpaces  "   Hello World! "  =>  "HelloWorld!"
```

Wichtige Hinweise:

1) Verwenden Sie geeignete Namen für Ihre Variablen und Funktionsnamen, die den semantischen Inhalt der Variablen oder die Semantik der Funktionen wiedergeben.

2) Verwenden Sie vorgegebene Funktionsnamen, falls diese angegeben werden.

3) Kommentieren Sie Ihre Programme.

4) Verwenden Sie geeignete lokale Funktionen und Hilfsfunktionen in Ihren Funktionsdefinitionen.

5) Schreiben Sie in alle Funktionen die entsprechende Signatur.