WS 2018/2019 Funktionale Programmierung
4. Übungsblatt
M. Esponda/R. Rojas/O. Wiese
Abgabe: 26.11.2018 10:00

1. Aufgabe (2 Punkte)

In der Vorlesung wurde das Exklusive-Oder (xor) mit Hilfe von der Konjunktion, Disjunktion und Negation dargestellt. Definieren Sie nun die Funktion xor in Haskell mit Hilfe von Pattern matching.

2. Aufgabe (8 Punkte)

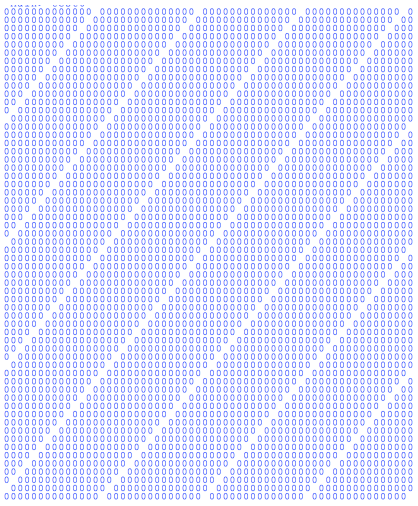
Lokale Definitionen sind oft hilfreich und nützlich. Denn die Namen können sich so gegenseitig überschatten. Betrachten Sie folgende Funktionen und betrachten sie die Sichtbarkeit aller eingeführten Namen. Machen Sie dazu kenntlich wo diese definiert werden (umkreisen) und wo diese verwendet werden (unterstreichen). Machen Sie deutlich in dem sie unterschiedliche Farben verwenden oder durch Pfeile von der Verwendung zur Definition, welche Variable andere überschatten.

```
f1 :: Int -> Int -> Int -> Int
f1 a b c = h a b - h a c - h b c
  where h b a = a + b

f2 :: Double -> Double
f2 x = let g x = sqrt(4 * x)
       in (g x)^2 + (g 3)^2

f3 :: Int -> Int -> Int
f3 x y = let g x z = h x y + h z x
                 where h x y = x - y
         in g x y + g y x
```

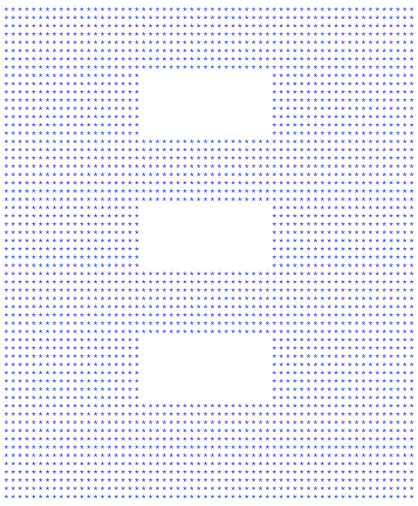
3. Aufgabe (16 Punkte)

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

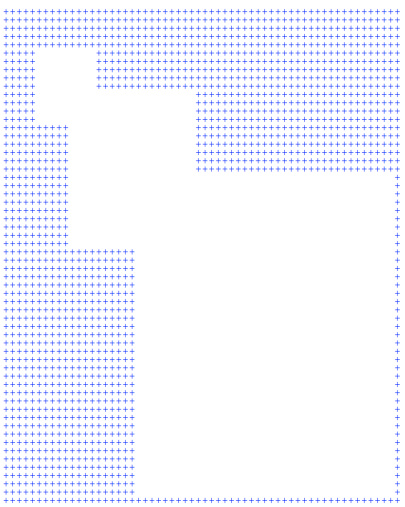
diags :: (Int, Int, Int) -> Char



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



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

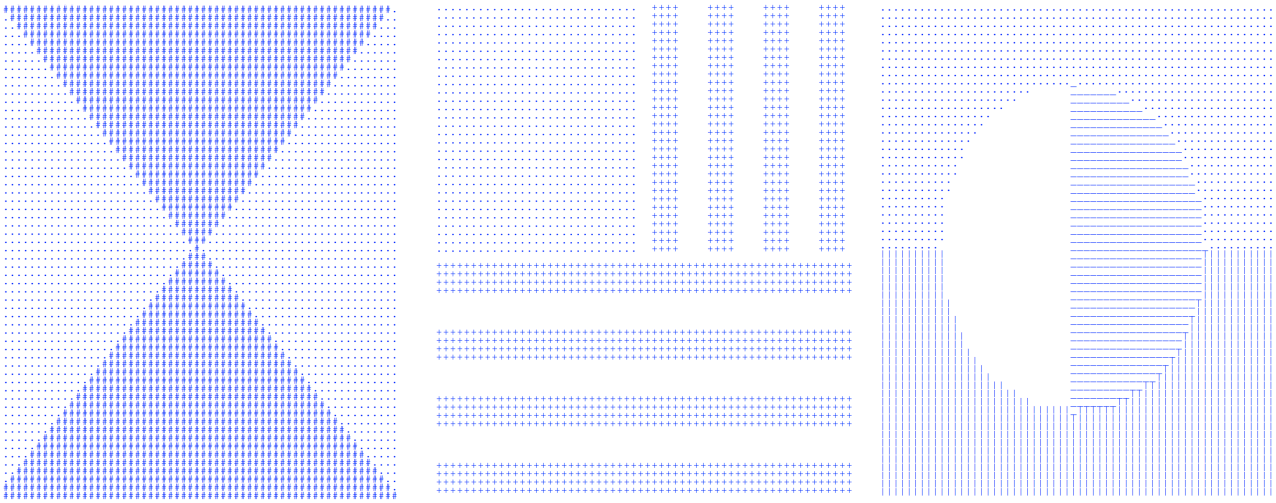


triangles :: (Int, Int, Int) -> Char

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).

4. Aufgabe (8 Punkte)

Definieren Sie die rekursive Funktion summe_aller_Primzahlen, welche die Summe alle Primzahlen zwischen den Zahlen von k bis n berechnet.

Beispiel: summe_aller_Primzahlen 0 10 liefert das Ergebnis 17 (= 2 + 3 + 5 + 7).

Hinweise:
Beachten Sie den Fall k > n. Sie dürfen die Funktion zum Primzahltest aus der Vorlesung verwenden.

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.