WS 2020/2021

Funktionale Programmierung

1. Übungsblatt

Prof. Dr. Margarita Esponda

1. Aufgabe (2 Punkte)

Betrachten Sie folgende Haskell-Funktionsdefinition, die überprüfen soll, ob die eingegebene Zahl **n** eine ungerade Zahl ist:

```
ungerade :: Integer -> Bool
ungerade n = rem n 2 == 1
```

a) Warum ist diese Funktionsdefinition inkorrekt?
b) Geben Sie eine korrekte Funktionsdefinition.

Lösung a): Das Ergebnis ist mit ungerade negative Zahlen falsch,
weil die rem-Funktion negative Werte, die ungleich 1 sind, berechnet.

Lösung b):

```
ungerade :: Integer -> Bool
ungerade n = rem n 2 /= 0
```

2. Aufgabe (10 Punkte)

Nehmen Sie an, wir haben folgende Datentyp-Synonyme definiert, um Kreis-Objekte im Haskell zu modellieren:

```
type Center  = (Double, Double)  -- Mittelpunkt eines Kreises
type Radius  = Double
type Circle  = (Center, Radius)  -- Mittelpunkt und Radius eines Kreises
```

Definieren Sie, unter Verwendung dieser Datentyp-Synonyme, folgende Funktionen:

```
area :: Circle -> Double
perimeter :: Circle -> Double
equal :: Circle -> Circle -> Bool       -- Überprüft, ob die Kreise identisch sind
                                           (Position + Große)
intersect :: Circle -> Circle -> Bool   -- Testet, ob die Kreise sich überschneiden
contain :: Circle -> Circle -> Bool     -- Testet, ob der erste Kreis den zweiten
                                           Kreis enthält
```

Anwendungsbeispiele:

```
intersect  ((3.0, 3.0), 2.5) ((9.5,8.0), 2.0)  => False
contain  ((1.0, 1.0), 6.0) ((2.0,1.0), 3.0)  => True
```

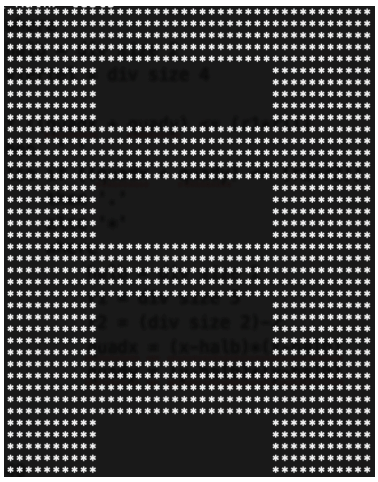
Lösung b):

```
{-- Hilfsfunktion aus der Vorlesung --}

type Point  = (Double, Double)
distance :: Point -> Point -> Double
distance (a,b) (c,d) = sqrt ((a-c)**2+(b-d)**2)

type Center = (Double, Double)
type Radius =  Double
type Circle = (Center, Double)

area :: Circle -> Double
area (center,radius) = pi*radius*radius

intersect :: Circle -> Circle -> Bool
intersect ((x1,y1),r1) ((x2,y2),r2) = (r1+r2) > (distance (x1,y1) (x2,y2))

contain :: Circle -> Circle -> Bool
contain ((x1,y1),r1) ((x2,y2),r2) = ((distance (x1,y1) (x2,y2)) + r2) < r1
```

3. Aufgabe (9 Punkte)

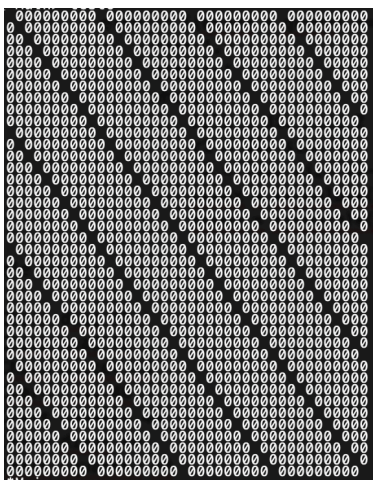
In dieser Aufgabe sollen 3 von 5 Funktionen definiert werden, die die unten stehenden Zeichenbilder-Muster erzeugen.

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



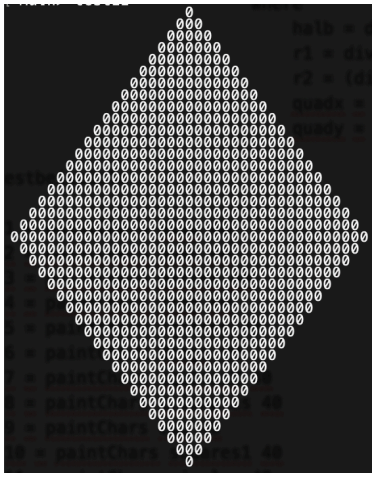
```
rectangles (x,y,size) | q1  = ' '
                      | otherwise = '*'
                     where
                       q1 = x>s4 && x<3*s4 && y4>=s8 && y<=(y4+1)*s8
                       s4 = div size 4
                       s8 = div size 8
                       y4 = mod y s4
```

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



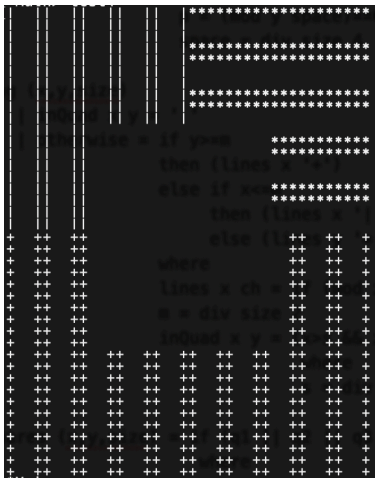
```
diags :: (Int, Int, Int) -> Char
diags (x,y,size) | isMult (x-y) = ' '
                 | otherwise    = '0'
                 where
                   space = div size 4
                   isMult m = (mod m space)==0
```

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



```
diamon :: (Int, Int, Int) -> Char
diamon (x,y,size) | (x+y)>half && (x+y)<3*half && half>(y-x) && (x-y)<half = '0'
                  | otherwise = ' '
                    where
                    half = div size 2
```

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



```
flag (x,y,size) | inQuad x y = ' '
                | y>=m       = lines x '+'
                | x<=m       = lines x '|'
                | otherwise  = lines y '*'
                  where
                  m = div size 2
                  lines x ch | (mod x 4)<2 = ch
                             | otherwise   = ' '
                  inQuad x y = (x>s && x<3*s && y>s && y<3*s)
                               where
                               s = div size 4
```

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

```
circles (x, y, size) | (quadx + quady) < (r1*r1) = ' '
                     | (quadx + quady) < (r2*r2) = '_'
                     | otherwise = 'o'
                           where
                               halb = div size 2
                               r1 = div size 3
                               r2 = (div size 2)-2
                               quadx = (x-halb)*(x-halb)
                               quady = (y-halb)*(y-halb)
```

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

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

Die drei zu implementierenden Funktionen bekommen jeweils als Argumente (**x, y)-**Koordinaten innerhalb eines Bildes und **size** (die Seitenlänge des Bildes) und entscheiden dann, welches Zeichen an die vorgegebenen **x**, **y** Position eingesetzt wird.

Innerhalb der zu implementierenden Funktionen darf keine Rekursion verwendet werden.

Vier **Bonuspunkte** können erworben werden, wenn Sie statt nur drei die fünf Funktionen definieren (zwei Bonuspunkte pro Zusatzfunktion).

4. Aufgabe (10 Punkte)

Definieren Sie eine Funktion **num2GermanWords**, die eine zweistellige ganze Zahl als Argument bekommt, und die Zahl als Wort wieder zurückgibt.

Anwendungsbeispiel: **num2GermanWords** 0 => "Null"
num2GermanWords 21 => "Einundzwanzig"
num2GermanWords (-21) => "minus Einundzwanzig"

Lösung:

```
nums2words :: Int -> [Char]
nums2words num | num<0 = "minus " ++ nums2words (abs num)
               | num == 0 = "Null"
               | num == 1 = "Eins"
               | num<13 = until12 !! num
               | num<20 = (until12 !! (mod num 10)) ++ "zehn"
               | num>=20 && num<100 = num2words' num
               | otherwise          = error "Keine zweistellige Zahl"
            where
            num2words' num | (mod num 10)==0   = (toUpper (head tens)) : tail tens
                           | otherwise = units ++ "und" ++ tens
                           where
                             units = until12 !! (mod num 10)
                             tens  = tList !! (div num 10)

            until12 = ["", "Ein", "Zwei", "Drei", "Vier", "Fünf", "Sechs",
                       "Sieben", "Acht", "Neun", "Zehn", "Eleven", "Zwolf"]
            tList = ["", "", "zwanzig", "dreizig", "vierzig", "fünfzig",
                     "sechzig", "siebzig", "achtzig", "neunzig"]
```

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) Geben Sie für alle Funktionen die entsprechende Signatur an.
6) Schreiben Sie getrennte Test-Funktionen für alle Aufgaben.
7) Die Lösungen sollen elektronisch (nur Whiteboard-Upload) abgegeben werden. **Keine verspätete Abgabe per Email ist erlaubt.**